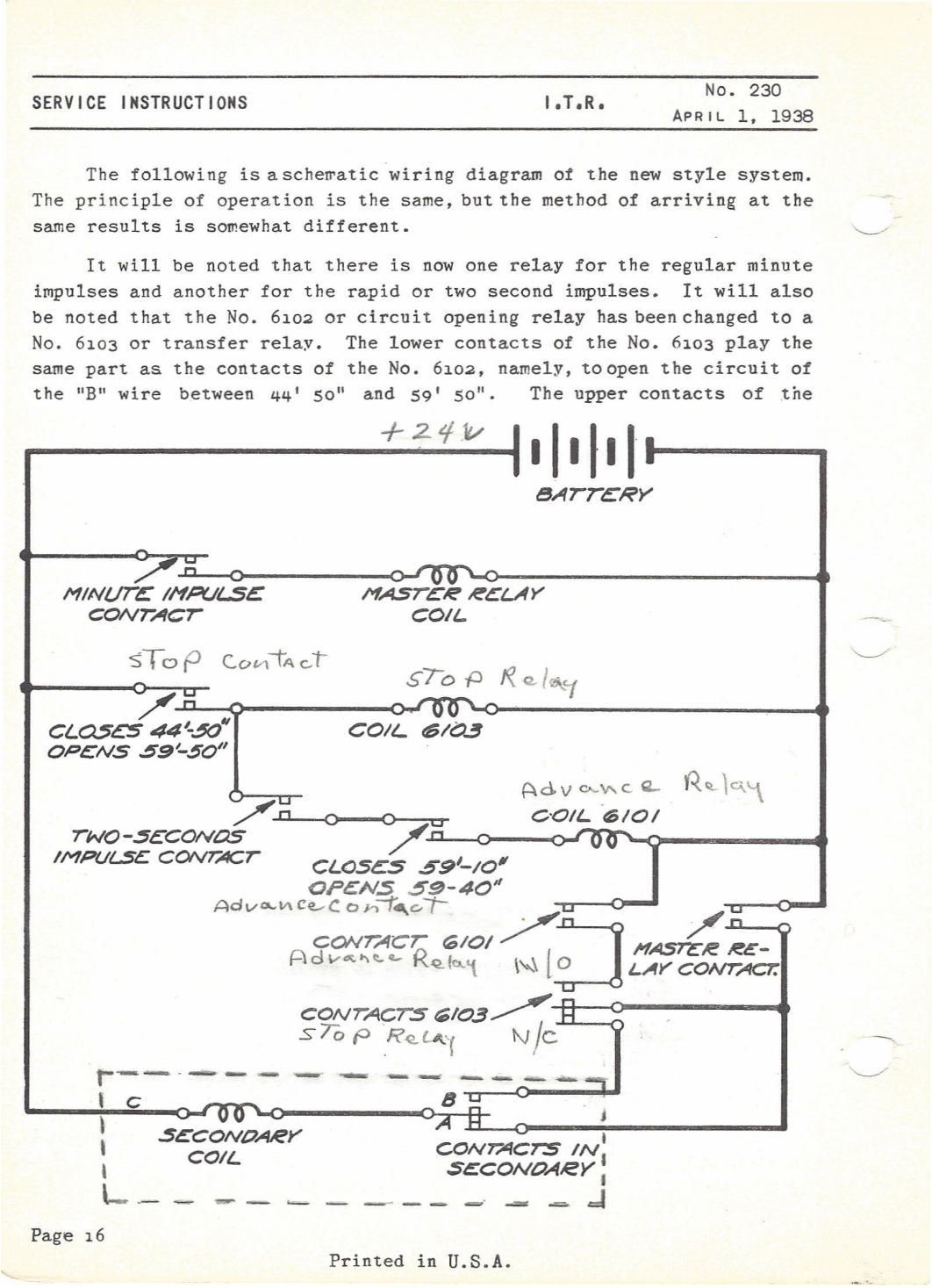
“A Computer Based Master for IBM Impulse Clocks”

By Joe Fox, KD4MS, Panama City, FL

Photo 1: ITR Impulse Secondary, from the Executive Offices, Woodlands Division, International Paper Co. Panama City Fl. Soft copper outer shell painted black. Featuring fancy hands and adorned with fifteen brass numerals, a scripted brass “International”, and sixty brass minute markers, each individually screwed to a nickle face.

Over the years I have seen many requests from people on various forums, asking if anyone has information on how to get an old IBM slave clock working as a time piece. This article describes a computerized equivalent master clock for an IBM Minute Impulse Self-Regulating clock system. So if you are lacking a functional master, build this project and get one or more IBM impulse secondary time pieces working accurately and electronically using the popular Arduino Project computer and a pair of Insulated-Gate Bipolar Transistor (IGBT) MOSFET devices. See Arduino Project information at: https://www.arduino.cc/en/Guide/Introduction The Arduino Project is an “Open Source” endeavor and “All Arduino boards are completely open-source, empowering users to build them independently and eventually adapt them to their particular needs. The software, too, is open-source, and it is growing through the contributions of users worldwide.” While the original Arduino [manufactured in Italy] and other compatible brands of the Arduino project will work just fine, I encourage you to support the Arduino Project by buying an original when possible.

These clock movements are often referenced as International Time Recording or ITR slave clocks. IBM impulse secondary clock movements, like the one in Photo 1, along with time attendance recorders, event programmers, buzzers, bells, and other time recording devices were used for decades in many schools, factories, and businesses for coordinating 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

However, a master clock is required to operate these old impulse secondary clocks and to take advantage of the three wire self correcting feature designed into this time and attendance system.

IBM also produced plain (non-correcting) minute impulse secondaries and self-correcting movements using only two wires. The latter by using diodes and polarized voltages. These two-wire designs however are not compatible with this master clock configuration but with program changes and the use of "H-bridge" drive circuits could be achieved. 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.

Technical History

IBM builds their products robust, to minimize failures. One of the reasons for a self-regulating design is the use of relays. In 1914, relay logic was high tech, state of the art. It will be 33 years before transistors are invented by the scientists at Bell Labs in 1947. So electromagnetic relays were part of the technical development of that time. Relay contacts operating with inductive loads, however have a couple of nasty drawbacks. They are miniature arc welders and their contacts produce an arc every time they make or break a circuit under load. This arcing causes vaporizing of of the contact's metal surface that eventually results in fouling and pitting. In The resulting fouled or dirty contact creates a high resistance connection which can reduce the current and voltage available to the devices they control. The arcing is created from the high energy counter electromotive force (CEMF) induced into the circuit when a coil's magnetic field (inductive load) suddenly collapses when the contacts open. There is also a counter EMF developed when contacts are closed and the magnetic field builds. By design, the contacts in the secondary's movements and most of the contacts in the master clock's movement open or close before current is applied; so there is no arcing to these contact sets. Relay contact damage caused by years of accumulative arcing and pitting, and contacts loose, dirty, or corroded can cause lost pulses and low voltages at the secondary's coils resulting in missed advances. Not to mention commercial power failures from storms and other natural disasters. Hence the self-regulating design. There is also a limit to the amount of current allowed through the master and other relay contacts. I refer you to the *Reference Manual of Electrical Characteristics* , IBM Form # 231-6401-0 for important electrical information for IBM time equipment devices, including model numbers, wire size, wire distances, voltage, current, and power requirements, as well as tables needed to calculate parameters of a complete time system installation. This same document with some newer information is also reformatted and printed as part of the *IBM*

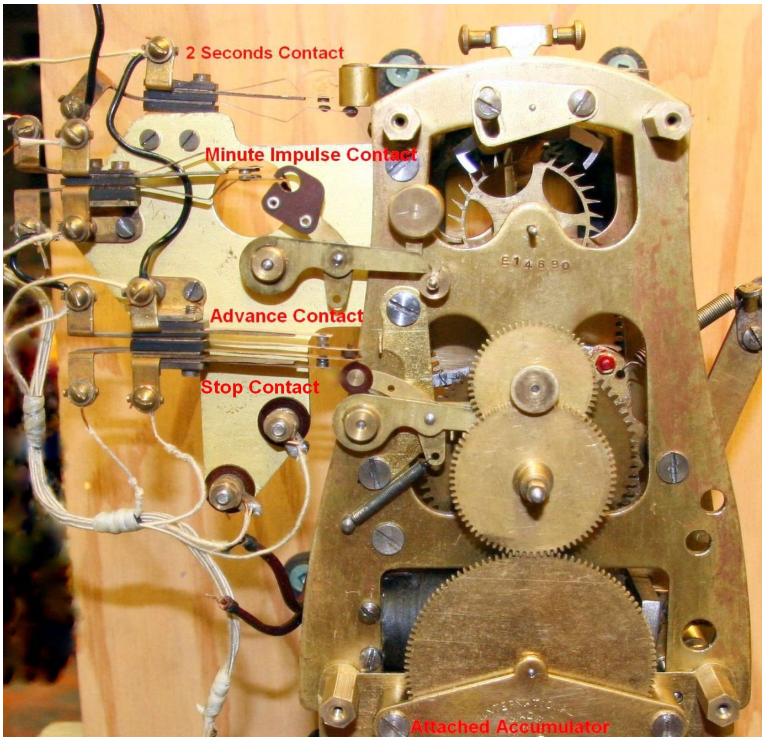


Photo 2: IBM Model 25 Master Clock movement, impulse wound, spring driven, with an accumulator attached.

Photo 3:1930 ITR Master Clock, from International Paper company, Panama City

Sales Manual, as the *Electrical Appendix*, Copyright IBM, December 1951, no IBM Form number. Please request an electronic copy of the reference manual from IBM Corporate Archives, Somers, New York, 10589. According to the *Reference Manual of Electrical Characteristics*, there are specific current limits to the IBM impulse system's relay contacts and rectifiers. For instance a relay type 6121 used in a 24 volt circuit is limited to a maximum of two (2) amps. So only a finite number of devices totaling 2 amps can be attached to a type 6121 relay circuit. Since the model 561-2 (@ 24 V) secondary requires 0.024 amps each, a maximum total of 83 units of these secondaries could be attached to a type 6121 relay circuit. Reduce that number by six if a model 25 master clock winding magnet (@ 0.132 amps) is also attached. In Table II, "Amperes Required By DC Impulse Magnets", page 8, for a list of electrical current requirements of twenty six models using coil voltages of 12, 24, 48,115, or 230 volts DC. An IRFZ44 IGBT transistor can control about 9 amperes of current with 5 volts at its gate, so you can see the advantage of the IGBT transistor over a relay. Discounting the size of the relays (minimum of three) and the space they take up, there are no arcing, pitted or dirty contacts, or corrosion problems. There is virtually no current required to drive the IGBT gate, so multiple IGBT gates can be tied together for multiple nine amp circuits using only one Arduino output. The IGBT transistors used here are only rated to 60 volts DC, between the Source and Drain, so the 115 and 230 volt DC coils will not work with the IGBT device used here. However there are higher voltage IGBT's available. Contact limits are not the only reason for the electrical current restrictions. The earlier copper oxide rectifier IBM produced to convert AC to DC, is rated for intermittent use at 2 amperes, 24 volts, DC, and have a continuous-duty rating of only 0.5 amperes. The later selenium rectifiers were better but not by much, with a rating for intermittent use at 3.5 amps @ 24 volts DC and its continuous-duty rating is 1 amperes. Both of these types of rectifier suffer from relatively low breakdown voltages and are heat sensitive. After silicon diodes were invented in the 1940's, and became mainstream, these older rectifier types were phased out and replaced with stud mounted silicon diode rectifiers.

IBM (formally International Time Recording Co.) produced these master clocks and impulse secondaries from about 1914 to 1958, in the United States. IBM sold its U.S. Time Equipment Division to Simplex, effective December 31, 1958. However, IBM World Trade, a subsidiary of IBM U.S., continued producing IBM time systems and components for a number of years after 1958, at least until 1970. I serviced these clock systems still on IBM maintenance back in 1968. In 1966, I purchased the IBM mahogany cased master clock, shown in Photo 3, from a fellow IBMer who literally saved it from an axe; the destruction method that was standard practice for traded-in clock equipment. Probably the reason the contact assembly was removed was so it could no longer be used as a master and be in competition with IBM. You can see original wiring and the contact assembly of a master I repaired for a friend in Photo 2. My master clock was originally installed in 1930, at International Paper Company's Southern Kraft mill in Panama City, Florida. Because its contact and wiring assembly had been removed, it can no longer control secondaries. Two of my most cherished secondaries, Photos 1 and 4, came from the same paper mill and another, Photo 5, from Cove Elementary School where I attended the second through sixth grades. Since the day I acquired my IBM Master clock I've wanted my own clock system. I attempted making my own master clock contact assembly using an LED/photo transistor device but that project ended in failure. I eventually procrastinated about 46 years before finally building a working master; albeit a computerized version of a master. Of course the Arduino Mega 2560 I received as a Christmas gift helped motivate me. Over time, these electronic masters (Photos 8 and 10) will keep better time than the brass, nickle, and mercury master clock living in my foyer. Even better, you can get your secondary or program device working without spending a thousand dollars on Ebay, buying a master. This little computer now drives all seven of my IBM secondaries, each happily clicking away in my garage and displaying the exact same correct time.

Photo 4: Installed in 1930 in the Engineering Department, International Paper Co. Panama City, FL . ITR logo, soft copper shell painted Black, face painted White, with a flat glass cover.

Photo 5: Installed in February, 1955, at Cove Elementary School, 2nd grade classroom, Panama City, FL. IBM logo on a steel face painted white, steel back plate and covered with a domed glass.

Either the Arduino Uno or Arduino Mega 2560 model can be used for this master clock and this configuration requires an Ethernet shield and a DIY impulse motor driver shield.

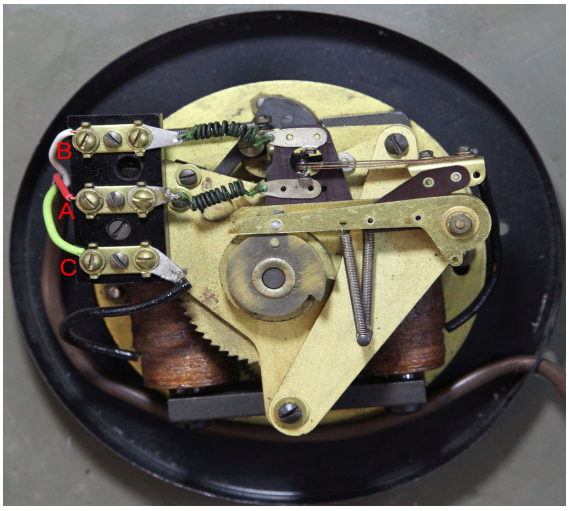


Photo 6: 1930 model 561-2, back of secondary movement of photo 4. Showing the terminals for connecting the A, B, and C system wires and the cam operated N/C and N/O contacts. Part of the black rotating armature is visible behind the "B" N/O contact.

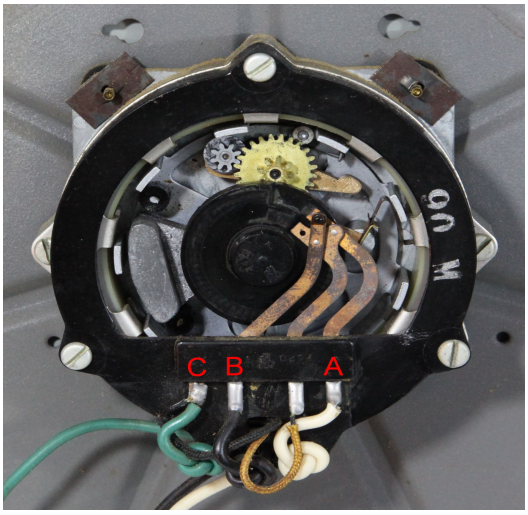


Photo 7: 1955 Model 565-2, back of secondary movement of Photo 5. Note the rotating armature appearing as the lighter white dashes inside the (black) circular coil. The darker dashes are the electro-magnetic poles of the coil. The center black disk is the "A" and "B" contact cam.

Self-Regulating Operation:

The Arduino based version of the IBM Minute Impulse Self-Regulating clock system is programmed to output two signals. 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. Secondaries that are fast (due to mechanical or electrical failure) will receive the A pulse once per minute until their 59th minute. Then they 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 signal.

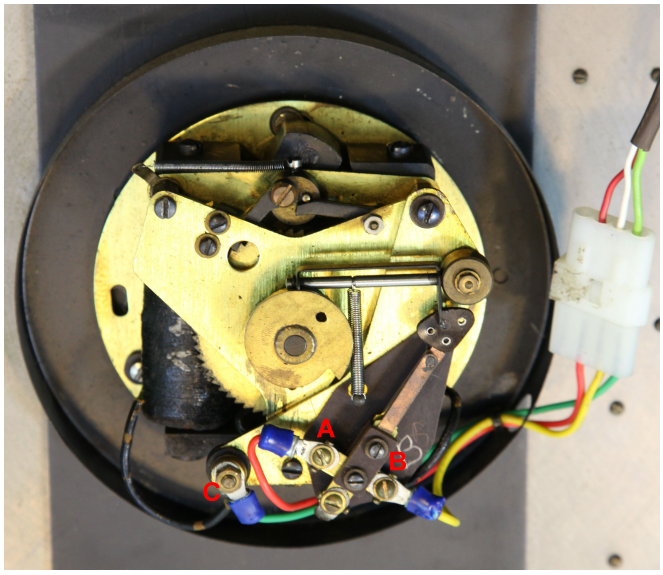


Photo 9: 1930 Model 561-2, back of photo 1, from Woodlands Div, International Paper Company. Note part of the black rotating armature visible behind the drive paw hub. The dark dots on the right side are the brass screw heads holding the brass numerals to the face.

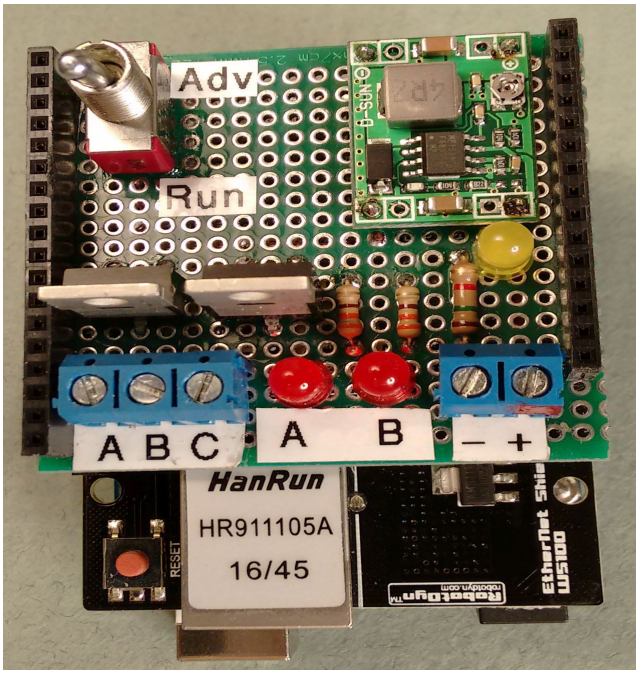


Photo 8: Stacked, Uno (bottom), Ethernet shield (middle), and DIY driver shield (top).

This secondary will switch back to the "A" contact around four minutes after the hour, depending on the model of the secondary. The B signal is raised once per minute at zero-seconds for each minute except for minutes 50 to 59. Secondaries that are late (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 the late secondary reaches its 59th minute, The secondary then switches its contacts to the "B" contact and waits for the next B pulse. The B pulse is **Stopped** at 50 minutes past the hour and only started again at zero-seconds, to step all the secondaries forward from their 59th minute to the top of the hour. The B pulse continues to advance the secondaries until the secondaries switch their cam operated contact back to the N/C "A" contact. The RUN/ADVANCE switch is used to manually advance all secondaries forward at a one second pulse rate (without correction) by pulsing both A and B lines while in the Advance position. A handy feature found on the original master clocks, and is used for maintenance, long power outages, and daylight savings time changes.

Electronic Operation:

The 5 volt logic levels of the Arduino allows the use of a common and cheap high current IGBT transistor whose gate is turned on by the Arduino's output pin. (See Schematic 2) Here is how the driver shield interfaces the secondaries. The Arduino outputs a +5 volt logic level at output pin "D9". This powers and illuminates the LED and presents a +5 volt EMF field at the gate of the IGBT.

Like a switch, this voltage field turns ON, or opens the gate of the IGBT, causing the source to drain junction to become almost a short circuit (0.046 ohms), effectively grounding the A line. Conventional current then flows from the +24 volt power supply, through line C, through the secondary's coil,

through the (N/C) A contact, through line A, through the transistor's Drain to the transistors Source, then to ground (-24V), completing the circuit and causing 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 armature to return to it's resting position, moving the minute and hour hands forward one minute. For the fact checkers, the electron flow actually is out the -24 volt power supply terminal, in to GND (ground), in through the source, out from the Drain, through the “A” line, through the A contact, into the secondary's coil, out through line C, then into the +24 volt power terminal.

Checking out a Secondary, or other impulse device:

Using the information from the previous paragraph, you can check out an impulse secondary electrically and manually. Manually on a 561-2 model by pressing the armature horizontally with your finger, then releasing, to advance the minute hand. On the 565-2 model, rotate the armature clockwise by pushing a pole piece with your finger, then releasing, advancing the movement one minute. Looking at photo 7; 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. To advance the movement electrically, simply apply +24 volts at the C terminal wire and momentarily shorting the A terminal wire (or B if the minute hand is between the 59th minute and the fifth minute) to **ground, the -24V power adapter connection**. The C wire from the movement is the one going directly to the coil, and is usually green. Wires going to the contacts will be the A (white) and B (Black). Grounding the A wire energizes the coil and operates the rotating armature for about 70 degrees of rotation. When you remove the short to ground, the armature returns to its resting position, pulling the minute hand forward. Repeating this procedure using the A terminal wire will advance the hand until the 59th minute, then the movement will stop. Now shorting the B terminal wire to ground 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. These things are slow! That is why there is a least a .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 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.

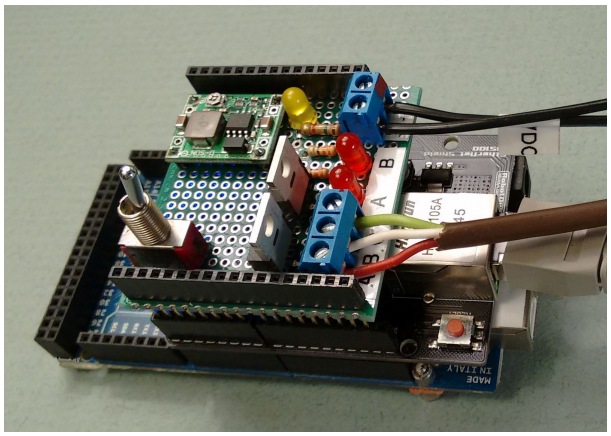


Photo 10: Stacked and wired, the Arduino Mega 2560 (bottom), Ethernet shield (middle), and DIY driver shield (top). You can see the connected Ethernet cable, the 24 volt power adapter wires, and the A, B, and C line wires going to the attached Secondaries.

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 photo 11, 12, and Table 1, below) Please label the modification so you will know what you did years later. The 12 volt secondary with the limiting resistor will consume twice the current of a 24 volt secondary but the added R1 resistor will not damage the movement or effect any other attached units.

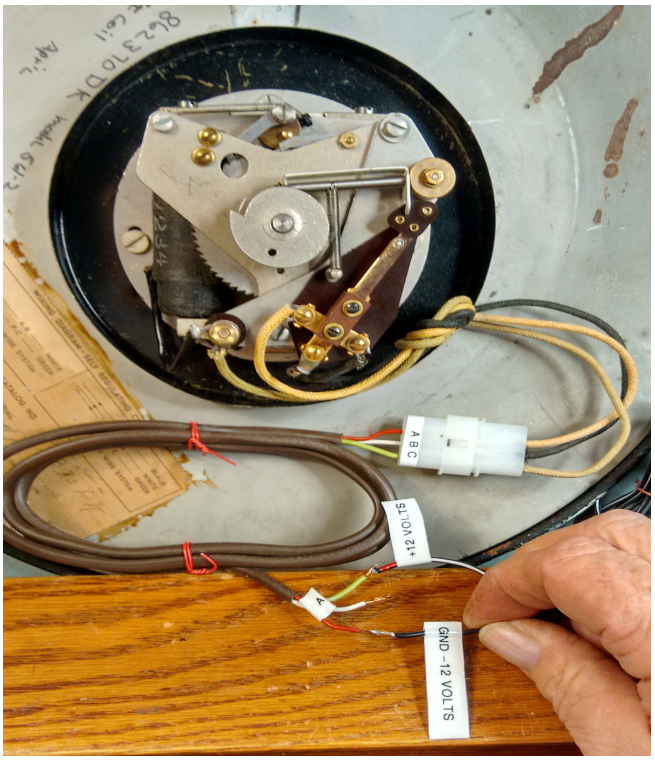


Photo 11: This 12 Volt model 561-2 secondary is energized manually with a 12 Volt DC, 1.5 amp power adapter. Armature is rotated horizontally with drive paw hub pin contacting the check paw. Tip of the drive paw can be seen through the hole in the rear plate. The gear is pulled forward after the GND wire is removed from the A wire (red).

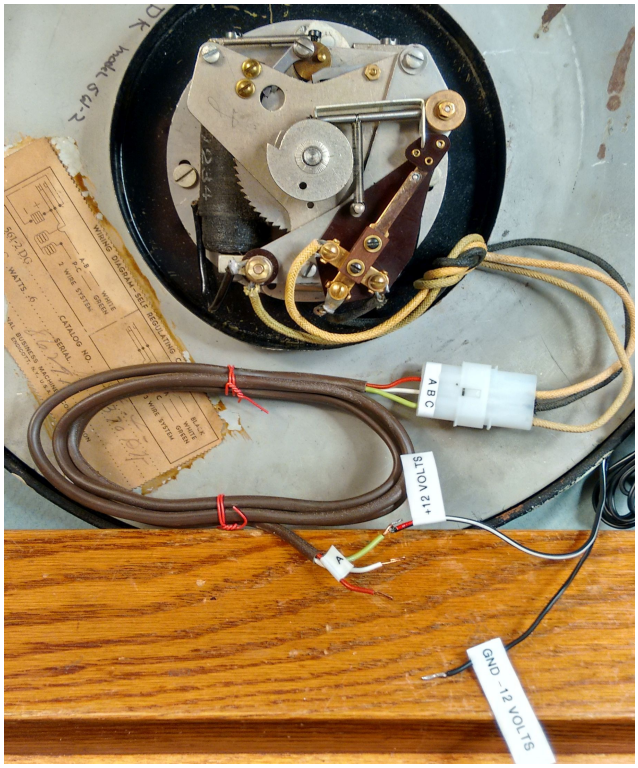


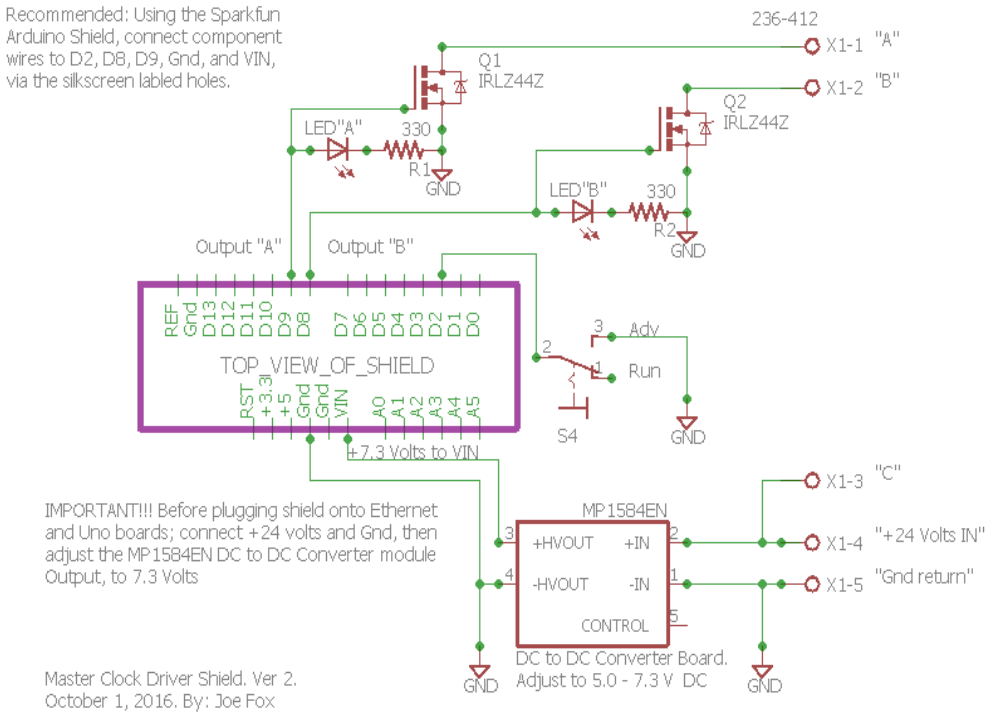
Photo 12: Secondary is shown de-energized with the armature returning to its rest position, after the GND wire is removed from the A wire. Repeating this procedure (grounding and then un-grounding) will advance the movement until it stops at the 59th minute). This 1948 model 561-2 has "12V" printed on the coil in white ink.

Just count it as two 24 volt secondaries for total current loading calculations. If all your units have 12 volt coils, just substitute the 24 volt power adapter with a 12 volt power adapter with the appropriate current rating. If you have multiple 12 volt secondaries and one 24 volt secondary then you have two options. Either add the resistor to each of the 12 volt secondaries and run the system at 24 volts. Or add two more IGBT transistor circuits, tying the "A" gates together and the "B" gates together and run a separate 24 volt circuit ("A" and "B" wires from the additional IGBT's Drains, and 24 volts from a 24 volt power source to the "C" wire of the 24 volt secondary. As you can see, this will require two power adapters, five more screw terminals and of course the additional two IGBTs. I would add the resistor (R1) to each 12 volt secondary and use 24 volts. An added benefit is that the resistor will slow the collapse of the magnetic field, therefore lowering the induced high voltage spike.

Power Adapter Revision:

This article is revision 2, so this Arduino configuration now uses a DC to DC switching power supply to convert 24 volts to 7.2 volts to power the Arduino Uno and Ethernet shield using the Arduino's onboard 7-12 volt regulator. For powering the "over all" master clock system, I recommend a computer notebook style switching power adapter, 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 secondary coils and the DC to DC buck switching power supply and attached Ethernet and Arduino computer. This eliminates the large 5 volt power supply I made and used in the earlier published article which results in a cost reduction, less heat, and a much smaller footprint. As you can see, power for the Arduino and Ethernet

boards are now from the MP1584EN, DC to DC buck boost voltage converter which comes already assembled except for connection headers pins. It is rated for a maximum of 3 amps and only cost me $2.00ea. Other models of DC to DC Buck converters are available. I connected the input of this voltage converter to the 24 Volts DC supply then adjusted the output to 7.2 volts **before** connecting and powering the Arduino and Ethernet shield.



Schematic 2: Impulse Secondary driver shield wiring. The screw terminal connectors on the right side of this schematic are connected to the wires shown on the left side of schematic 3.

Model	Description	12 Volt	24 volt	12V to 24 V Conversion Resistor
25	Master's Clock 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)
561-2	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	.170amps	.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 1: Selected model numbers: Amperes required by DC impulse magnets. To Convert a 12 Volt impulse magnet device to work on a 24 volt system, use a current limiting resistor (R1 is schematic 3). Chart shows closest standard value resistor to the computed value. Resistor values computed from R=E/I. Power in watts computed from P=E*I . Wire the resistor in series between C line and the magnet's C terminal. Although resistors are only active for 0.4 seconds per minute wattage rating must be picked in case of a computer crash leaving an output active, or a shorted A or B line failure.*

This 7.2 volt output is wired to the Arduino's 7-12 volt regulator's input labeled “Vin”. Changing the converter's input from 24 volts to 12 volts does not require readjusting the 7.2 volt output of this converter. IBM used an unregulated 24 Volt RMS power supply rectified with either copper oxide or selenium rectifiers to power the movement coils until switching to silicon rectifiers probably in the 1950s. My use of a regulated 24 V dc source is an attempt to protect the secondary's coil winding and the IGBTs from the higher peak voltage produced by unfiltered rectifiers connected to a transformer (24 volts RMS times 1.414 equals 33 volts peak voltage). This peak voltage and possible high voltage spikes generated by the collapsing magnetic fields of the coils may or may not damage the nearly 100 year old coil insulation in these old clock movements. So I feel more comfortable using a regulated DC switching power adapter for this reason. Plus they are usually current limited and short circuit protected.

Driver Shield:building notes

Building your own driver shield is not complicated. There are only five connections to the Arduino needed. Two are power (+7.2 volts) and ground. The other three are data connections, one input and two outputs.

1. Start by selecting the perforated circuit board and preparing the headers. You can buy an Arduino prototyping board for $7.00 from Arduino.com, The advantages of these boards is there is no spacing problem between I/O pins “D8” and “D9”. The Arduino has an odd spacing between its I/O data pins “D7” and “D8”. This spacing is 0.175 inches, not 0.200 inches as it should have been. I don't know if it was a stupid mistake or if it was an attempt to force you to buy the Arduino branded shields. Both Arduino models have the issue. I have only had one problem so far and that is with an intermittent connection with a recently purchased $9.00 Ethernet shield from Robot Dyn, and my driver shield data pin “D8”. Forming the pin so it is inserted straight into the shield's header below it, seems to fix the problem. I suspect the problem lies with the header used on the new Ethernet shield because all of my driver shields have worked fine or more than two years with the Sainsmart and Arduino Ethernet shields.
2. To build you own shield, you can easily find inexpensive perforated circuit boards, both phenolic or 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. Search Amazon.com for header pins too. Look for “stackable headers” available in 8, 10, 12, and 18 single row pins. The specifications 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. Note: As shown in the photos, I used a 20 pin strip trimmed to 18, then removed the pin between positions “D7” and “D8”, leaving 17 pins. But two 8 pin headers will work fine since you are only connecting to three of the pins on this side of the shield. This missing pin also serves as a visual and physical key when connecting the driver board to the Ethernet board. 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. (I used a strip of 15, removing a pin between positions “A0” and “5V”. 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 as detailed above offers great stability, holding the driver shield onto the Ethernet shield firmly.

Refer to the impulse secondary driver circuit as shown in Schematic 2. I place my components onto the board, planning where common connections will fall and where components will fit and look nice. Solder four [single] male header pins to the MP1584EN DC to DC converter. One in each corner. For the DIY driver shield, I usually run the ground buss with bare wire first, then the wires to the DC to DC Buck power converter. Verifying my connections a couple of times.

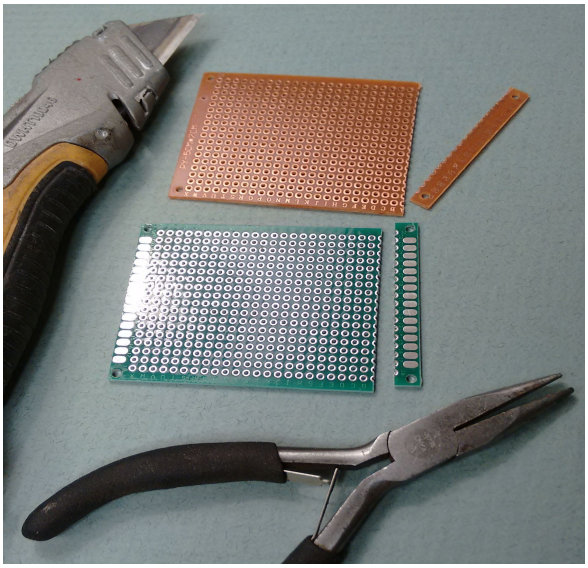


Photo 13: Score along the holes a couple of times, both sides, then snap off the excess with pliers. You need a total of 20 holes showing for the headers to align with the Arduino headers. I found one offer on Amazon for 25 fiberglass boards with plated through holes that needs no trimming, for $7.54, w/prime shipping.

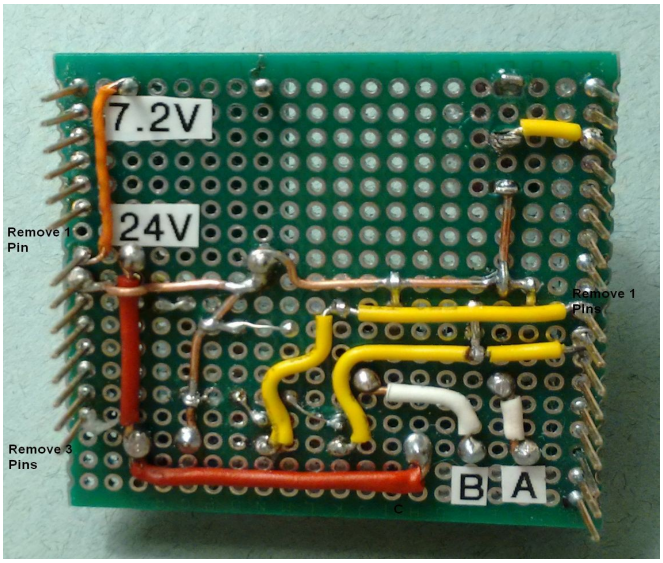
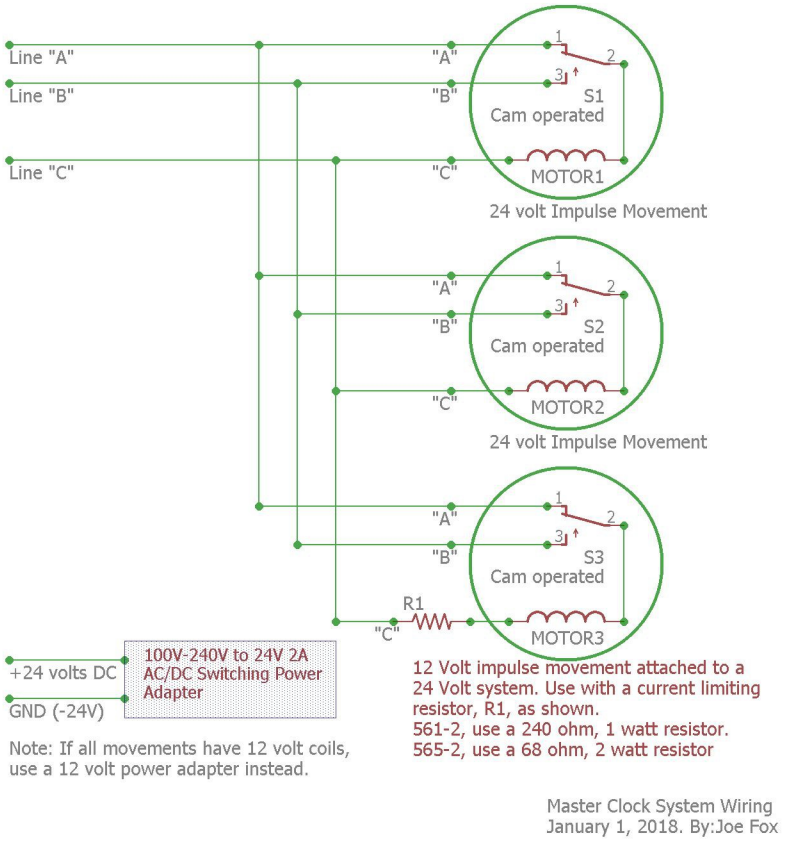


Photo 14: Back of Fiberglass DIY Driver Shield shown in photo 9. Notice the missing header pins. Short yellow wire is Run/Advance switch center lead to I/ O input pin "D2". Long yellow wires are the red LED Anodes, IGBT Gates, and I/O data pin connections "D8" and "D9". Red wire is +24 Volts from the power adapter to the C line screw terminal, and DC to DC converter's +24V input.

3. 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 20 AWG (American Wire Gauge) hookup wire as the GND wire from the (-24V) screw terminal to the (-) minus input side of the converter, and to both source leads of the two IGBT transistors and at least 24 AWG wire to the Arduino's GND and +7.2 volt shield pins. Run 20 AWG insulated wire from the (+24V) screw terminal to the input (+) plus side of the DC to DC converter board and to the C screw terminal. Also 20 AWG insulated wire from the IGBT Drain connections to their respective A and B screw terminals. All other connection wires can be as small as 30 AWG wire wrap wire. I have found that solid Cat5 E cable wires, usually 24 AWG, is a cheap and plentiful source of hookup wire and is easy to work and solder. The lower the gauge number the larger the wire. In addition to the GND wires mentioned above, run a GND wire to one of the Run/Advance switch's outer leads, and to the two LED resistors. Run an insulated wire from the DC to DC plus (+7.2V) output to the Arduino's "VIN" pin. This is the input to the Arduino's 7 to 12 Volt onboard regulator. After wiring the above and verifying things are correct, You can connect the +24V and -24V DC power supply wires to the screw terminals and adjust the output of the DC to DC converter to 7.2 volts at this time. Then verify that 7.2 volts is present at the input "VIN" pin, and only the "VIN" pin, before connecting the shield to the Ethernet and Arduino boards.



Schematic 3: Wiring of system secondaries and a 24 volt, 2 amp AC to DC switching power adapter. All secondaries are connected in parallel: A to A, B to B, C to C. Please note: The wire connections on the left side of this schematic connects to the screw terminals on the right of schematic 2, of the driver shield.

4. Refer to the impulse secondary driver circuit as shown in Schematic 2. The Light Emitting Diodes (LED) and their resistors are actually optional, but they 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). In the photo, my DIY driver board shows a 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. My LED will also illuminate at 12 volts but will be dimmer. This yellow LED is not necessary but I find it helpful to know at a glance if the shield is powered ON and that +24 volts (or 12 volts) is present.
5. The drain (center lead) of the IGBT is connected to its respective screw terminal A or B. Here, I also recommend you use 20 AWG 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.

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:

Correct time is acquired from the internet once per hour via the Network Time Protocol (NTP) to update the Arduino's system timer (system clock). This is necessary because neither of 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. My Arduino's system timers gain or lose about a second per hour with current code 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.

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, two minutes before synchronizing all the clock movements at the top of each hour. In other words, this master clock will never be off more than about ½ second 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: National Institutes of Standards and Technology (NIST) at : http://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..."

New Wireless LAN Master Clock Controller:

My son-in-law gave me a new wireless internet computer (abt $11.00ea) on his visit on the 4th of July, 2016, that he programmed with the clock code, including code updates for the Run/Advance feature. This new Digistump.com wireless processor named Oak, uses an ESP8266 processor with wireless 802.11 on board so no need for the Arduino Uno or Mega processor, the Ethernet shield, and the hard Ethernet wiring needed for access to the internet. Plus this Oak wireless internet processor, the 24Volt to 5Volt DC to DC converter, two IGBT devices, two LEDs and their resistors, Run/Adv switch, and screw terminals will all easily fit onto a single 2" X 2.36" perf-board driver shield. This newer, smaller, and much cheaper version Master Clock Controller does have an issue with the Internet of Things

(IoT) internet connections to overcome so when we solve those, I will write an update covering the Digi-Stump Oak processor build. See Photo 14 and 15 of the Oak configuration that I have already built. The Oak has its own on-board 5 volt input regulator supplying the 3.3 volt operational voltage needed for the ESP8266 processor. So the DC to DC converter is adjusted to 5 Volts instead of 7.2 volts. I have been testing this new system board for a while with no apparent master clock controller problems; other than it just quits working if it looses it's IoT internet connection for a period of time. This problem has been traced to the Internet of Things cloud server at Particle.com Re-establishing the wireless configuration gets it running again without reloading the clock code most of the time.

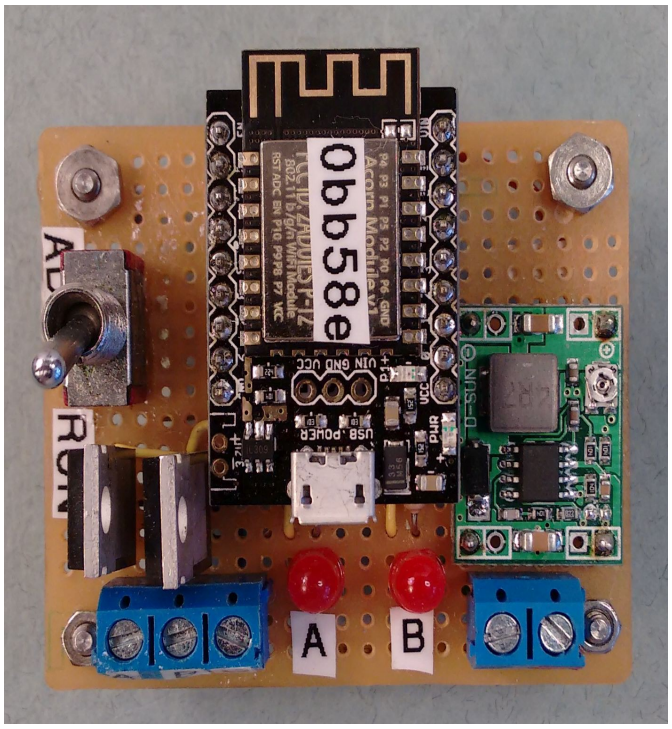


Photo 15: The Digistump Oak wireless Ethernet processor , DC to DC converter [adjusted to 5 volts], IGBT drivers, Run/Adv switch, LEDs, and screw terminals, all on one 2" X 2.36" prototyping board.

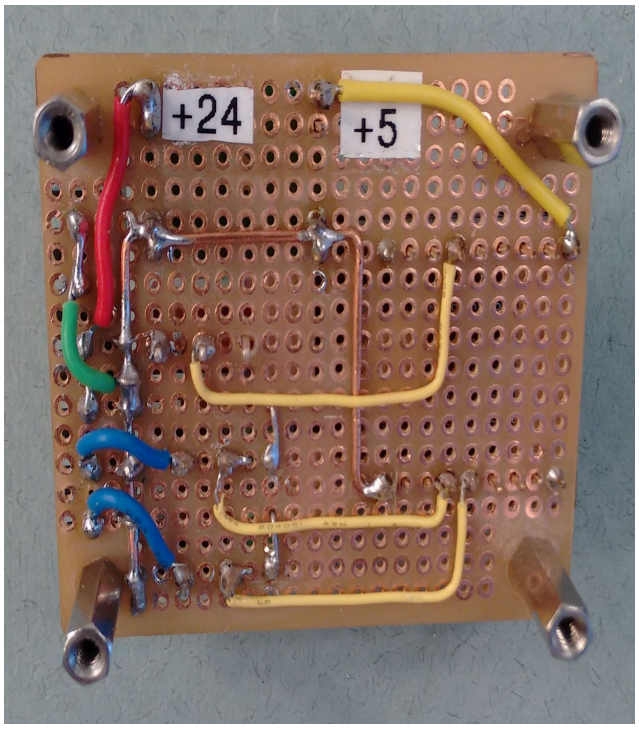


Photo 16: Back of the Oak driver board, showing how I wired the board. Copper wire is ground buss. Shown rotated 90 degrees, screw terminals on the left, WIFI Antenna on the right)

This is however tedious and time consuming, even for an experienced computer programmer. Although the software direction we are looking at would cripple the IoT mode and maybe the Over-The-Air (OTA) updating features, it is not really needed for the dedicated master clock application anyway. We are looking into just loading the code using the Arduinos IDE serially without the IoT and OTA updating features. This should make the Oak unit as reliable as the direct connect Ethernet version and realize a benefit of about an 80 percent reduction in costs over the Arduino/Ethernet shield version, and no wires. The Oak's ESP8266 processor operates on 3.3 volts instead of five volts, and since the IRFZ44 IGBT's will not reliably turn ON with 3.3 volts, I had to find a substitute. The FQP30N06L is a low gate threshold IGBT transistor that works on 3.3 volt logic levels and seems to drive the Secondaries as well as the IRFZ44N version.

Software:

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 but I am including a step by step document, to guide the inexperienced in loading the software onto their Arduino. 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 on the WEB. Those more experienced at programming 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 software code works on both the Arduino Uno and Mega 2560 model computer boards, and the Arduino Ethernet shields. 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 and 3, 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.

Example:Serial output

```
  57:58
  57:59
A  58:00   NTP Request        (A means A LED ON)
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
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
  58:05
  58:06
  58:07
  58:08
 58:58
  58:59...
Then:
A  59:00   off                (A time off  means A LED turned ON, time displayed, then OFF)
```

Then: at 59 minutes, 10 seconds after the minute, A LED ON then OFF every other second

```
   59:09
A  59:10   off
   59:11
A  59:12   off
   59:13
A  59:14   off
   59:15
A  59:16   off
   59:17
A  59:18   off
   59:19
A  59:20   off
   59:21
A  59:22   off
   59:23
A  59:24   off...
```

Then:

```
   59:58
   59:59
AB 00:00   off
   00:01
   00:02...
```

(Top of the hour, both A and B LEDs ON, minutes and seconds, then OFF)

You can uncheck auto-line feed at the bottom 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.

This is the code where the MAC address is assigned (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"
```

Screen output from the "Oak Cloud Serial Terminal", from Particle.com . Notes added are in red text. This is on the "Cloud". So when it works, this can be seen anywhere via the internet

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 132.163.097.001, **(NIST changed it November, 2017)** and its new time server name is time-a-wwv.nist.gov . Phil is considering working on a DNS lookup function so multiple servers would be available by using a global address name. 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 .
Until then, the NTP access is via IP address.

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  Edit 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. 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.

1. Ethernet Shield Sketch came from: http://arduino.cc/en/Tutorial/UdpNtpClient

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.

For those who do not know Hex numbers, here is an example: A byte is two digits, 00, 01, 02, 03, 04, 05, 06, 07, 08, **09**, 0A, 0B, 0C, 0D, 0E, 0F, **10**, 11, 12,...18, **19**, 1A, 1B,...1E, 1F, **20**,...FD, FE, FF, 100. (FF plus 1 carries to a third digit, and so on.) 100Hex = 256 decimal. Four digits of decimal numbers (0000 through 9999) = 10,000 unique numbers. Four digits of Hex numbers (0000 through FFFF) = 65,535 unique numbers.

There are many sources for components for this project. Check for 24 (or 12) volt power adapter 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. Do not forget to 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 Jameco.com other than as a customer. All their parts listings have a convenient link to data sheets for reference, and a fair price for the components.

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).
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

DC to DC Buck Voltage converter, 5pack MP1584EN ultra Small DC-DC 3A power Step-Down Adjustable Module Buck Converter adjustable 5/$9.00 (7/2017) Amazon Prime

ABI 24V 72W 3A AC Adapter Power Supply Driver for 24V LED Strip Light
Sold by: JacobsParts Listed on Amazon.com $11.95 11/2017

Uxcell a14103100ux0411 10 Piece 2.54 mm Pitch PCB Board Prototype Breadboard Single Side Copper 50 mmx70 mm
Sold by: uxcell Listed by Amazon.com $6.12 11/2017

Ethernet Network Shield Module W5100 Micro SD Card Slot For Arduino UNO Mega 2560 1280 328

by HALJIA $9.99 listed on Amazon.com /*Prime 11/2017*

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