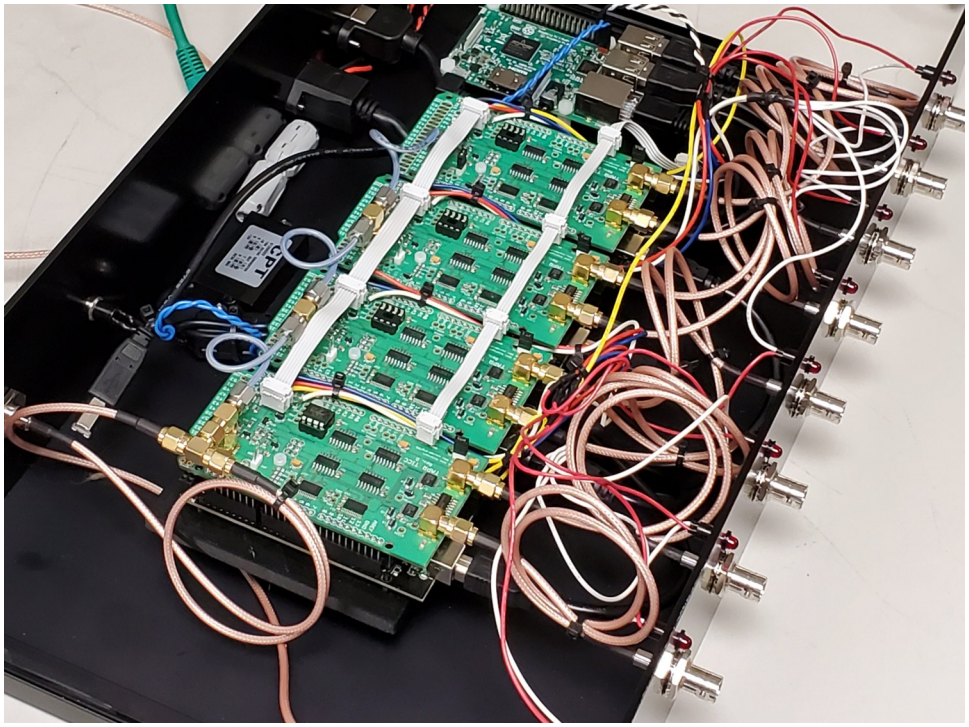
# multi-TICC:
# A Multi-Channel Timestamping Counter based on the TAPR TICC



## Introduction

The TAPR TICC (https://tapr.org/kits_ticc.html) is a two-channel “timestamping” counter with a resolution of <60 picoseconds. It is a “shield” that mounts on an Arduino Mega 2560 processor board, and it communicates via the Arduino’s USB port.

A timestamping counter is one that maintains an internal timescale (i.e., a counter clocked by an external 10 MHz reference that starts at power-on and runs continuously) that marks the occurrence time of external events based on that timescale. Every time a pulse appears on one of the TICC’s two input channels (called “chA” and “chB”), the TICC outputs the timestamp of that event and the channel on which it occurred.

A timestamp is a basic measurement that can be used directly to compare the period and stability of the input signal compared to the reference clock. You can also compare timestamps from channel to channel. For example, if separate pulse-per-second signals are

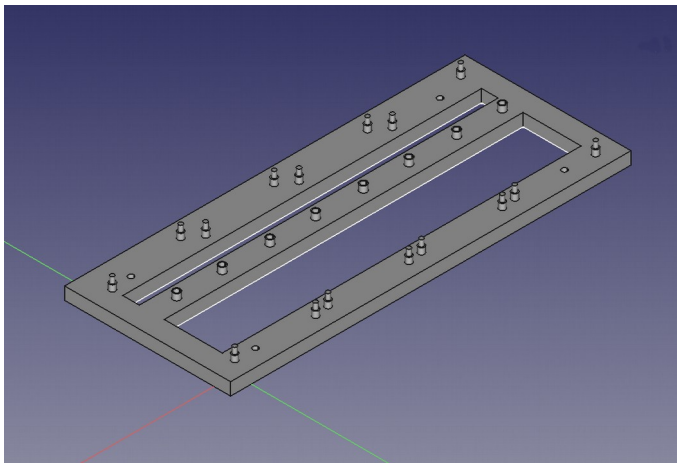
applied to chA and chB, the time interval between the signals can be calculated simply by subtracting the timestamps. Timestamps are very useful building blocks.

While the TICC is a two-channel device, it provides on-board headers that allow multiple TICCs to be driven by a common external reference and share a single timescale. In this way 4-, 6-, 8- channel or even larger counters can be assembled, with direct comparison possibl across all those channels. This application note describes the three aspects of assembling and using a “multi-TICC”: hardware setup, firmware updates, and host processor software.

## Hardware Configuration

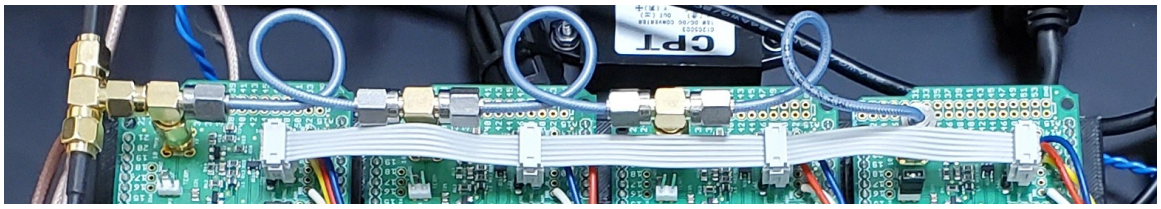
In the multi-TICC, one board is configured as the “master” and provides clock and synchronization signals to the other “slave” boards. All boards run off a common 10 MHz reference clock. Their USB outputs are sent to a host processor which makes the data streams available via ethernet.

To make it much easier to hold the boards together, I created (with a lot of help from Mike, W8RKO) a 3-D printed carrier that will align and secure 4 boards. That carrier can in turn be mounted to a base plate.



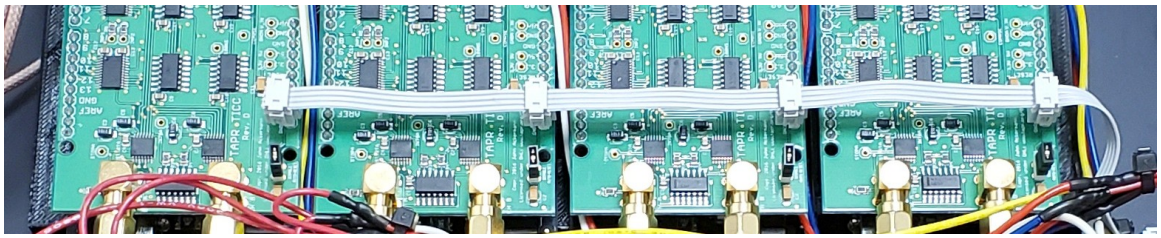
The carrier design files (and all files referenced in this document) are available from the TICC github repository (https://github.com/TAPR/TICC).

There are three points of electrical interconnection between the boards in a multi-TICC system.



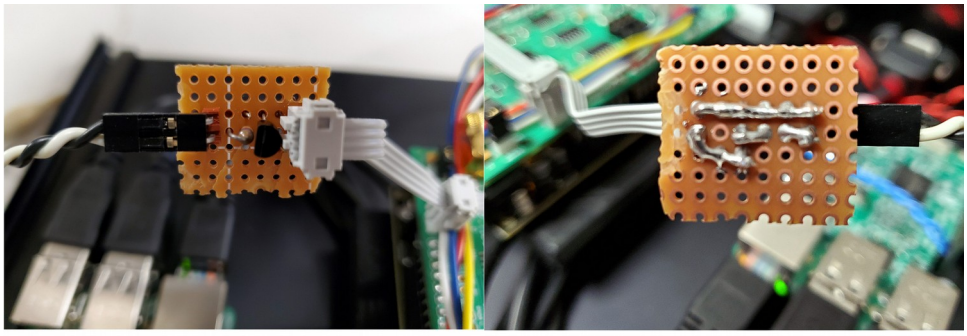
1. As shown in the photo above, the 10 MHz clock is applied to all boards through short coax cables and “tee” connectors. The “TERM” termination jumper should be installed **only on the last board in the chain.**

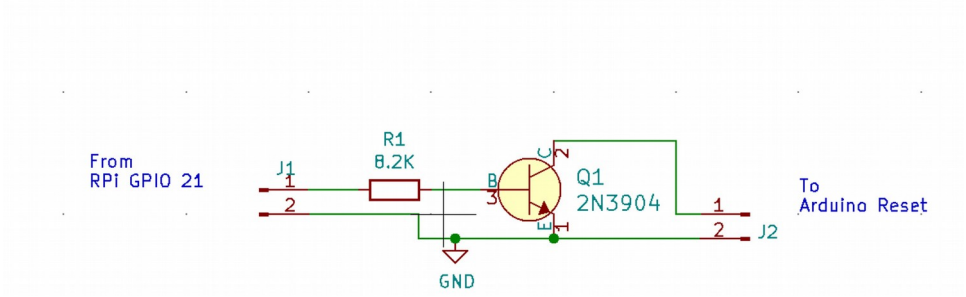
2. Also as shown above, the three pins of the JP2 header are also connected in a daisy-chain fashion across all the boards. The easiest way to make this cable is to use 0.1 inch spaced IDC ribbon cable connectors with 6 positions – just make sure you plug the same row of the connector onto each JP2!. If you use the 4-board frame described below, the nominal spacing between JP2 headers is 2.25 inches.



3. Connect the RESET lines of all the boards in parallel as shown above, so that they can be reset simultaneously. To do that, solder a 2 pin header in the holes marked “GNDB” and “RESET” located on the right front of the board, and made a daisy-chain connector tying them together. Again, it’s easiest to do this with a 4 pin ribbon cable. One note: with the reset lines in parallel, if one of the TICCs is powered down, it will pull the reset line of the others low with the result that they will not start. Make sure all the TICCs are powered up when the reset daisychain is connected.

4. If using a Raspberry Pi as the host computer one of the Pi’s GPIO pins can be used to cause the Arduinos to reset via software command. The problem is that the Arduino uses 5V logic and the Raspberry Pi is 3.3V. The very simple level translator shown below can be used to interface between the two boards. It’s easy to assemble this on a small piece of perf board.






5. Install shorting blocks on the “DISABLE AUTO-RESET” header (JP1) on each board. If this is not done, each time a connection is made to the Arduino serial port, that TICC will reset

and that will cause synchronization problems. With this jumper installed, you will not be able to upload new firmware without performing tricks. The multi-ticc_updater.py program discussed below does the appropriate tricks and you can use it to update firmware even if the DISABLE AUTO-RESET header is shorted. If you don't use that tool, remember to remove the jumper block before programming!

6. On "slave" boards, remove IC10 (the 12F675 PIC). This is because the coarse clock signal generated by the PIC on the master board is routed via JP2 to the slave boards.

With these changes, your set of boards is ready to emerge as a multi-TICC!

## Firmware Configuration

The firmware shipped on recent TICC units (version 20170309.1) has basic multi-TICC capability included. A newer version (20191202.1) is available in the github.com:/TAPR/TICC repository that adds three helpful features for multi-TICC use:

First, you can assign channel IDs other than A and B.

Second, you can use a "PROP_DELAY" configuration variable to set an offset value in picoseconds for each channel which will be added to the values reported from that channel. This allows you to compensate for the length of the interconnect cables and other sources of board-to-board delay. (PROP_DELAY serves the same purpose as the existing "FUDGE0" variable, but I've been requested to make two delay settings available. The PROP_DELAY and FUDGE0 settings are additive.)

Third, a bug fix enables an external LED to show that the board is being clocked. If the LED is attached to the last board in the string, it will give you some assurance that the board interconnections are correct. The LED can be attached to the A11 pin on the right rear area of the board. (External channel activity LEDs can be connected to the A12 and A13 pins as well.)

To enable multi-TICC operation, use the configuration "Y" command to set the master board as "M"aster, and to set the slave boards to "S"lave mode. Optionally, use the "N" command to change the channel ID to any single printable ASCII character. Since the master board by default has channels A and B, it's sensible to set the second board to "C D", the third to "E F" and so on. Finally, if desired use the PROP_DELAY or FUDGE0 variables to set the offset in picoseconds. Note that the Arduino serial monitor program is a bit funky when it comes to data input; you'll probably have better luck if you use a "real" serial terminal program to set configuration parameters.

## Data Interface and Host Processor

If you have 4 TICCs configured configured in master/slave mode, you also have 4 USB ports carrying serial data. It would be nice to consolidate those data streams, and maybe even make them available via Ethernet. A Raspberry Pi is an excellent tool to provide this function – its four USB ports are a perfect match for a four-board multi-TICC configuration. I've written a simple Python TCP server program to serve as a data aggregator spitting all the TICC data out over a telnet connection.

If the TICC system is configured as described above, you will be able to connect and disconnect input signals at any time without disturbing the logging of data from other channels. And, when an idle channel starts logging pulses, it will do so on the same time-scale as the other channels. This means that the TICC system can be kept running across many measurement cycles, and measurements from multiple channels can be matched in sequence by their timestamps.

In order to keep this document reasonably short, and also because the code is still being revised, I won't go into a lot of details, but here are the basics on the main program as well as some other programs that make managing the system easier. At some point when things have had some time to be debugged, I will create a ready-to-run SD card image so all you'll need to do is plug that in.

Described in a text file located with the programs is a description of software prerequisites that need to be installed to use the software, as well as some other setup information. It describes how to persistently assign serial port names dev/ttyTICC0, dev/ttyTICC1, dev/dev/ttyTICC2, and dev/ttyTICC3 to the four devices. Those names are used in all the programs described here, and the assumption below is that the system contains four TICC units.

`multi-ticc_server.py` – if run with no options provided will look for four TICC units on connected to the Raspberry Pi serial ports. It outputs several streams of data on different TCP ports, by default:

port 9190: Outputs multiplexed data from all active TICC channels, unsorted
port 9191: Outputs multiplexed data from all active TICC channels, sorted by timestamp
port 9192: Outputs data from chA
port 9193: Outputs data from chB
port 9194: Outputs data from chC
port 9195: Outputs data from chD
port 9196: Outputs data from chE
port 9197: Outputs data from chF
port 9198: Outputs data from chG
port 9199: Outputs data from chH

There are many tools you can use to connect to the server from another machine and access the data. The easiest is with the telnet command:

```
telnet <server IP address> 9190
```

To make it easier to log the data to a file, you can use the Linux netcat program:

```
nc <server IP address> 9192 > datafile.dat
```

`multi-ticc_server.py` can be left running on a console and will show startup and connection status. Unfortunately, at this point the program can only support one client connection per data stream, but I'm hoping to add multi-connection support.
Here are some other utility programs that make managing the multi-TICC easier:

`multi-ticc_reset.py` will momentarily send Raspberry Pi GPIO pin 21 high, which if the level translator circuit described above is installed, will cause the TICCs to reset. Run this after starting the Raspberry Pi and before attempting to connect to the TICCs; on startup the TICCs normally freeze and kicking them with this program will cause a clean restart with all boards synchronized.

`multi-ticc_updater.py` will update all TICCs with a new firmware file in hex format. Supply the file path and name as a command-line argument.

`miniterm.sh` is a shell script that will launch a simple terminal program allowing you to communicate with one TICC. Just give the TICC serial port name as a command line argument.

## Miscellaneous

If you want to put your multi-TICC in a nice enclosure, I've designed a complete 2U rack enclosure using the Front Panel Express design tools. The files are available at github.com/TAPR/TICC/multi-ticc/