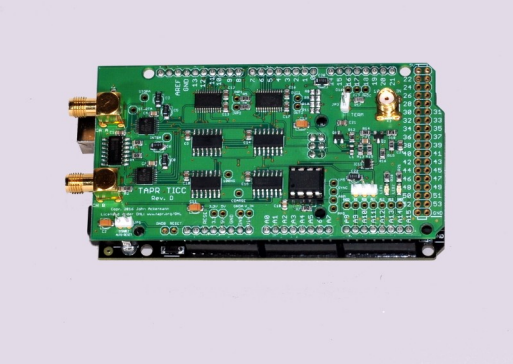
TAPR TICC Timestamping Counter Operation Manual

Revised: 16 February 2017



Introduction

The TAPR TICC is a two-channel timestamping counter ("TSC") implemented as a "shield" daughterboard for an Arduino Mega 2560 controller. It can perform more than 100 measurements per second (both channels) with single-shot resolution of less than 60 picoseconds, and RMS jitter in the 60-70 ps range, yielding a one-second noise Allan Deviation of about $7x10^{-11}$.

A timestamping counter is a bit like the time clock on a factory floor where each employee "punches in" and that time is recorded. The output from a TSC is a record of the arrival time (in seconds since start-up) of each input event, measured against the counter's reference clock. For example, a series of pulse-per-second events might look like this:

```
104.897999794440
105.897999794492
106.897999794549
107.897999794551
108.897999794553
109.897999794552
110.897999794667
```

Note that this data increments by one second for each reading, which matches a pulse-per-second input (if the input were at a 10 PPS rate, each reading would increment by 0.1 second). But also note that the interval isn't *exactly* one second – no two clocks have exactly the same rate, and they all have some amount of noise in their readings. Timestamp information allows one to determine frequency offset (difference between measured and nominal event rate) and stability (noise from reading to reading). Thus, a TSC can be used to characterize important parameters for clock performance.

A single TSC channel compares a low repetition rate source such as a PPS signal against a reference oscillator (in the TICC's case, an external 10 MHz source). To measure the time interval between two PPS signals – for example the output of a clock and a GPS timing receiver – a two-channel counter can be used. Each channel is referenced to the same time scale, so their timestamps can be directly compared. By measuring both channels and subtracting one reading from the other, the time between the two events can be determined. This is the equivalent of the "time interval" mode offered by traditional counters. Time interval data can be used to determine the frequency difference and other information related to the two input signals.

When measuring time interval, the reference clock serves only as a transfer standard and within reason, its quality does not impact the results. Thus an inexpensive reference can be used to measure a pair of very high quality devices.

A TSC can also derive other information from timestamp data, such as period (current timestamp minus last timestamp), ratio (number of pulses on channel A compared to number on channel B), etc.

The TICC has unusually good single-shot resolution of about 60 picoseconds. This is comparable to the best time interval counters commercially available today. High resolution allows more meaningful measurement results in a shorter time. For example, the TICC's noise (measured with the Allan Deviation, or ADEV, statistic) is below is well below $1x10^{-10}$ in one second and below $1x10^{-13}$ in 1000 seconds. Because of its unique design, the TICC requires no calibration.[1]

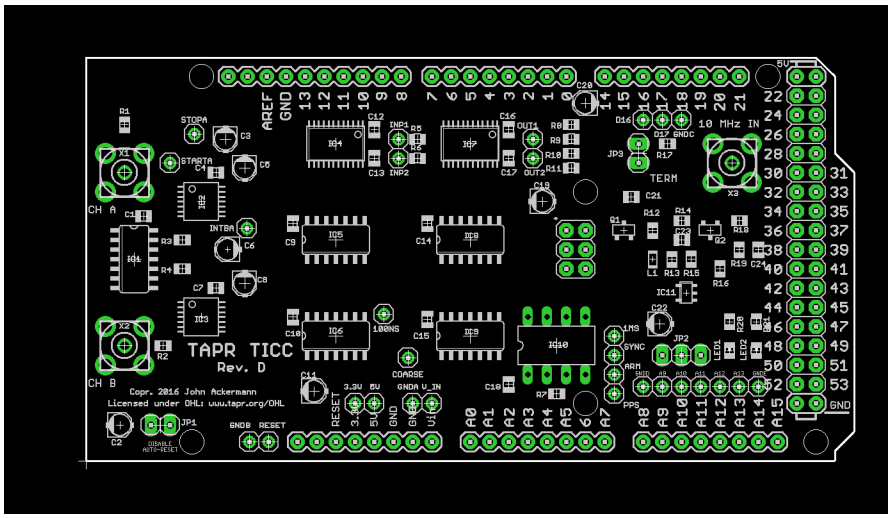
The TICC outputs data in ASCII serial format via USB. The output files can be read by standard analysis software such as W. J. Riley's Stable32, and John Miles' TimeLab software can directly interface with the TICC. (Appendix B describes how to configure TimeLab for use with the TICC).

1 There may be opportunities to improve performance via individual calibration and tweaking of some program variables, but this is not necessary for normal operation.

Operation

The TICC circuit board is a "shield" that mounts to an Arduino Mega 2560 processor. The Arduino is loaded with the TICC software via its USB connection. The software is available at https://github.com/TAPR/TICC. TICC systems provided by TAPR will have the board and processor mated, and software already loaded. Appendix A describes how to install updated software to the Arduino.

Hardware Configuration



Power is provided by the attached Arduino, so no separate connection is required. And the Arduino can operate from power provided by the host computer via the USB connection – no external power source is required.

Connect a 10 MHz reference source to X3 (vertical SMA receptacle) on the board, which feeds a sine-to-square wave converter that accepts signals from -10 to +13 dBm. JP3, if installed, provides a 50 ohm termination.

The channel "A" (X1) and "B" (X2) SMA connectors accept digital signal levels of up to 5 volts without damage. The trigger level is about 1.7 volts. The input impedance is 1 megohm. By default, the TICC triggers on the rising edge of an input signal. This can be change in the configuration menu, though the TDC7200 datasheet recommends using the rising edge when best performance is desired.

Note that many "old school" clocks, such as HP Rubidium and Cesium standards, and some GPSDO, have PPS output signals that significantly exceed TTL level. Experience so far indicates that these signals will not damage the TICC input circuit, but the high amplitude results in ringing that can cause false triggering. If you use a PPS source that has a peak amplitude of more than 5V, I recommend using an attenuator to reduce its level.

Momentarily shorting the "RESET" input (lower left edge of the board) to ground will reset the board.

The default Arduino behavior is to reset the board every time a serial connection is made. This is not helpful if you want to maintain continuous measurements between connections. For example, you might want to run a program on the host computer that connects to the TICC every 1000 seconds to get data and then disconnects. A reset each time this happens would lose the timestamp history. Shorting jumper JP1 ("DISABLE AUTO-RESET) will change this behavior so that the Arduino will not reset on a serial connection; only shorting the reset pin or doing a power cycle will cause a reset. Note that shorting JP1 will

interfere with the Arduino IDE software upload function. Remove the jumper before uploading new software to the board.

Three pin header JP2 is used when slaving multiple TICC boards. See Appendix C for details on its use.

LED1 and LED2 (lower right corner) are under program control. With current software versions, LED1 turns on during the channel A processing loop, and LED2 turns on during the channel B processing loop. Starting with software version 20170219.1, the pads connected to Arduino I/Os A12 and A13 (just below the LEDs) follow the LED status. These outputs go to +5V when the LED is on.

The TICC board provides several other attachment points for connections to additional signals from the Arduino. These are intended for future expansion and are not currently used.

Host Computer Connection

Communication with the TICC occurs over USB with serial port emulation. The port parameters are 115200 baud, 8 bit, no parity, 1 stop bit. All data is 7-bit ASCII.

If your computer does not recognize the Arduino when it is connected to a USB port, you may need to install a driver. Drivers are included in the Arduino IDE, but if you do not want to install that I have extracted the "driver" directory into a zip file and put it in the github directory (see next section) in the "binaries" folder. Download and unzip "arduino_usb_drivers.zip" and when Windows complains that it can't find a driver, point to that location. Here is a (non-warranted!) link to information on driver installation: http://www.hobbytronics.co.uk/arduino-installation

Software Versions and Installation

The TICC software is maintained in a "git" repository located at https://github.com/TAPR/TICC. The repository includes supporting documentation and other materials as well as the source code files in Arduino sketch format, located in the "TICC" folder. Version numbering is date-based. At significant points in the development cycle, I issue "releases" (snapshots of the repository at that point in time), and also create binary files that can be uploaded to the TICC/Arduino without the need to install the full Arduino IDE.

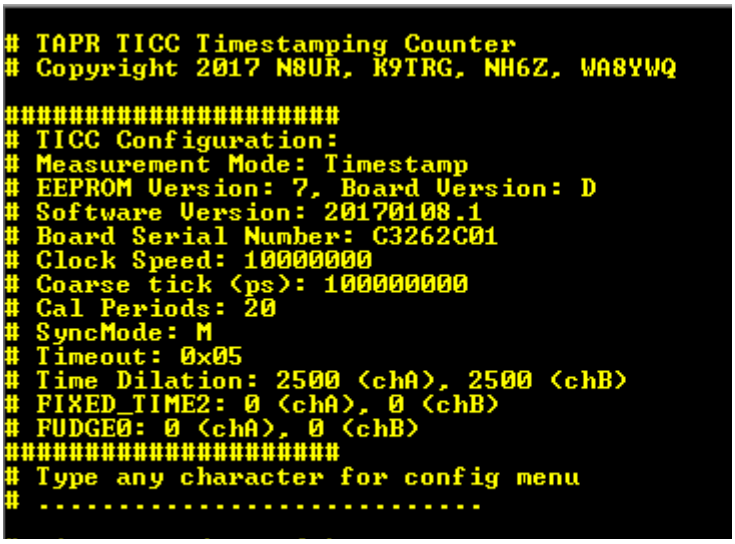
TICC boards manufactured in the initial TAPR production run (winter 2017) are pre-loaded with software version 20170108.1, and except as noted this manual reflects that version.

See Appendix A for information on building and installing new versions of the code.

Software Configuration

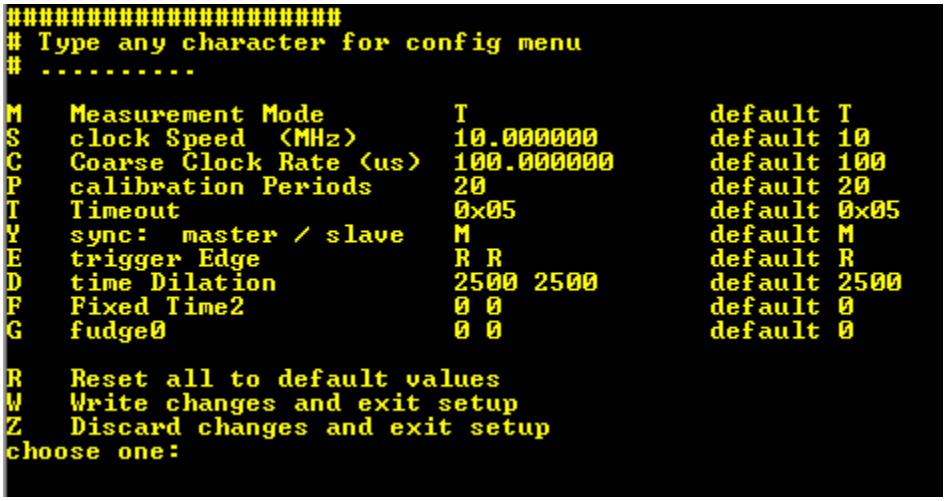
Assuming you have a serial terminal program set to the appropriate serial port number and parameters, at startup the TICC will display its configuration settings and pause for about 5 seconds, as shown below.[2] Any serial port program should work, including the Arduino IDE "Serial Monitor" window – although that has quite limited functionality. On a Windows system, I've found that RealTerm works well: https://sourceforge.net/projects/realterm

2 Apologies to George Byrkit for getting his call wrong in this screen. It's been corrected to "K9TRV."



The version numbers shown on your board may be different, but the numeric default values should be the same. If you do not see these values, it is possible that the default values have not been written to EEPROM on your Arduino. Go into the configuration menu and select "Reset all to default value" and then restart the board.

If you enter a character before the timeout, you will see a configuration menu like the following:



Press the letter for the desired option, and either the change will take effect or a sub-menu will appear allowing further input. The options (as of the 20170108.1 software release) are:

"M" – Measurement Mode, which allows the following choices:

> **Timestamp** outputs the timestamp of each event received on either channel A or B as it is received. Each measurement includes "chA" or "chB" to identify the channel. One or both channels may provide input, and inputs may be connected or disconnected at any time.
>
> **Period** outputs the difference between the current reading for a channel and the last reading for that channel. Each measurement includes "chA" or "chB" to identify the channel. One or both channels may provide input, and inputs may be turned on or off at any time.
>
> **Time Interval** outputs the difference of a pair of measurements. When a new event occurs on

each of channels A and B, the TICC subtracts the channel A timestamp from the channel B timestamp and outputs the difference. No output occurs until a pair of readings have occurred. Both channels must be fed with events at the same nominal rate.

TimeLab outputs three measurements for each pair of readings: channel A timestamp, channel B timestamp, and a pseudo-timestamp labled as "channel C" that consists of the (chB – chA) time interval added to the integer part of the channel B timestamp. This mode works with the multi-channel input capability of the TimeLab software to allow 3-corner hat measurements (see Appendix B for more information). This mode has not yet been thoroughly tested.

Debug outputs the raw data output from the TDC7200 chip, intermediate results, and the final timestamp calculation. The output fields are: time1Result, time2Result, Clock1Result, Cal1Result, Cal2Result, time-of-flight, PICcount, timestamp, and channel identifier (use the source, Luke).

"S" – Clock Speed. The frequency of the external reference clock applied to X3. Default is 10 MHz, and valid range is 1 to 16 MHz. The TDC7200 data sheet indicates that the noise performance improves with higher clock frequency. NOTE: if you change the clock speed, you will need to modify the PD15 firmware in the PIC divider to match; the installed firmware expects a 10 MHz clock rate.

"C" – Coarse Clock Rate. Fixed at 100 us with the current TICC Rev. D board and TVB PD15 PIC firmware; this is the rate of the coarse clock used by the TICC. Do not change this value!

"P" – Calibration Periods. This is the number of 10 MHz clock cycles the TDC7200 chip uses for its calibration process after every measurement. Valid settings are 2, 10, 20, and 40. 20 is the current default; further experimentation is needed to determine if this is optimum.

"T" – Timeout. This is a value from 0 to 255 that sets the TDC7200 delay before terminating a measurement. This value should not be required at all, but due to either a TDC microcode issue, or a programming error on my part, this must be set for proper operation. The value is set experimentally to 5; longer timeouts limit the measurement rate, and too short a value will cause grossly improper results.

"Y" – Sync Master/Slave. Normally set to "M" for master mode. Set to "S" when the board is a slave in a multi-board configuration. See Appendix C for further information.

"E" – Trigger Edge. Sets trigger edge for each channel independently. The TDC7200 recommends using rising edge, and that is the default.

"D" – Time Dilation. Please forgive the cute name. This is an adjustment for an apparent small non-linearity in the TDC7200 output over the range from 0 to 100 nanoseconds. If set too low, there will be a sawtooth visible in the phase results from slowly drifting signals. This requires further investigation, but the default setting of 2500 seems to work well.

"F" – Fixed Time2. This is an optimization opportunity. One step of the TDC7200 measurement process when used in the TICC hardware configuration calculates a value that is (or should) always be the same. Replacing the calculation with a fixed value reduces the measurement jitter by a factor of about 1.4. When FIXED_TIME2 is set to 0 (default), the calculation proceeds normally. To experiment with the lower noise calculation method, set to the appropriate value. How do you determine the appropriate value? Set the TICC measurement mode to "Debug" and among the data output will be a value for "time2Result". Capture some data from each channel, and use the average (each channel will be different). The value will typically be in the range of 1100 to 1300, and an average of 100 samples is more than sufficient.

"G" – Fudge0. This is an optional adjustment, in picoseconds, that can be applied to each channel. One use might be to trim out propagation differences by applying a pulse to both channels through identical cable lengths. Changing the Fudge0 value for one of the channels allows compensating for the delay difference.

"R" – Reset to default values, "W" – write changes and exit, and "Z" – discard changes and exit all do what they say.

Beginning with software version 20170129.1, two additional commands are added:

"H" – Poll Character. This is normally unset in which case the TICC acts as a "talker" outputting every data value on the USB cable. If this is set to a single character, the TICC will switch to "poll" mode where it will only output a measurement after receiving the specified character. This is useful where the host program only wants to save samples of the data. For example, the desire may be to log PPS time interval data once per hour. Polling will not work in two-channel timestamp mode as only one result is output per poll character.

For example, if the poll character is set to "!", the TICC will discard measurements until the host computer sends "!" over the serial connection. The TICC will then output the results of the next measurement. Lather, rinse, repeat.

"X" – Clear EEPROM. This command does not appear in the menu and is meant only for debugging. Entering "X" will clear the entire EEPROM space in the Arduino, setting the board back to its factory condition. Restarting the TICC after this will cause a new serial number to be generated and default configuration to be reloaded.

Output Format

Each line of non-result output from the TICC (e.g., the startup screen) is preceded by the character "#" which will cause most data logging programs to treat that output as comments.

When the TICC software runs on an Arduino board for the first time, it will create an 8 digit "serial number" derived from a random number generator. This is intended to help users identify multiple units. Clearing the Arduino's EEPROM memory will cause a new serial number to be generated. TAPR does not maintain a log of unit serial numbers.

All data fields output by the TICC are units of seconds with 12 decimal places. The least significant digit is 1 picosecond. After the configuration screen displays, the TICC will generate no further output until it has measurement results to report. All data is output in seconds with 12 decimal places resolution. The second field of each line indicates the channel and in some instances the measurement mode.

Circuit Description

A limitation of digital counters is that their resolution is tied to the clock speed. If there is only one clock tick every millisecond, you cannot measure with greater than one millisecond resolution. A clock rate of 1 GHz, which is challenging to achieve, provides a resolution of one nanosecond.

In order to obtain higher resolution, it is necessary to interpolate between clock cycles. Traditionally, this has required analog circuits using methods such as measuring the decreasing voltage over time across the terminals of a capacitor. The best of these schemes can yield resolution and jitter less than 100 picoseconds, but they are complex and require periodic calibration.

The TICC uses a different method to measure sub-clock-cycle times, thanks to the Texas Instruments TDC7200 time-to-data converter chip. The TDC7200 is at the core of the TICC's design, but a significant amount of additional logic is required to create a complete timestamping counter.

High Level Design

The TICC is a clock that measures when external events (logic pulses on the TICC inputs) occur. It does this through a combination of hardware on the TICC shield, and software on the Arduino. It's difficult to look at either the hardware schematic or the software source code in isolation to gain an understanding of the system. This high-level design description should help.

Central to the design is a software counter implemented in the Arduino that counts the number of 100 us "ticks" since system startup. The counter is a variable called `PICcount`, and it increments via hardware interrupts each time a 10 kHz clock (`COARSE_CLOCK`) on the TICC board ticks. `PICcount` is therefore a timescale based on the number of 100 us ticks since the system started.[3] Note that the "time" referred to here is not related to an outside timescale like UTC; it starts from zero each time the system initializes.

The goal of the TICC is to provide data with resolution measured in picoseconds, and by itself the 100us tick of the `COARSE_CLOCK` does not come close to meeting that requirement. The TDC7200 chip can measure time intervals with <60 picosecond resolution, so it is used to measure the time of an event within the window between two `COARSE_CLOCK` ticks.

When an event appears at a TICC input, the associated TDC7200 chip starts its measurement.[4]

Once the TDC chip has started timing, the next `COARSE_CLOCK` tick to arrive[5] stops the measurement, and the TDC then calculates the elapsed time, called the "time of flight" or "`TOF`," and sends that value to the Arduino via the SPI communication bus. The same `COARSE_CLOCK` tick also raises an Arduino hardware interrupt that causes the software to copy the current value of `PICcount` into another variable called `PICstop`.[6] The value in `PICstop` is always a timestamp after the event occurred.

The `TOF` measurement tells us, with picosecond resolution, the time from the event until the timestamp captured in `PICstop`. Since `PICstop` is always captured *after* the event, *subtracting* `TOF` from it gives the event's actual timestamp with the full resolution of the TDC7200. The TICC software calculates the timestamp to one picosecond resolution and either directly outputs this value on the serial port, or uses it as part of a time interval or nother calculation.

---

3 `PICcount` is a 64 bit variable. Even at a 100 us clock rate, it would take millions of years to overflow.

4 See the next section for more details on how the TDC7200 does this.

5 This is not quite true. The TDC chip requires a minimum time period between its "START" and "STOP" inputs. A circuit described below called the "STOP GATE" ensures that the minimum time period is met and could result in the second, rather than the first, COARSE_CLOCK tick be the one that is used.

6 PICstop is stored separately for each of the two channels.

The TICC has two input channels that operate independently, but work with the same time scale. Thus measurements on each channel can be compared with one another. Multiple TICC boards can be synchronized to allow comparisons of 4, 6, 8, or more channels. Multiple-unit use is described in Appendix C.

TDC7200 Operation

Texas Instruments designed the TDC7200 to measure the flow rate of fluids using ultrasonic transducers. In normal use, it starts timing when a "ping" is sent and stops when the transducer hears one or more echoes – the resulting time of flight value can be used to calculate the speed of the fluid flow. However, the chip can be used for other purposes. It is essentially a stopwatch with extremely high resolution. Its START and STOP pins serve as signal inputs.

Like the TICC at a higher level, the TDC7200 has two timing circuits, one (relatively) coarse and the other (very) fine, used to measure the time between a pulse arriving on the chip's START pin and another arriving on the STOP pin. The coarse timer is a conventional digital counter using a 10 MHz clock to provide 100ns resolution, while the fine timer is a ring oscillator with 63 inverter stages providing a period of about 57 ps[7] that interpolates between clock ticks. The 10 MHz clock is the same one used to derive the COARSE_CLOCK tick, so all timing on the TICC board is synchronous.

A ring oscillator[8] is a chain of an odd number of digital logic inverters hooked output-to-input – sometimes described as "a snake eating its own tail." The feedback from the end of the chain to the beginning results in a free-running oscillator with a period based on the propagation delay of the string of gates. The TDC7200 fine timer counts the number of times the ring oscillates during the measured interval.

While the ring oscillator is very fast, it is not stabilized and its speed varies with temperature and other factors. One of the really clever features in the TDC7200 design is the calibration cycle that occurs at the end of every measurement where the chip counts the number of ring oscillator periods over a number of 100 ns ticks of the external 10 MHz reference. From that data the Arduino software can calculate the actual period of the ring counter at the time of the measurement. This provides temperature compensation among other things.

When the TDC7200 sees a START signal, the ring oscillator runs until the next 100 ns clock edge, and the chip counts the number of ring cycles; the actual time is the number of cycles times the period determined by the calibration routine. When a STOP signal arrives, the ring counter starts again and continues until the next edge of the 100 ns clock. The illustration below (from the TDC7200 datasheet) shows how this works.

7 TI states that the nominal resolution is 55 picoseconds, but our measurements consistently show 57+ ps.
8 See https://en.wikipedia.org/wiki/Ring_oscillator

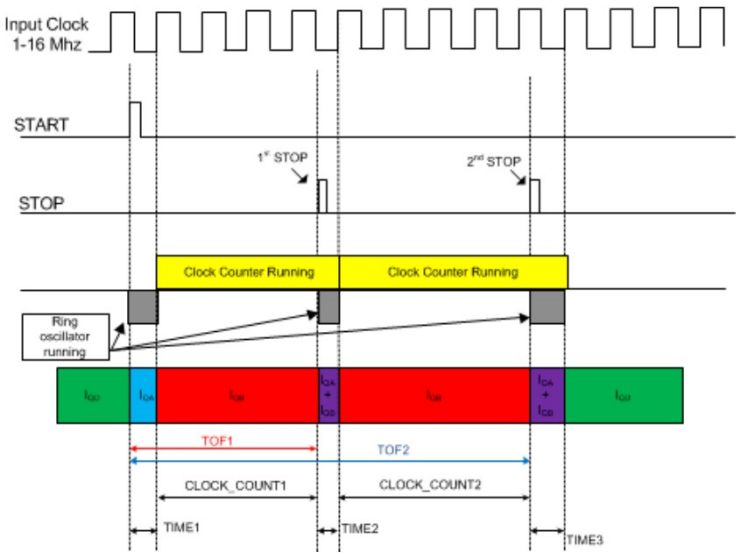


Figure 18. Measurement Mode 2

The elapsed time (what TI calls "time of flight" or "TOF") is thus TIME1 + CLOCK_COUNT1 – TIME2. (In some applications, multiple STOP signals may be received after a single START signal, so the illustration shows additional measurements. The TICC uses only the first STOP pulse.)

The TDC7200 has two limitations that prevent it from being a useful standalone time interval counter: first, the time from START to STOP must be greater than 12 nanoseconds; and second, the maximum time it can measure is about 6 milliseconds. Thus, it won't work for general-purpose use.

The first limitations is overcome by the STOP GATE circuit described below, while the `COARSE_CLOCK` on the TICC and `PICcount` timer on the Arduino address the second.

TICC Timestamp Logic

The TICC adds circuitry around the TDC7200, together with software in the Arduino processor to which the TICC is mounted, that creates a fully functional timestamping counter.

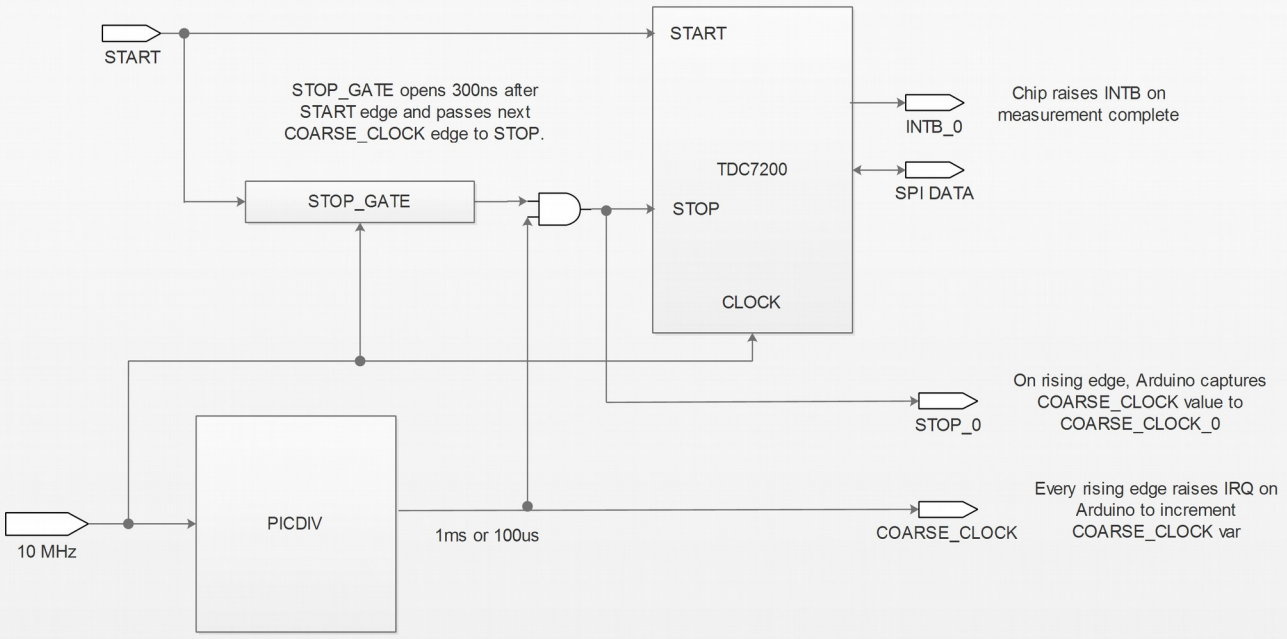
A PIC chip loaded with Tom Van Baak's frequency PD15 divider firmware generates 100 us pulses from the same 10 MHz reference that drives the TDC7200. In addition to triggering an interrupt on the Arduino, the 100 us `COARSE_CLOCK` signal provides the STOP signal to the TDC7200 through a logic block I call the Stop Gate that consists of flip-flops and a shift register. The Stop Gate ensures that the TDC7200's 12 ns minimum time requirement is met by passing only a `COARSE_CLOCK` tick that meets the timing requirement.

The rising edge of each pulse that arrives from the device under test ("DUT") triggers the START pin of the associated TDC7200 chip, and also arms the Stop Gate. The Stop Gate is clocked by the 100ns system clock and waits for three system clocks (300ns) before allowing a `COARSE_CLOCK` tick to pass. Once that time has passed, the next `COARSE_CLOCK` will be routed to the TDC7200 STOP pin and the Arduino hardware interrupt pin. This ensures that the TDC7200 minimum START-to-STOP time is met. As a result, the TOF will range from a minimum of 300 ns (`COARSE_CLOCK` arrives just as the gate opens) to 100.299...us (`COARSE_CLOCK` arrived just under 300 ns after, so wait for one full cycle).

The following diagram shows the overall functional diagram of one TICC hardware channel.

TICC FUNCTIONAL DIAGRAM
6 March 2016

One of Two TSC Channels



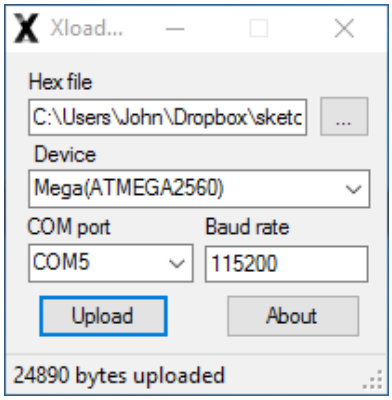
APPENDIX A
Building and Installing TICC Software

There are two ways to install updated software to the TICC/Arduino system. You can install the Arduino IDE, download the TICC source code from the git repository, and then build and upload binary files through the IDE. If you want to modify the code, this is the required method. Precompiled binaries of selected versions are also available from the repository, and those can be installed without the Arduino IDE. This Appendix provides a short description of how to do both.

Install Precompiled Binaries.

Download the desired binary file from https://github.com/TAPR/TICC/binaries, along with the "Xloader.zip" file. (Xloader is a Windows tool; if you are using another operating system, you will need to use another program. Almost all Arduino uploader programs use the same "avrdude" program, which is available for many platforms – they just provide a more user-friendly wrapper.)

Install and open the "Xloader" software. Browse to the location of the "TICC.ino-YYYYMMDD.X.hex" file. Select board type "Mega (ATMEGA2560)" and select the appropriate com port. Set baud rate to 115200. Click "Upload" and after a few seconds the program should indicate the number of bytes sent, as shown below.



You're finished!

You can now connect the TICC to a serial monitor (115200, N81) and should see the startup sequence described in the manual. Note that new software versions may update the stored configuration data. Changed configuration format should be detected when the Arduino starts, and if required default values will be rewritten.

Build from Source.

Install the Arduino IDE (available for many platforms from https://www.arduino.cc/en/main/software) and install it. Download the TICC repository from https://github.com/TAPR/TICC either with git tools, or simply by clicking the "Clone or Download" button, then "Download ZIP," and extracting into your Arduino folder.

Open the Arduino IDE and select File/Sketchbook/TICC/TICC. You should see the TICC source files appear in the editor window.

When you are ready to upload to the Arduino, from the menu select Tools/Board/Mega or Mega 2560, make sure the board is connected via USB, and then select the serial port via Tools/Port.

Upload the file by selecting either Sketch/Upload, or clicking on the Upload button near the top of the IDE window (round button with right-pointing arrow). If the upload succeeds, you should see a message at the bottom of the IDE showing the memory usage and other data. You can now connect the TICC to a serial monitor (115200, N81) and should see the startup sequence described in the manual.

Note that new software versions may update the stored configuration data. Changed configuration format should be detected when the Arduino starts, and if required default values will be rewritten.

APPENDIX B
Using the TICC with TimeLab

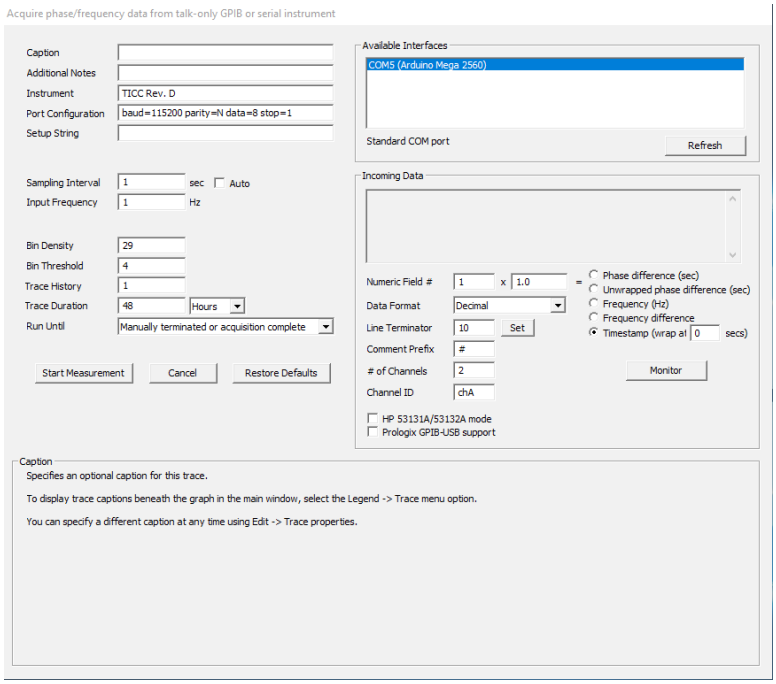
TimeLab, a free software program written by John Miles, KE5FX, has become a standard tool in the "time-nuts" community. The latest version is available at http://www.ke5fx.com/timelab/beta.htm

TimeLab can communicate directly with the TICC via USB serial port. To capture data, use the "Acquire/Acquire from counter in Talk-Only mode" menu option. Select the appropriate serial port in the "Available Interfaces" box and set the parameters to 115200 baud, no parity, 8 bits, 1 stop bit.

Set "Numeric Field" to "1" and the multiplier to 1.0. Set the "Comment Prefix" to "#".

- For single-channel timestamp data, tick the button for "Timestamp" and enter "0" as the wrap value.
- For time interval (channel B minus channel A) mode, tick the "Phase difference" button.

If you use the most recent TimeLab beta version (1.29 or later), you can do a couple of interesting multi-channel measurements. Before describing them, here is what the "Acquire from counter…" setup screen looks like in version 1.29:



The simplest trick is measuring timestamp data from channels A and B simultaneously. To do that, enter "2" as the "# of Channels" and "chA" as the "Channel ID". Note that this requires both channels to have the same nominal measurement rate (e.g., PPS). When the measurement run begins, TimeLab will generate two sets of measurements, one for each channel.

If you are more adventurous, you can do "three-cornered hat" measurements that theoretically allow the individual performance of each device to be extracted from a set of three measurements: A-B, A-C, and B-C. To do this, set the TICC to "TimeLab" mode and set the TimeLab number of channels to 3. In this mode, the TICC will output data that looks something like this:

```
17.000000001234 chA
17.000000001235 chB
17.000000000001 chC
```

The lines for "chA" and "chB" are the timestamps for each channel, just as in normal timestamp mode. The line for "chC" is a hack – it is the time interval (`chB - chA`) appended to the integer part of the last "chA" reading. This provides the "B-C" reading necessary for the three-cornered hat.

I won't go into the details here of configuring TimeLab to display three-corner results in a pretty way, and there are a bunch of warnings about the usefulness of the results, but these settings will generate the needed data values.

APPENDIX C
Master/Slave Mode

Multiple TICC units may be connected to extend the number of measurement channels. In this configuration, each unit operates independently but is fed from a common source of 10 MHz and 10 kHz clock signals and has a common starting point for its timescale counter (the `PICcount` variable). Each unit outputs data for its two measurement channels via its USB serial connection. The external host computer needs to have enough USB ports to connect to each TICC, and software to multiplex the data streams as required.

There are three elements required for synchronous operation:

1. Each TICC is fed from the same 10 MHz source, preferably in a way that provides minimal phase difference at the input of each board.

2. Each TICC shares a common, in phase, 10 kHz `COARSE_CLOCK` signal. This is accomplished by designating one TICC as the "MASTER" and feeding its COARSE_CLOCK signal to the other "SLAVE" boards. The 12F675 PIC chip (IC10) that divides 10 MHz to 10 kHz is not installed on the SLAVE boards.

3. At startup, a synchonization pulse is sent on an I/O pin from the MASTER to the SLAVE boards. This pulse resets the PICcount value to zero so the timescale epoch of all boards is the same.

Connector JP2 is used to link the boards. Pin 1 carries the `SLAVE_SYNC` pulse, pin 2 is ground, and pin 3 carries the 10 kHz `COARSE_CLOCK` signal.

To configure MASTER/SLAVE operation:

1. Feed each board from a common 10 MHz source via a distribution amplifier, splitter, or even via "tee" connectors (in that case, install the TERMINATION jumper only on the last board in the chain).

2. Connect JP2 pins 1, 2, and 3 of all the boards in daisy-chain fashion.

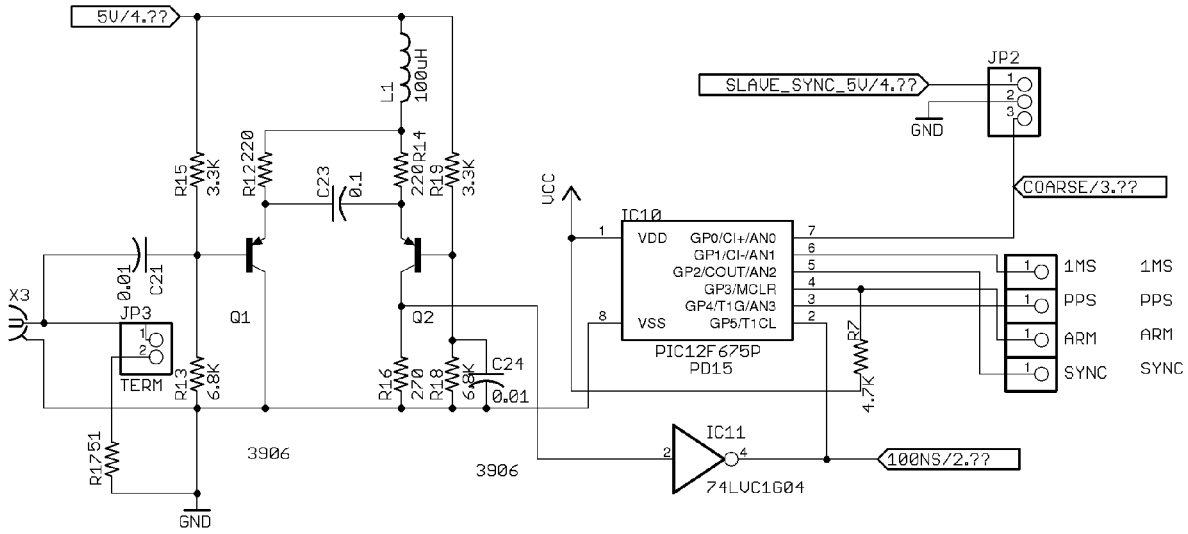
3. Remove IC10 from the slave boards.

4. Daisy-chain the RESET pin of all boards, so they can be reset simultaneously. (This is not a requirement, but is likely to make life easier.)

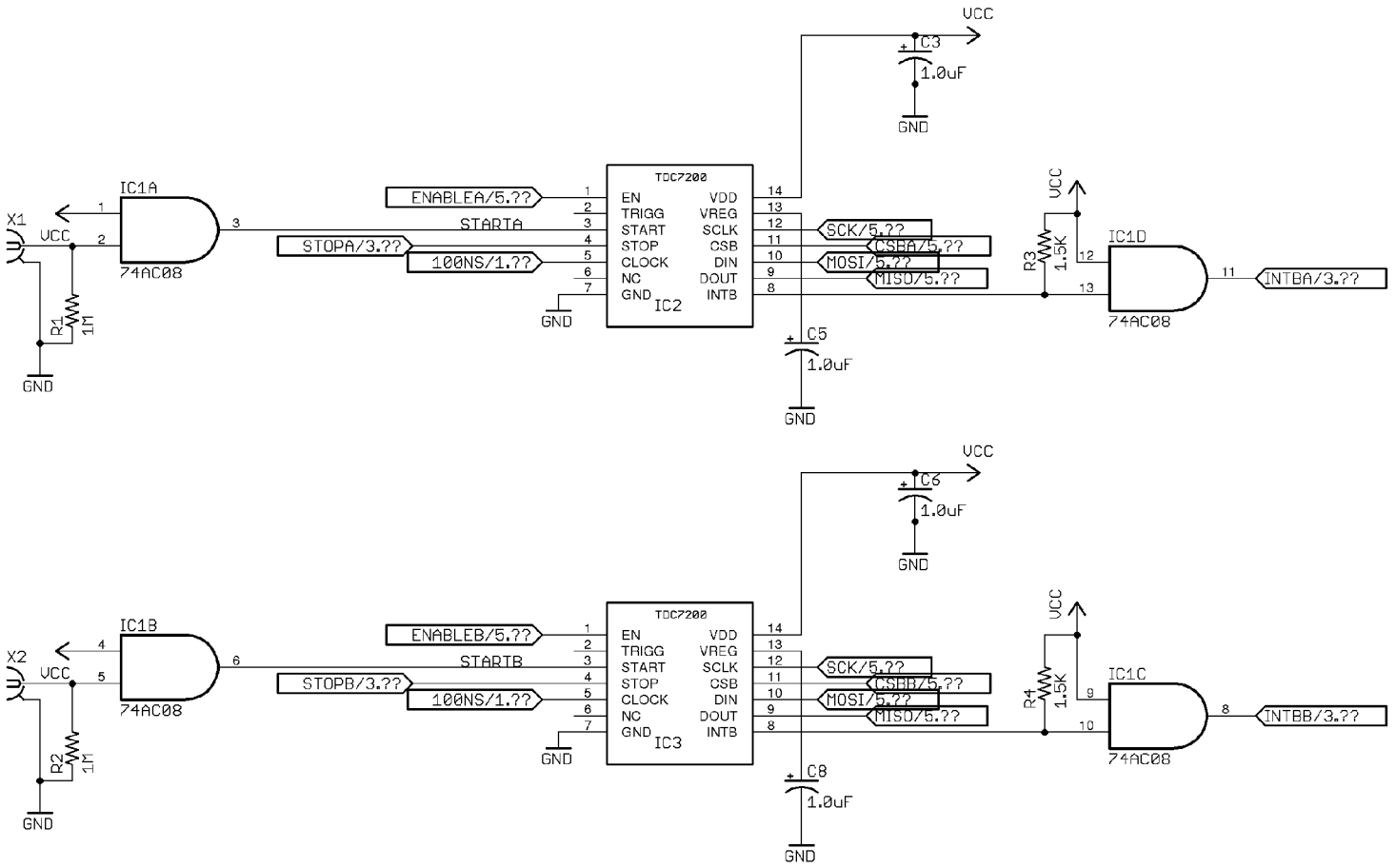
5. In the board configuration menu, set the MASTER/SLAVE parameter to match each board. There may be only one MASTER and an unlimited number of SLAVES.

After a reset, the boards should be synchronized.

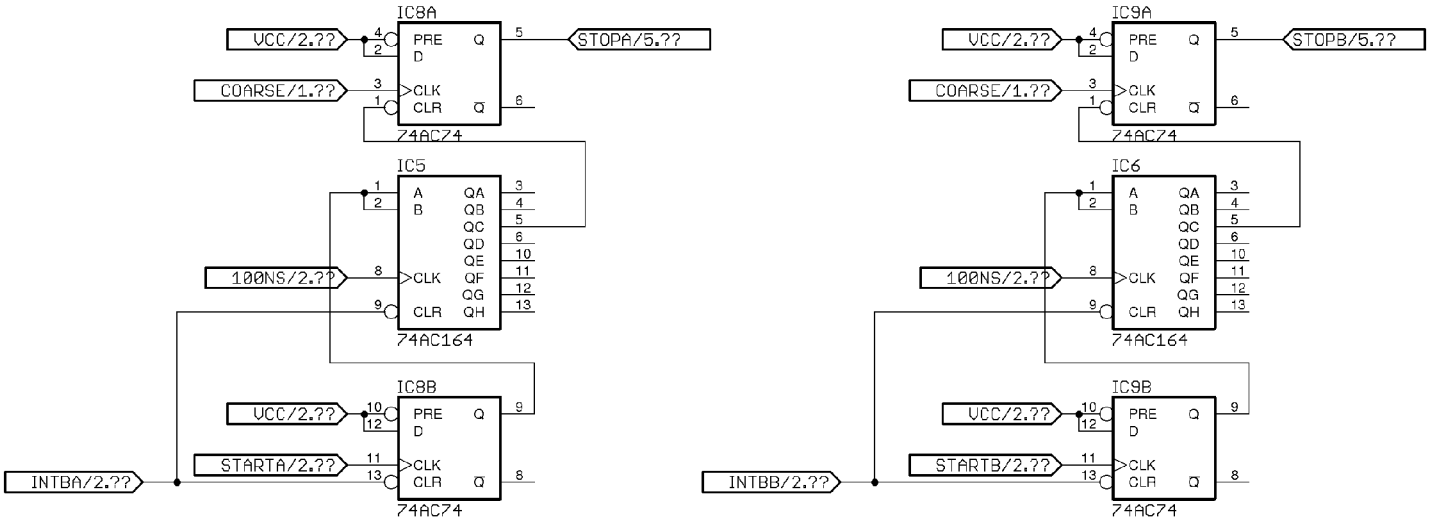
APPENDIX D
Schematics



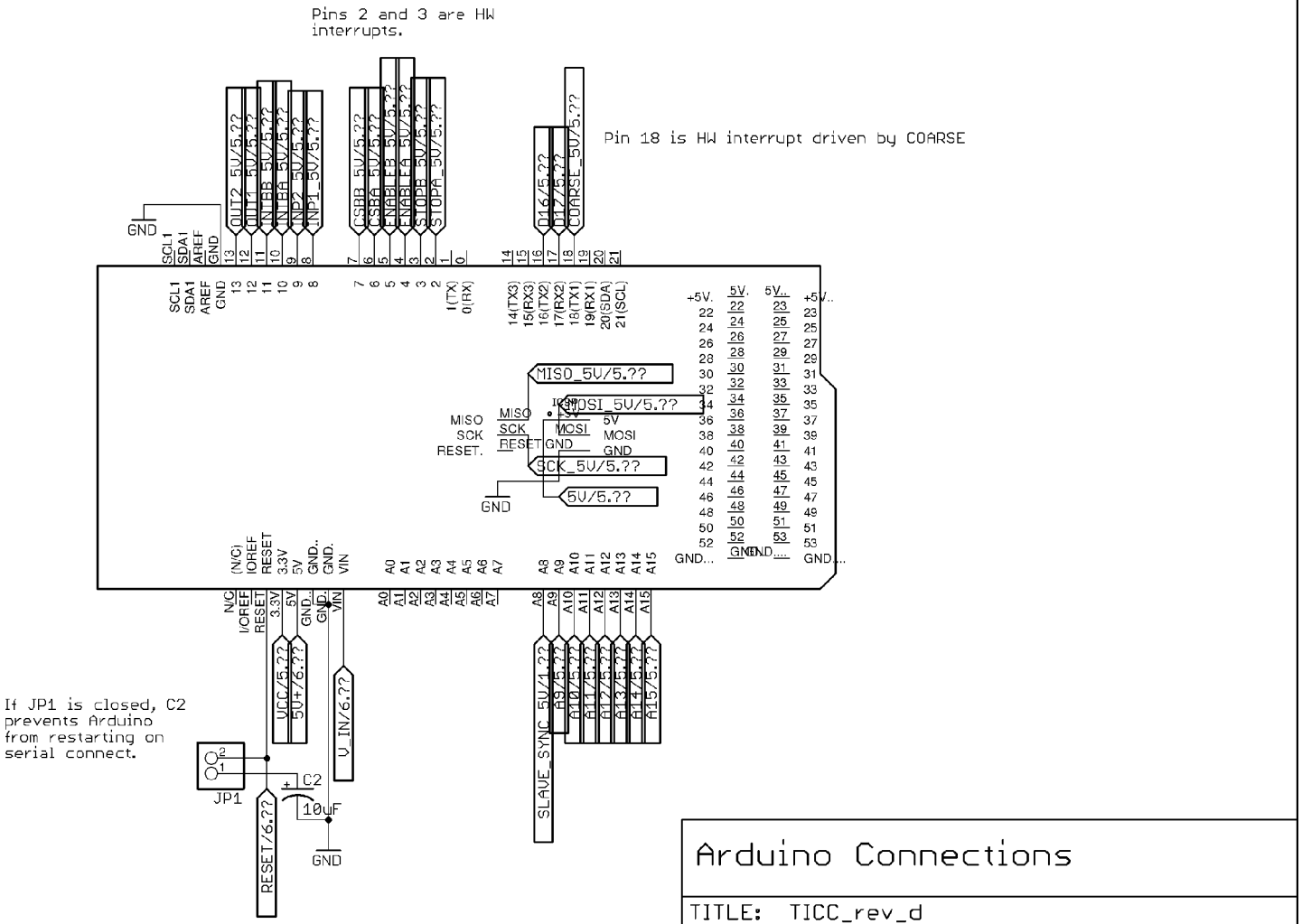
Sine-Square Conv. and PICDIV	
TITLE: TICC_rev_d	
Document Number:	REV:
Date: 9/25/16 2:06 PM	Sheet: 1/6



TDC Chips		
TITLE: TICC_rev_d		
Document Number:		REV:
Date: 9/25/16 2:06 PM	Sheet: 2/6	

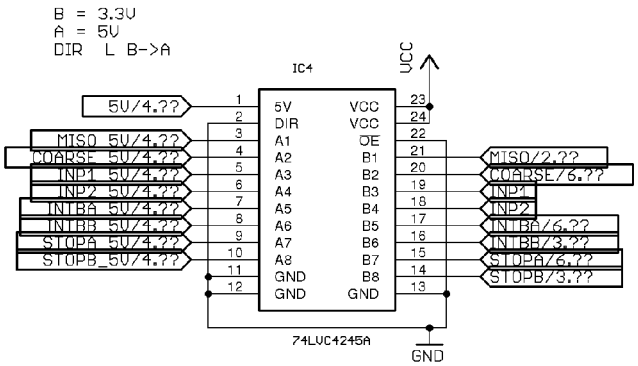


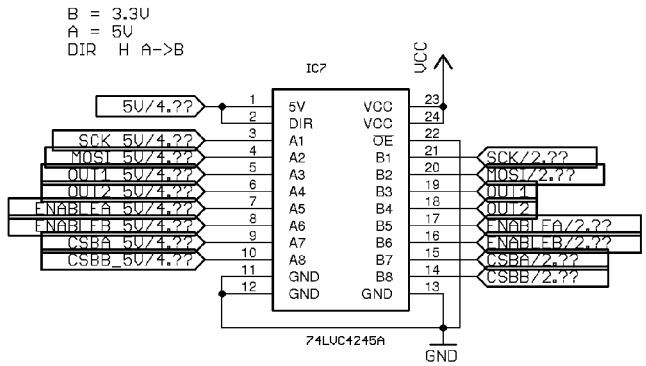
Stop Gate		
TITLE: TICC_rev_d		
Document Number:		REV:
Date: 9/25/16 2:06 PM	Sheet: 3/6	

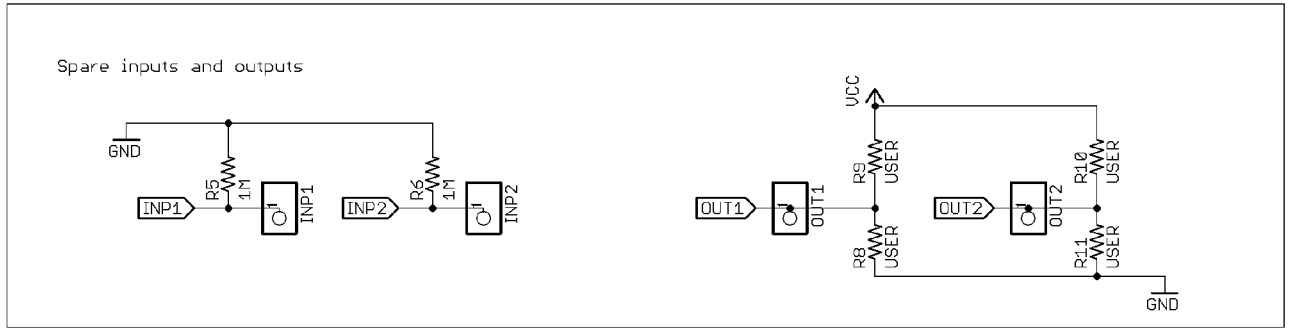


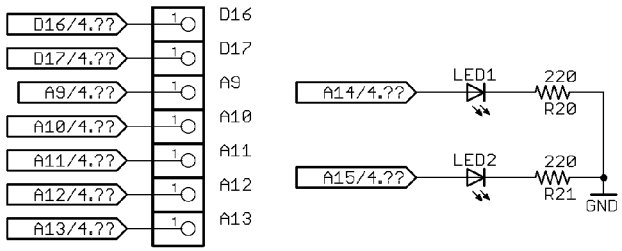
Arduino Connections

TITLE: TICC_rev_d	
Document Number:	REV:
Date: 9/25/16 2:06 PM	Sheet: 4/6

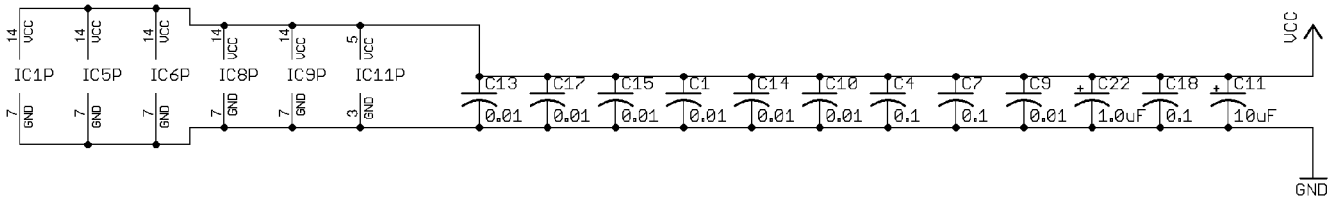


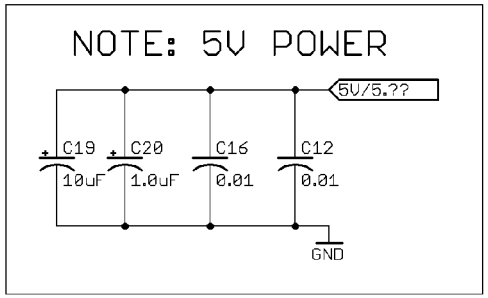


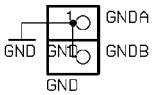


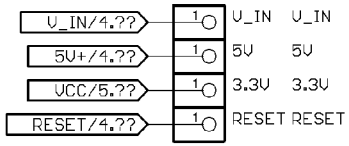


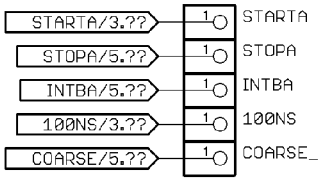
Level Translation and I/O	
TITLE: TICC_rev_d	
Document Number:	REV:
Date: 9/25/16 2:06 PM	Sheet: 5/6











Power and Test Points	
TITLE: TICC_rev_d	
Document Number:	REV:
Date: 9/25/16 2:06 PM	Sheet: 6/6