6-16-2021

Self-Contained Photon Coincidence Counting with NI myRIO Ecosystem

Georges Oates Larsen
Portland State University

Andres H. La Rosa
Portland State University

Self-Contained Photon Coincidence Counting with NI myRIO ecosystem

by

Georges Oates Larsen

An undergraduate honors thesis submitted in partial fulfillment of the

requirements for the degree of

Bachelor of Arts/Science

in

University Honors

and

Mathematics

and

Physics

Thesis Advisor

Andres H. La Rosa

Portland State University

2021

Self-Contained Photon Coincidence Counting with National Instruments myRIO Ecosystem

G. Oates Larsen[1] and A. H. La Rosa[1]
Portland State University

(Dated: 22 June 2021)

Digital coincidence counting units (CCU) have made experimental verification of fundamental quantum mechanical principles financially accessible to undergraduate level teaching programs. However, recent implementations of these systems are not easily ported to National Instruments (NI) FPGAs, making them unsuitable for physics departments that have heavily invested in the NI ecosystem. Therefore, there is clear need for a detailed implementation based on an NI FPGA. We present a formal description of one such implementation, based on the NI myRIO (NI's lower-cost, student-oriented offering) which achieves 6.9 ns minimum guaranteed-distinguishable delay and 32.2 MHz peak coincidence counting rate with four input channels and simultaneous monitoring of all possible coincidence types.

I. INTRODUCTION

Coincidence counting units (CCUs) are experimental apparatus that can be used to perform relatively simple experimental validations of the most basic principles of quantum mechanics, including the particle/wave duality of light, and Bell's inequality, by detecting the simultaneous arrival of individual photons on separate input channels.[1] The input channels used are referred to as single photon detectors (SPDs).

CCUs based on digital signal processing techniques are significantly less expensive than their analog counterparts.[2] However, older implementations have relied upon discrete digital integrated circuits, and were therefore bulky, complicated to assemble, and fixed in design.[2] Newer designs based instead on low-cost field programmable gate arrays (FPGAs) eliminate the need for digital logic to be assembled physically, enabling more sophisticated CCU logic that can be reconfigured on a whim.[2,3]

However, FPGA programming is not easily transferable between competing FPGA platforms. This is often due to the use of platform-specific features or proprietary development toolchains. For instance, the recent implementation of a low-cost CCU based on an Altera brand FPGA by the Mark Beck group at Reed College is not transferable at all to another FPGA ecosystems due to its use of the platform-specific "ECO Fit" feature to accomplish pulse shortening.[1,3]

This problem is especially pronounced for the National Instruments (NI) FPGA ecosystem. NI brand FPGAs force the use of the proprietary visual programming language, LabVIEW, which uses a completely different paradigmn than the low-level hardware description languages commonly used by other FPGA ecosystems.[4–6] In fact, LabVIEW enforces a number of constraints directly preventing the typical approach to pulse lengthening on an FPGA, thus requiring a totally new approach.

The Nano Optics Group at Portland State University, like many others, is invested heavily in the NI ecosystem. Therefore, it is of direct interest to us to develop an approach employing an NI FPGA. Such an approach is presented here.

Additionally, many SPD units, including our own, cannot be operated off the shelf, having card edge connectors or exposed electronics that require an enclosure for safe operation.[7,8] Therefore, in contrast with the implementation of the Beck group,[5] and in congruence with the implementation of Han et al.,[6] we have opted to directly integrate the SPD unit, FPGA, and power electronics into a single housing. We present a full description of the assembly and electronics involved.

II. BACKGROUND

The reader may be unfamiliar with a number of topics herein referenced. This section serves as a brief introduction to these topics.

A. CCU Theory of Operation and Prior Implementations

Coincidence counting units monitor the detections of an array of single photon detectors, sensitive devices capable of detecting single incident photons. If several SPDs are triggered simultaneously (within a small number of nanoseconds), this is known as a coincidence.[5] CCUs detect and count these coincidences. Advanced CCUs can distinguish between all possible coincidence types (all possible combinations of channels).[2]

1. Analog CCU Implementations

Older, and more expensive implementations of CCU rely on nanosecond-precision analog timing circuits that measure the time difference between detections on separate SPD channels, and report a coincidence if this time difference is ever below a threshold, known as the coincidence window. These timing circuits are expensive, and can only monitor a single pair of channels, making the monitoring of a large number of channel combinations prohibitively complex and expensive.[2]

2. Discrete Digital Implementations

A discrete digital CCU, rather than directly measure the time difference between channel pulses, treats each channel

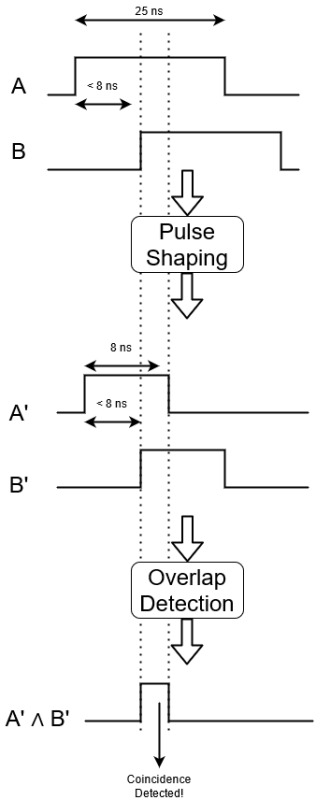

FIG. 1. An example of pulse shaping using 25 ns input pulse length and 8 ns coincidence window. A and B are raw pulse signals from an SPD. A' and B' are pulse-shortened versions of A and B respectively. $A' \wedge B'$ is the result of the application of an AND gate to A' and B', and its momentary HIGH output at the intersection of A' and B' indicates the detection of a coincidence.

as a digital signal to be processed using discrete digital logic circuits.

Digital implementations first perform what is known as pulse shaping, wherein the raw detection pulses provided by the SPDs are modified to have a consistent length, equal to the coincidence window.

These shaped signals are then sent to an AND gate, which outputs a high signal when all of its input signals are high. If two detections on two seperate SPDs are closer to one-another than the coincidence window, this will cause their shaped pulses to overlap, which the AND gate will detect as a coincidence.[2] See figure 1 for a visual example of pulse shaping.

Discrete digital implementations which use pulse shaping rather than attempt to directly track the time deltas between detections are simpler and far less expensive than their precision analog counterparts. However, discrete digital circuits still need to be manually assembled, and so are fixed in design once built, and furthermore still limited by the practicality of their assembly, albeit to a lesser extent.[2]

B. FPGA-based CCU Implementations

Field programmable gate arrays are devices that allow digital logic to be authored digitally and flashed (programmed) rather than physically assembled. FPGAs are a natural choice for the implementation of experimental apparatus such as CCUs since the implemented digital logic can be updated on a whim as the needs of a particular experiment evolve, while the lack of physical assembly allows for much more complex designs, and hence more sophisticated digital signal processing.[3,5]

1. Clocking

FPGAs are not without their own complexities. Digital logic is assembled at run time on an FPGA by enabling and disabling pre-built connections between an on-silicon array of pre-built generalized logic units.[9] These silicon units are small, and do not have perfectly consistent timing,[10] and the connections formed between them can be quite lengthy or complex. If allowed to respond to input signals immediately, these logic units would very quickly end up out of sync. To overcome this shortcoming, all logic units must be driven by a synchronizing clock, so that they will only respond to the current value of supplied inputs on the rising edge of each clock cycle. This keeps the behavior of all logic units across the FPGA silicon consistent, allows time for signals to fully propagate through the silicon, and simplifies timing design considerations for complex designs.[9]

2. Synthesis

The process of generating the final firmware sent to an FPGA (known as the bitstream) is called synthesis.[9] Due to the complexity of synthesis, it has largely been automated. This automation comes in two major forms: Logic synthesis and high level synthesis (HLS).[11]

Logic synthesis converts a semi-low-level hardware description into a logic-gate level representation that is easily translated into an FPGA bitstream. The antecedent description is typically at the register transfer level (RTL), wherein circuits are described as an array of hardware value registers and associated gates operating synchronously on said values. RTL descriptions are most commonly authored in the Verilog or VHDL hardware description languages (HDL).[9]

HLS, instead, takes as its input a high-level programming language such as LabVIEW or C++, and synthesizes this into an RTL description that can be fed to typical logic synthesis.[11]

3. National Instruments vs. Altera

The chosen FPGA platform greatly impacts the tools and processes available for logic authoring. These differences are, in fact, a key motivation for this research.

Of particular relevance are the differences between the Altera FPGA platform, used by the Beck research team, and the National Instruments FPGA platform, preferred by our own. Whereas Altera provides authoring software (Quartus) that can sythesize both from VHDL and Verilog, NI lock their FPGA motherboards down so that only their proprietary visual programming language, LabVIEW, can be used and synthesized into bitstreams. This makes the programs written in VHDL by the Beck team completely incompatible with the high-level, dataflow-oriented approach required by NI FPGAs.[4,5]

Furthermore, Quartus has a special synthesis feature known as "ECO Fit" which the Beck team uses to implement pulse shaping.[5] No such feature exists within LabVIEW, and so pulse shaping must be implemented explicitly.

C. Single Photon Avalanche Diodes

Single photon avalanche diodes (SPADs) are a form of avalanche photodiode (APD) commonly used as a single photon detector. This is the case with our implementation.

1. Theory of Operation

SPADs operate by establishing a strong reverse bias voltage across a semiconductor photodiode. The exact voltage used varies by SPAD, but in order to achieve single-photon sensitivity, it must be above what is known as the breakdown voltage of the underlying photodiode, typically on the order of several tens of volts.[12] Excelitas Technologies, the manufacturer of our chosen SPAD (SPCM-AQ4C) does not specify the employed bias voltage.

Normally, the SPAD does not react to this bias voltage. A depletion region with no free-moving electrons is formed, through which no electron flow (current) can occur.

When a photon strikes the depletion region of the photodiode, it can be absorbed by a trapped electron, thereby exciting it and making it free to move, converting it into what is known as a carrier, in that it can literally carry potential energy stored in the bias voltage. Once freed, the carrier is accelerated by the bias voltage. The bias voltage is sufficiently strong that even a single freed carrier gains enough kinetic energy to interact with other trapped electrons and excite them into carriers as well. These new carriers can then excite other electrons, resulting in an exponential increase in total carriers known as an avalanche. This will continue until the bias voltage is sufficiently depleted to prevent further carrier production.[12]

While the initial current contribution from the first carrier is not measurable, the resulting avalanche certainly is. By watching for this avalanche signal, individual photons can fairly reliably be detected.[13]

2. Quantum Efficiency

Not all incident photons successfully create a carrier electron, and not all carrier electrons successfully initiate an avalanche. The percentage of incident photons on a SPAD which successfully result in an avalanche is known as the quantum efficiency (or, alternatively, the photon detection efficiency) of the SPAD. It is worth note that quantum efficiency can vary with wavelength.[7]

3. Afterpulsing

All SPADs, to one degree or another, exhibit a behavior known as afterpulsing. During an avalanche, a substantial amount of current is released across the junction of the diode. A small percentage of carriers can become temporarily trapped on the way out, and released several tens of microseconds later. If these trapped-and-released carriers trigger another avalanche (and hence, a spurious second pulse), the phenomenon is known as an after-pulse.

The afterpulsing probability specifies the probability that any given avalanche will have an afterpulse.[14]

4. Dark Counts

SPADs also exhibit a behavior known as dark counts. This is a form of spurious signal which occurs spontaneously without any incident light on the sensor. While there are a number of underlying phenomena credited with creating dark counts, the resultant behavior is identical; the SPAD will exhibit a baseline rate of spurious detections (typically at a rate of several hundred per second) regardless of whether there is incident light.[15] The SPCM-AQ4C has a manufacturer-guaranteed upper-bound dark count rate of 500 counts per second.[7]

5. Gating

To counteract afterpulsing and dark counts, many commercial off the shelf SPAD units support a feature known as gating.

Essentially, gating allows an external signal to quickly and precisely enable and disable the bias voltage on the SPAD. The SPAD is only able to avalanche while there is a high bias voltage applied.

Therefore, if the gating signal can be kept perfectly in sync with the arrival of the light being analyzed, only the analyzed light will be able to generate an avalanche. By keeping the gate open for a period shorter than the afterpulsing delay, and then closed until after the afterpulsing delay has elapsed, most afterpulses can be ignored. This will also reduce dark counts, as the APD will only be sensitive to them for the very short amount of time (typically on the order of less than a microsecond) required to detect the expected pulses of light.[14]

Such timing is only possible with periodic pulsed light sources, and typically the same signal generating the light pulses will be used as the gating signal.

6. Dead Time

SPADs cannot detect photons continuously. After any detection event, there is a short period of time during which the SPAD cannot detect another photon. This is known as the dead time of the SPAD. This creates an upper bound for the maximum count rate for SPADs, although the actual maximum count rate may vary.[16] The SPCM-AQ4C has a dead time of 50 ns, as well as a maximum count rate of 5 MHz burst and 2 MHz continuous.[7]

7. Power Considerations

In spite of the relatively small energy contributions made by triggering photons, the resultant avalanches result in a significant amount of current (on the order of tens of milliamps), and therefore a significant amount of power (on the order of several watts) being released as thermal energy within the SPAD silicon at high count rates.[7] Consequently, SPADs must be carefully thermally managed, and care must be taken not to exceed a count rate that would cause the SPAD to overheat. The datasheet for the given SPAD should be consulted to determine these limits.

III. IMPLEMENTATION

We implement a single-enclosure (monolithic), single photon coincidence counting unit utilizing an NI myRIO-1900 FPGA and Excelitas Technology SPCM-AQ4C four channel SPAD, and associated support electronics.

A. System Description

1. Overview

The CCU implementation consists primarily of a four-channel SPAD, a driving FPGA, and a variety of minimal required support electronics, enclosed in a single aluminum housing. Figure 2 shows a high-level diagram of the CCU. The chosen FPGA is the NI myRIO-1900, selected for its relatively low cost and undergraduate-oriented durability. This FPGA motherboard features a Xilinx Zynq-7010 FPGA.[17]

The chosen SPAD is the Excelitas Technology SPCM-AQ4C, a four-channel SPAD with an upper bound of 500 dark counts per second, 50 ns dead time, 25 ns pulse width, 0.5% afterpulsing probability, and a peak quantum efficiency of roughly 60% at roughly 700 nm.[7] This SPAD has exposed electronics and is electronically interfaced with a card-edge connector. Therefore, support electronics and an enclosure are necessary for safe operation.

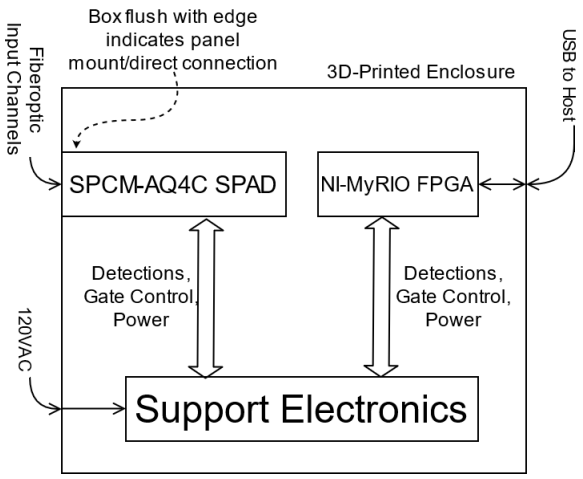

FIG. 2. A high-level diagram of the CCU.

FPGA, SPAD, and support electronics are packed into a single enclosure. Ideally, this enclosure would be metallic. However, due to time constraints and long manufacturer lead times, a plastic surrogate had to be 3D-printed. Future work will aim to transition to a metal enclosure.

2. Support Electronics

Ideally, one would implement all support electronics on a single, purpose-built printed circuit board (PCB) featuring a card edge connector receptacle for the SPAD, directly integrated power electronics, and a direct parallel connector directly attachable to the FPGA. However, the design overhead and prototyping costs were deemed to outweigh the potential size and integration benefits of such an approach.

Instead, we have opted to utilize the Exelitas Technologies SPCM-AQ4C-IO, a breakout board providing a power breakout connector, card edge connector receptacle, and BNC style ports for all detection signals and gate controls.

The SPCM-AQ4C-IO is mounted with 3D-printed brackets to a perfboard (a type of generic prototyping PCB), which we use to perform basic interconnect wiring and power supply implementations.

Custom 50-ohm terminations (figure 3) are, as required by the SPCM-AQ4C-IO, attached to each SPAD pulse output channel, and then routed to the FPGA digital IO channels via the perfboard. An additional ground-only BNC breakout is used to connect the ground signal of the SPCM-AQ4C-IO to one of the DGND reference ground pins of the FPGA. Without this connection, the FPGA cannot reliably sample the pulse channels.

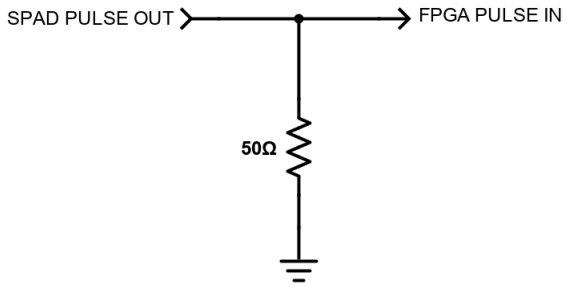

FIG. 3. Electrical diagram of the 50 ohm termination required by the SPCM-AQ4C-IO.

The SPCM-AQ4C-IO is configured to use a single gating signal for all channels. The gate signal is provided directly via internal BNC cable connected to the backside of a BNC port at the front of the CCU.

According to Excelitas, the SPCM-AQ4C requires three distinct supply voltages:[7,8]

- 30 VDC must be supplied with at least 0.3 A current capacity.
- 5 VDC must be supplied with at least 1 A current capacity.
- Isolated 2 VDC must be supplied with at least 2 A current capacity.

Additionally, the NI myRIO-1900 requires external DC power. While the exact voltage range and minimum current are not specified by NI, the power supply the myRIO ships with provides 12 VDC and is rated for 1.5 A. Out of caution, we take 1.5 A as the lower-bound supply specification for the myRIO.

To generate these voltages, we start with mains voltage, supplied via an International Electrotechnical Commission (IEC) C14 receptacle with 1 A fuse and switch on the back of the unit.

This mains is routed to two AC to DC power supplies: A Mean Well RS-25-12, 2.1 A 12 VDC supply to power the FPGA, and a CFM40C300-DR 1.33 A 30 VDC to provide one of the supplies required by the SPAD. This supply also powers the supplies responsible for the other two power inputs required by the SPAD.

The necessary current rating for the 2 VDC isolated supply required by the SPAD is quite high. Isolated DC to DC converters with this current and voltage range simply do not exist on the market at the time of writing. However, 5 VDC isolated supplies in this current range are much more common, and so we pair one (PQDE6W-Q24-S5-T) with a non-isolated DC to DC step-down converter (OKX-T/5-W5P-C) to generate the desired high-current, isolated 2 VDC.

The 2 VDC supply is adjustable, and so a simple trimming circuit is required to achieve exactly 2 VDC. This circuit is diagrammed in figure 4.

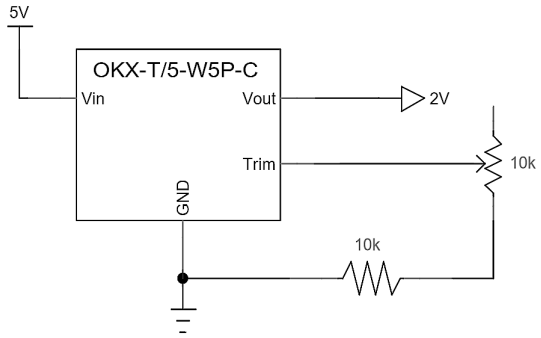

FIG. 4. Electrical diagram of the trimming circuit used to achieve exactly 2V. The 10k potentiometer is adjusted manually until 2V is reached, then fixed using adhesive.

For simplicity, we use an additional PQDE6W-Q24-S5-T to generate the final, non-isolated 5 VDC power supply required by the SPAD. The SPCM-AQ4C-IO internally shorts the output ground of this supply to the output ground of the 30 VDC supply making it functionally non-isolated.

Figure 5 shows a component-level diagram of the power and data electronics.

B. Physical Layout

The externals of the CCU consist of a 3D-printed enclosure (printed in two pieces secured together with screws), 3D-printed front and back panel, and various panel-mount components, namely the gating signal BNC input, the IEC C14 power input with switch and fuse, a USB port for FPGA interfacing, and the SPCM-AQ4C SPAD itself.

Internally, the SPCM-AQ4C-IO and perfboard are mounted to each other with a pair of identical 3D-printed brackets. Together they form the main electronics chassis. The 3D brackets are sized so that the main electronics chassis fits snugly in the main enclosure, but can slide along the forward axis. The main chassis is never affixed to the main enclosure, but rather, held in place by the card-edge connection between the SPCM-AQ4C and SPCM-AQ4C-IO. Once this connection is made, the main electronics chassis also serves to support the back-end of the SPCM-AQ4C.

The 5 VDC and 2 VDC power supplies are directly affixed to the perfboard (screwed and soldered, respectively). The 12 VDC and 30 VDC supplies are affixed directly to the main chassis with Velcro. Given the completely-self-contained nature of these two supplies, there is no risk of an electrical short if they should become internally detached, and given the strength of the Velcro, this is highly unlikely to begin with. Hence, Velcro was deemed an acceptable solution for mounting them.

The NI myRIO is friction fit into a gap between the main electronics chassis brackets and the wall of the main enclosure. This further locks the entire assembly into place.

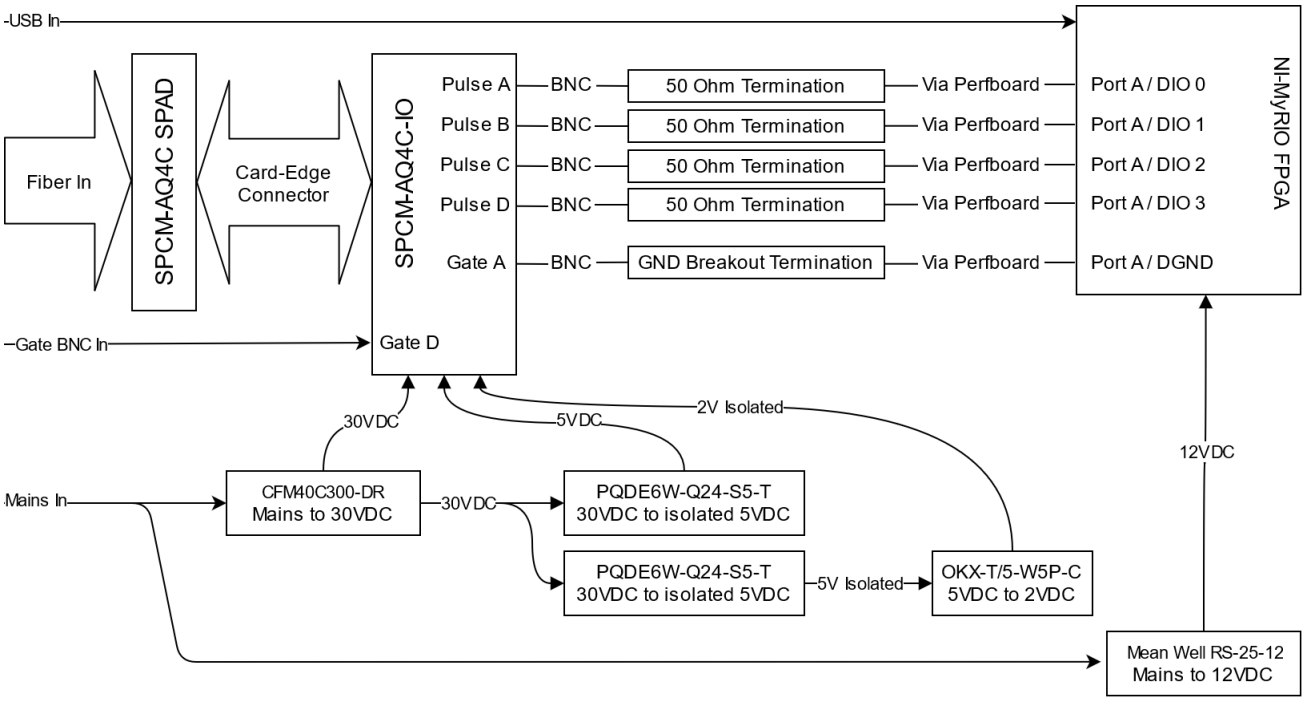

FIG. 5. Component level diagram of CCU signal and power electronics.

All 3D-printed parts have threaded inserts pushed into them using a soldering iron. This allows them to be screwed into the various components they attach to. All inserts are M3. Various lengths of screw are used.

1. *Parts List*

These are the key components used in our implementation. This list excludes the menagerie of miscellaneous crimping inserts, wires, connectors, terminals, ferrules, cables, screws, 3D-printed parts and threaded inserts used. Many of said components came from kits and do not have part numbers. However, the assembly section details the relevant information wherever possible.

- (1x) National Instruments myRIO-1900 (FPGA)
- (1x) Excelitas Technology SPCM-AQ4C (Four Channel SPAD)
- (1x) Excelitas Technology SPCM-AQ4C-IO (Interface Module for SPCM-AQ4C)
- (1x) CFM40C300-DR (30V, 40W, ACDC power supply)
- (1x) Mean Well RS-25-12 (12V, 25.2W, ACDC power supply)
- (2x) PQDE6W-Q24-S5-T (5V, 6W, DCDC isolated power supply)
- (1x) OKX-T/5-W5P-C (0.8V to 3.6V, 17W, DCDC adjustable power supply)

As previously mentioned, our implementation uses a 3D-printed enclosure. However, we recommend that prospective implementers purchase a Rose Enclosures 07504011 aluminum enclosure. Our 3D-printed enclosure is designed to mostly match up with this enclosure. Some internal arrangements, especially the 3D-printed brackets, may need to be adjusted to better fit the aluminum enclosure. In the event that a prospective implementer chooses identical SPAD, FPGA and power supply units, CAD files for 3D-printed parts are available upon request. Please note, however, that these designs will not fit the metal enclosure without adjustments. Figure 6 shows a picture of the 3D-printed enclosure used.

C. Assembly

First, mounting holes for 3D-printed brackets and 5 VDC power supplies are drilled into the perfboard. Next, the 2 VDC power supply and trimming circuit are soldered onto the perfboard. A 4-conductor JST receptacle and a 34-pin insulation displacement connector (IDC) receptacle are also soldered to the perfboard and connected to each other. These are used to transfer detection pulses to the FPGA. Finally, a terminal block is added for connecting the SPCM-AQ4C-IO and myRIO ground signals. Figure 7 shows the fully assembled perfboard with electronics and power wires (described in the next step) attached.

Next, wires are cut, crimped, zip-tied, and affixed to all of the power supplies in the power supply chain. Crimp types include spade connectors, fork connectors and ferrules. The preassembled HiRose HRS DF7-6S-3.96C cable supplied with the SPCM-AQ4C-IO is also used. Figure 8 shows the full

FIG. 6. Picture of the 3D-printed enclosure used by this implementation.

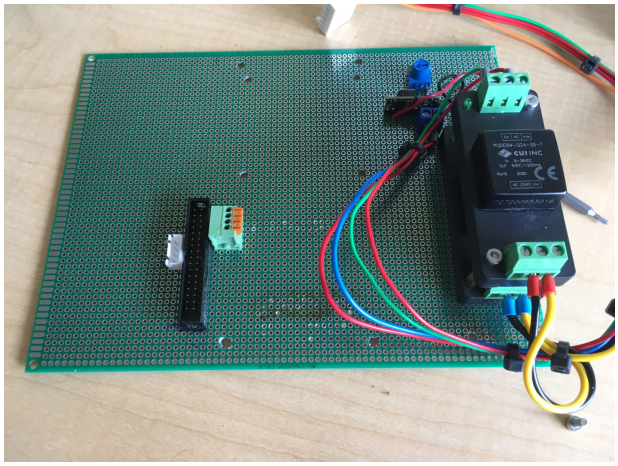

FIG. 7. Picture of the perfboard with all components soldered or affixed, and all power wires inserted.

power supply chain and wiring.

At this point, a quick-power-on test is performed, and the 2 VDC trimmer is adjusted until 2 VDC is achieved. The trimmer is then fixed in place with hot glue. Figure 9 shows the 2 VDC circuit and fixed trimmer.

Meanwhile, the four 50 ohm BNC terminations, are assembled and crimped to a 4-pin JST connector. These terminations consist of a screw-terminal BNC breakout, a single solid core wire, and a 50 ohm resistor. The solid core wire is crimped to the resistor using a ferrule, which is inserted into the signal channel of the BNC breakout. The other side of the resistor is crimped to another ferrule which is inserted into the ground channel of the BNC breakout. Clear heat shrink is applied over the whole assembly to prevent shorts and keep it secure. Additionally, the ground-only BNC termination is created. This consists simply of a solid core wire crimped

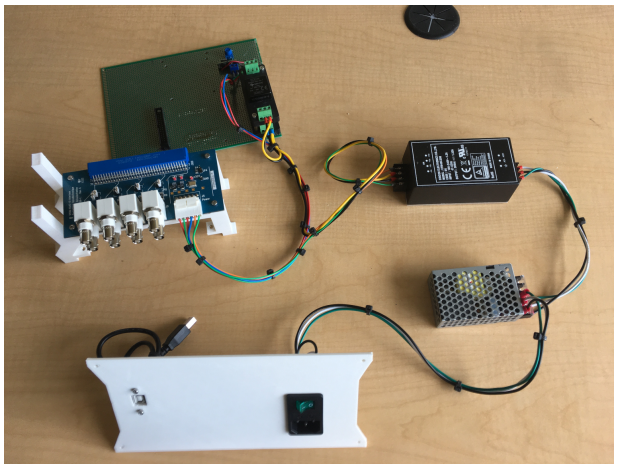

FIG. 8. The fully wired power chain. Clockwise from middle left: SPCM-AQ4C-IO, perfboard, 30 VDC power supply, 12 VDC power supply, back panel with IEC C14 port and USB port. HiRose connector is inserted into the SPCM-AQ4C-IO.

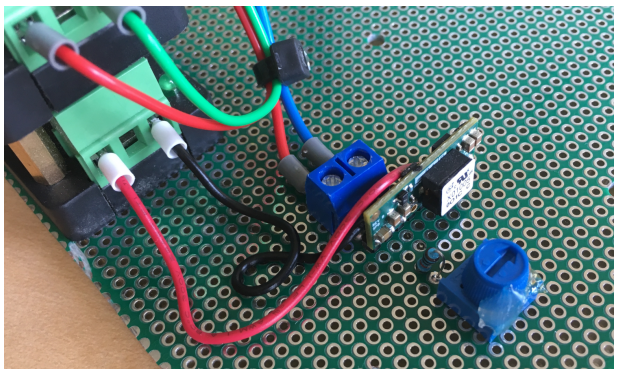

FIG. 9. A close view of the 2 VDC power circuit, with voltage adjust trimmer fixed in place with hot glue.

to a ferrule and screwed into the ground terminal of a BNC breakout. Figure 10 shows the terminations in detail. Note, however, that the HiRose connector and perfboard that should be present by this stage are missing from the picture.

These five BNC terminations are now assembled onto the SPCM-AQ4C-IO, which in turn is screwed onto the perfboard, and configured for single-gate mode.[8] Figure 11 shows the fuse configuration for the SPCM-AQ4C-IO that implements single-gated mode. The JST connector is inserted into the JST receptacle, and the ground termination is inserted into the DGND terminal block. This completes the main electronics chassis. Figure 12 shows a picture of the fully assembled main electronics chassis. Note, however, that the HiRose connector that should be inserted by this stage is left freestanding in the picture.

Now, the 12 VDC and 30 VDC power supplies are fed through the back of the enclosure to the front, where they are affixed with Velcro. This is pictured in figure 13.

Meanwhile, the SPAD and BNC gating input are affixed to the 3D-printed front panel (see figure 14), and a BNC cable is affixed to the internal side of the gating input (see figure

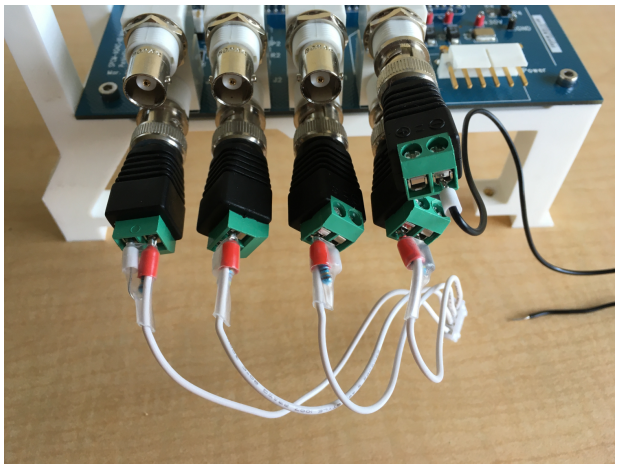

FIG. 10. A close shot of the 50 ohm and GND-only BNC terminations mounted to the SPCM-AQ4C-IO. Bottom row are 50 ohm terminations. Upper right is GND-only termination.

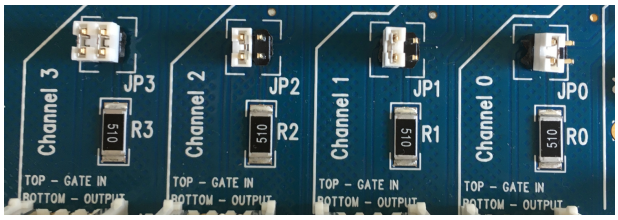

FIG. 11. Picture of the fuse arrangement on the SPCM-AQ4C-IO that enables single-gated mode.

15). This cable is run through the front of the enclosure to the back. Once this is complete, the front panel is affixed to the enclosure.

Next, all power cables and the gating control cable are fed through the interior of the main electronics chassis, carefully routed to avoid interference between the cables and chassis

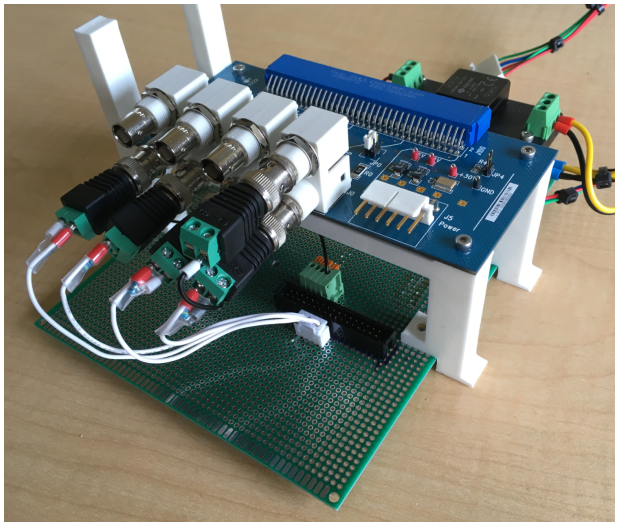

FIG. 12. Picture of the fully assembled main electronics chassis.

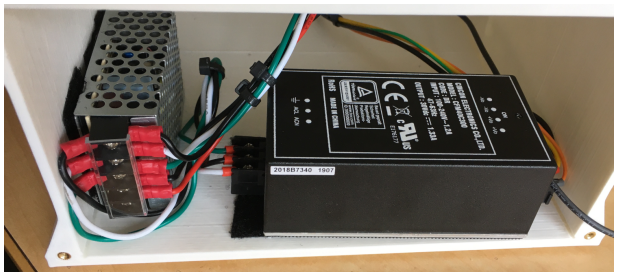

FIG. 13. Front view of the enclosure with 12 VDC and 30 VDC power supplies internally affixed.

FIG. 14. Picture of the front panel, showing SPCM-AQ4C and gate signal BNC input affixed.

assembly, and the main electronics chassis is partially inserted into the enclosure. This is pictured in figure 16.

It is now possible to connect the HiRose power connector and the gating control BNC cable to the SPCM-AQ4C-IO. This is pictured in figure 17.

Once this is complete, a 34-pin IDC cable is plugged into the perfboard, and the main electronics chassis is carefully slid toward the front-end of the enclosure, until it begins to make contact with the card edge connector. Care must be taken to avoid interference between the wiring, chassis assembly, and card-edge connector, and the slot for the FPGA must be kept

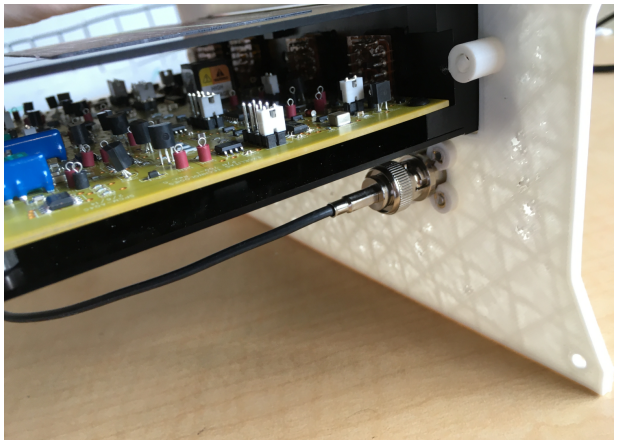

FIG. 15. Picture of the reverse side of the front panel, showing SPCM-AQ4C and a BNC cable attached to the internal side of the gating signal input.

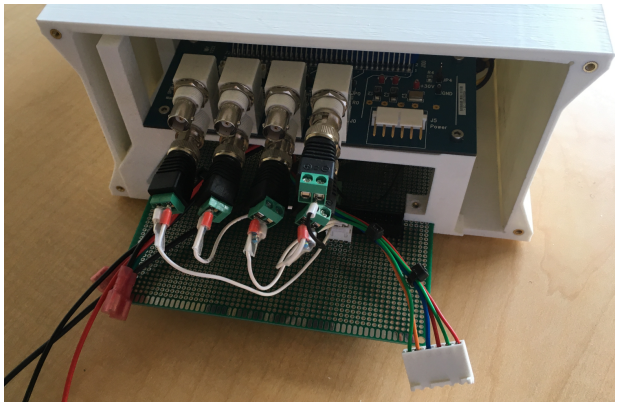

FIG. 16. Rear view of enclosure showing partially inserted main electronics chassis with cables from the front passed through to back.

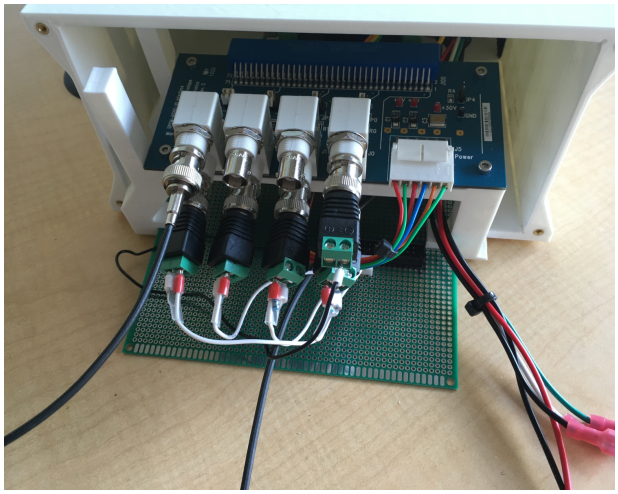

FIG. 17. Rear view of enclosure showing partially inserted main electronics chassis with cables from the front passed through to back and HiRose power connector and BNC gating signal attached to the SPCM-AQ4C-IO.

clear.

After double-checking visually that there is no physical interference between the wiring and chassis assembly, and that the card-edge connector is aligned and free of obstruction, a firm push on the main electronics chassis causes the card edge connector to mate. The SPCM-AQ4C-IO and SPCM-AQ4C are now connected.

Meanwhile, the back panel is assembled. The IEC C14 connector and USB panel mount connector are affixed to the back panel, and the IEC C14 port is wired so that the switch is in series with mains.

The other end of the IDC cable from the main electronics chassis, the 12 VDC barrel jack from the front power supplies, and the USB cable from the back panel are now inserted into the FPGA. This is pictured in figure 18. Note that the IDC cable must be inserted into connector A of the FPGA.

The FPGA is now friction-fit into its slot (figure 19).

Finally, the spade connectors for mains are attached to the IEC C14 connector (see figure 20), and the back panel is affixed to the rear of the enclosure (see figure 21).

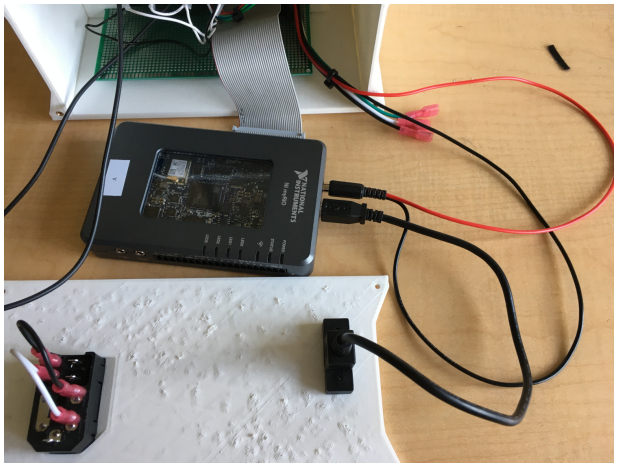

FIG. 18. Rear view of the implementation after main electronics assembly is fully inserted and 34-pin IDC, power, and USB cables are attached to the FPGA. IEC C14 receptacle backside is visible in lower left.

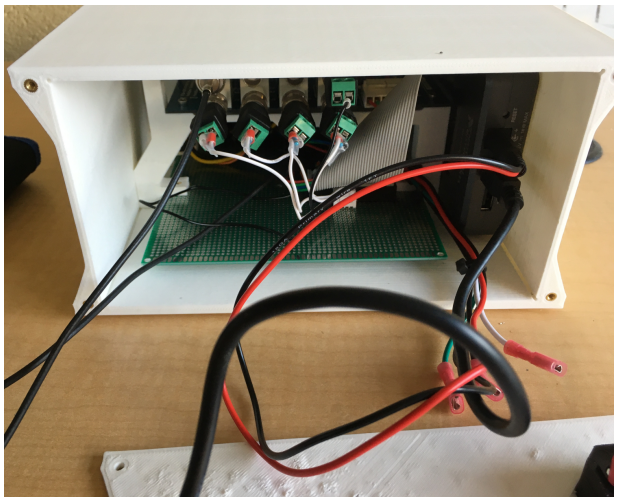

FIG. 19. Rear view of implementation, showing the fully inserted FPGA.

The unit can now be plugged into an IEC C13 power cable, and connected to a host computer via USB cable to be programmed and operated.

D. Programming in LabVIEW

The precision of an FPGA-based CCU is, ultimately, driven by the resolution at which signal pulse shortening can be performed. On non-LabVIEW FPGAs, it is common to implement pulse shortening by passing the input signal and a marginally phase-shifted copy of itself through an AND gate with single inverted input or an XOR gate.[2,6] This results in pulses whose length is exactly equal to the phase shift amount. Thus, the shortest achievable pulse length (and, hence, best achievable CCU precision) is directly determined by the min-

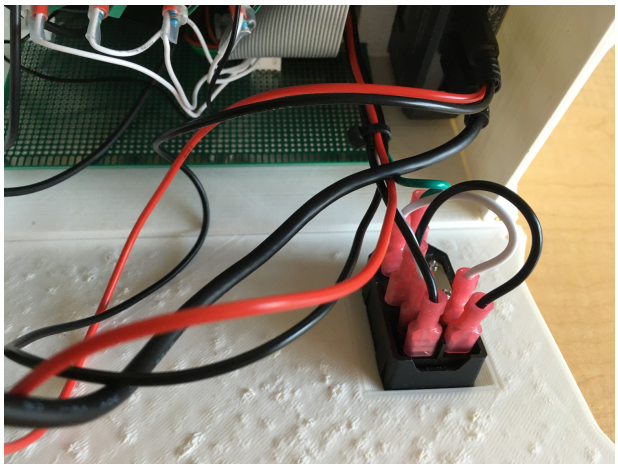

FIG. 20. Close view of the affixed IEC C14 port with mains wires attached with spade connectors.

FIG. 21. Picture of the affixed back panel of the implementation.

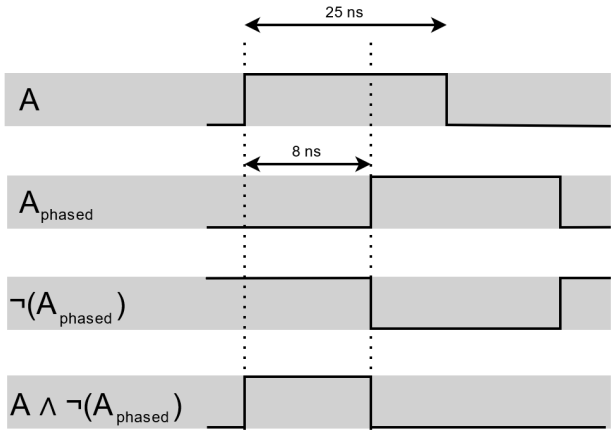

FIG. 22. An example of phase-shift-based pulse shaping. A 25 ns input signal (A) is shaped into an 8 ns output signal ($A \wedge \neg\left(A_{\text{phased}}\right)$). First, A is phase-shifted by 8 ns (the desired output pulse length) to produce A_{phased}. Next, A_{phased} is inverted to produce $\neg A_{\text{phased}}$. Finally, A is intersected with $\neg A_{\text{phased}}$ to produce the final shaped pulse. Note that this method can only shorten pulses. Pulse lengthening is not possible.

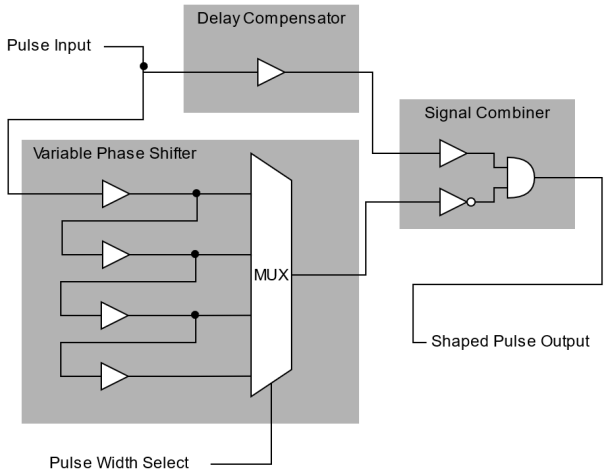

FIG. 23. An example of buffer-and-multiplexer phase-shift-based pulse shaping with functional blocks highlighted in grey. Raw pulses enter the variable phase shifter, and must pass through an increasing number of buffers to reach each successive input of the MUX. Each buffer adds delay so that each input to the MUX is phase shifted by the buffer delay relative to the previous input. Thus, the MUX can be used to select a desired phase shift from these four inputs. The delay compensator buffer counteracts the propagation delay of the MUX itself. The signal combiner intersects the compensated raw pulse with an inverted version of the selected phase-shifted pulse to produce the final shaped pulse, as per figure 22.

imum reliable phase shift achievable with the FPGA. Figure 22 gives an example of phase-shift-based pulse shaping.

In a non-LabVIEW FPGA programming environment, it is possible to directly use buffer elements to incur phase shifts on signals (arising from the propagation time of the buffers themselves). With careful attention to clock propagation, this will result in the minimum phase shift possible on the FPGA. The outputs of an array of cascading buffers can be fed to a single multiplexer to implement a dynamically-configured phase shift[5,6]. A diagram of this approach is given in figure 23.

However, in a LabVIEW FPGA environment, none of this is possible. LabVIEW is a data-oriented programming language based on a concept referred to as dataflow, and so can only describe circuits which transform data in a manner which is congruent with a high-level dataflow program.[4] While concurrency is certainly supported, non-synchronous circuits and single-buffer delays such as those used by Beck and Han are outright impossible to implement in LabVIEW.

In fact, the only way to achieve any form of precision timing in LabVIEW is through the use of single cycle timed loops (SCTL). Any code placed inside an SCTL is guaranteed to be executed in lockstep with a specified clock signal from the FPGA.[18] While this allows much greater timing precision than is otherwise possible in LabVIEW (since the enclosed code has an associated guaranteed frequency that can be made to approach the upper limits of the FPGA), it comes with the very significant drawback that LabVIEW requires the entire

portion of code within the loop to complete within the driving clock period. Furthermore, input digital signals are sampled at the clock frequency, rather than piped directly to the gates that receive them. In other words, the entirety of the code within the SCTL must be synchronous, forcing all phase shifts to be a multiple of the driving clock frequency and making buffer-based phase shifts completely impossible.

In short, any CCU implemented in LabVIEW is forced to operate by sampling the digital input at a fixed frequency that is directly linked to the total execution time of the coincidence counting logic, and this sampling rate will be the absolute minimum achievable phase shift for any such implementation. Hence, when working with an NI FPGA, maximizing the driving clock frequency becomes a critical concern for maximizing performance.

The execution time of an SCTL can often be reduced, without limiting code complexity, by using a technique known as pipelining, wherein partially processed data is passed to the next iteration of the SCTL (using a register or feedback node of some kind) to be processed concurrently with the input data for the next cycle.[19] However, registers and feedback nodes are not without delay of their own, and hence, it is not possible to reduce the execution time of an SCTL to any less than that of a single node and its input/output feedback nodes. LabVIEW also inserts hidden circuitry to enforce dataflow known as the enable chain, further increasing the minimum execution time of an SCTL containing complex programs.[4] While some NI platforms allow the enable chain to be disabled, the myRIO platform is not among them. Finally, the RTL synthesis engine itself may choose to include extra buffers if a single output is routed to a large number of inputs (this is called fanout) even further contributing to the minimum execution time for complex circuits.[9]

In general, then, having the minimum phase shift limited to the driving clock period is a major drawback to the LabVIEW platform, and results in significant reductions to the upper bound performance of a potential CCU. Fortunately, in spite of these setbacks, we have found an implementation in LabVIEW which performs comparably the Beck implementations (which employed a much older FPGA). This is mostly due to the significant difference in overall FPGA performance between our chosen FPGA and the FPGA employed by Beck et al. As we will demonstrate in the validation section, our implementation is significantly slower than the Non-LabVIEW implementation by Han et al.

Given the restriction imposed by LabVIEW that phase shifts last an entire driving clock period, there is significantly less impetus to pursue the buffer-and-multiplexer approach of Han. Instead, we have opted to use a numeric solution based on DSP48E1 digital signal processing (DSP) slices, a specialized, 48-bit logic and arithmetic element that can in theory execute relatively complex mathematical operations with total execution times that are similar to single standard logic elements.[20] All counting logic is placed within a single SCTL. In practice, we find these units to be somewhat slower than single logic elements, meaning it may be possible to get higher clock speeds by using purely traditional logic. But, for the purposes of this implementation, we found the DSP

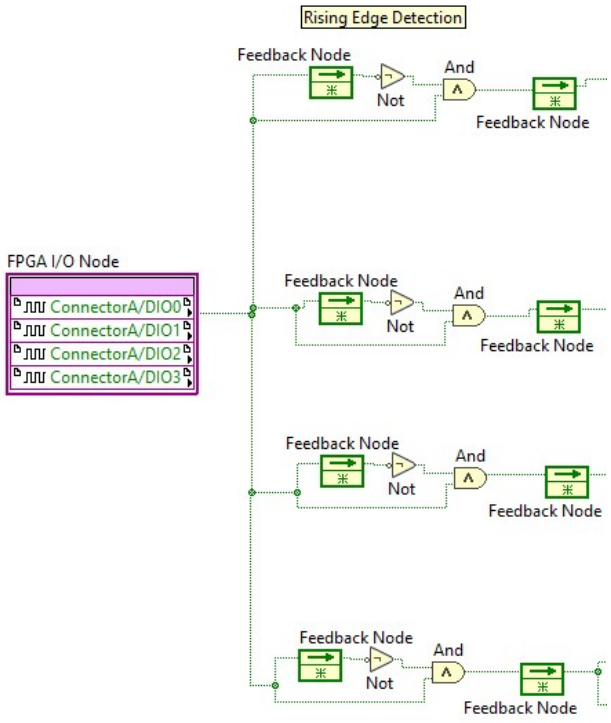

FIG. 24. LabVIEW rising edge detection program operating on DIO0 through DIO3 inputs from connector A. Each input is passed to a feed-forward node (labeled by LabVIEW as a feedback node) in series with a NOT gate, the output of which is passed to an AND gate along with the original signal. The result is an output which is HIGH if and only if the input is currently HIGH, but was LOW the previous clock cycle. The output is finally fed to a pipelining feed-forward node.

slices to be adequate. The use of these slices also comes with some benefits. The coincidence totals and pulse shaping logic are all 48-bit, allowing an extremely large range of coincidence windows and extremely high total coincidence count to be achieved.

The numeric method we employ works as follows: First, rising edge detection is implemented on all digital inputs. Every clock cycle, the previous value of the digital input is compared to the current value. If they differ and the latest value is HIGH, then a HIGH output is generated, exclusively for the clock cycle on which this transition occurred. This is pictured in figure 24.

All four channels have a negative-counting accumulator with a negative-number pattern detector implemented using one DSP slice each. Whenever a pulse rising edge is detected, the DSP slice is reset, and one cycle later, the coincidence window value is numerically added to the current accumulator value. Every cycle, the accumulator value decreases by 1. The pattern detect feature of the DSP slice is configured to output a HIGH value whenever the accumulator value is negative. These configurations cause the output of the pattern detect to be LOW for $N+1$ clock cycles after every pulse rising edge,

where N is the configured coincidence window. We invert this signal and treat it as the shaped pulse. Adjusting the (48-bit) coincidence window value thus directly alters the pulse length generated by this pulse shaping program. The minimum possible coincidence window is zero, corresponding to an actual pulse length of one. The pulse shaping program is pictured in figure 25.

Rather than send the shaped signals directly to the coincidence detection logic, we first perform a process we refer to as coincidence latching. Coincidence latching eliminates the double-counting of various coincidence types. For instance, without coincidence latching, two simultaneous pulses on channels A and B would result in a count for A, a count for B, and a count for the AB coincidence counter. However, with coincidence latching, only the AB coincidence counter is incremented.

Coincidence latching works by passing each shaped signal to a latch which latches HIGH whenever its assigned channel outputs HIGH, but resets back to LOW when and only when all four input channels are LOW. All latches reset simultaneously when this condition is met. Whenever the latches reset, their value immediately before the reset is treated as a 4 bit integer representing the coincidence type, and this integer is passed off to the counting program.

Suppose, for example, a pulse occurs on channel A. This will latch the channel A latch to HIGH. Consider, then, a pulse which occurs shortly after on channel B. If the delay between these pulses is greater than the coincidence window, then the shaped A signal will go LOW before the shaped B signal goes HIGH. Therefore, for a brief moment, all signals will be LOW, and the latches will reset, treating A as an individual A count. Shortly after, the B pulse will count as an individual B count as well, as would be expected. However, if the delay is less than the coincidence window, the shaped B signal will go HIGH before the shaped A signal goes LOW, and there will not be a moment when all shaped signals are LOW simultaneously. Thus, the A and B latches will both be HIGH when the latches reset (after the conditioned B signal goes LOW), and this will count as an AB coincidence.

Figure 26 shows the pattern latching program.

The counting program performs a direct equality check on all integers between 0 and 15, and instructs the corresponding counter (implemented using another DSP slice) to increment whenever the equality holds true. In practice, the zero coincidence is never actually triggered; all-latches-LOW is the latch reset state, and in this state, the integer 16 is passed along as the coincidence type, preventing any zero counts from occurring.

The coincidence counting logic also features a simple cycle counter for accurate time measurement, and a RESET register, which can be used to reset all counters, including the cycle counter. In future revisions, the non-functional zero-type coincidence counter and the cycle counter will be combined to use a counter.

Finally, registers are used to copy count values off to a lower-frequency loop, where they are made available to the front-panel. Directly writing the values to the front panel would significantly hinder maximum clock speed. The coincidence type counters, cycle counter, RESET register read, and counter register writes are all pictured in figure 27. The low-frequency loop used to copy register values to and from the front-panel is pictured in figure 28.

The full LabVIEW project is available on request.

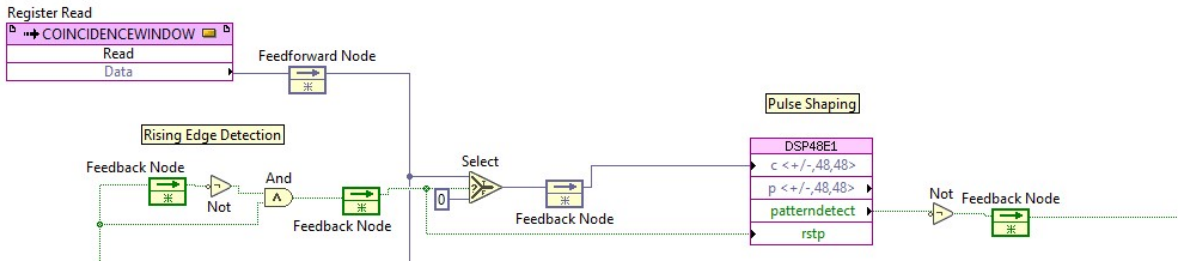

FIG. 25. Single channel of LabVIEW rising edge detection (left) and DSP48E1-based pulse shaping (right) program, featuring coincidence window input register in the upper-left. Rising edge detection is the same as pictured in figure 24. Raw input pulses are fed to the rising edge detection circuit, which produces rising edge ticks that reset the DSP slice and feed its c register, via a feedback register and one cycle later, with the coincidence window on every pulse rising edge. The pattern detect output on the DSP slice is configured to output true if the accumulator value is negative. Thus, after every rising edge, the patterndetect output of the DSP slice will be LOW for exactly $N+1$ cycles, where N is the coincidence window cycle, and will be high at all other times. This signal is negated to produce a shaped version of the input pulse, and then fed via a pipelining feed forward node (labeled as a feedback node) to the pattern latch pictured in figure 26. Note that this pulse shaping program can both shorten and lengthen pulses. The exact configuration of the DSP slice is pictured in figure 32 in appendix A.

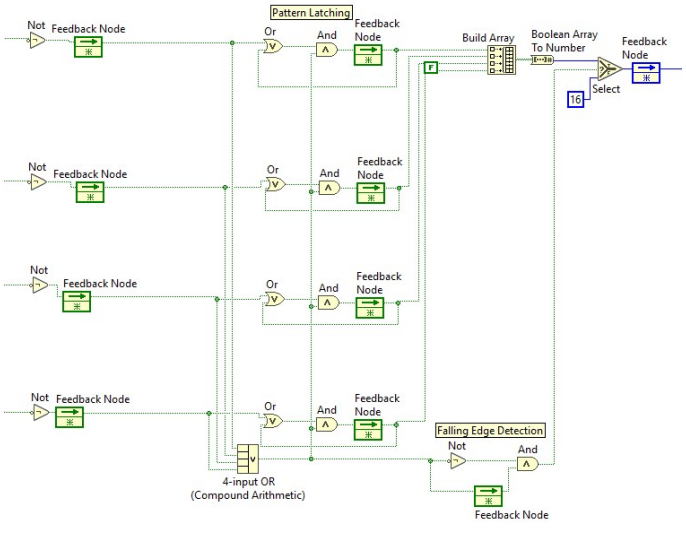

FIG. 26. LabVIEW pattern latching program. Shaped pulse outputs from figure 25 are fed in from the left to an array of four latches, each consisting of an OR gate, an AND gate, and a feedback node. Each latch will go high if its corresponding shaped pulse input goes high. So long as any of the four shaped inputs is high, the latches will retain their value. On the exact cycle this condition is violated, all four latches will reset, and their values just prior to reset will be converted to an integer and output via a pipeling feed-forward node to the counters pictured in figure 27.

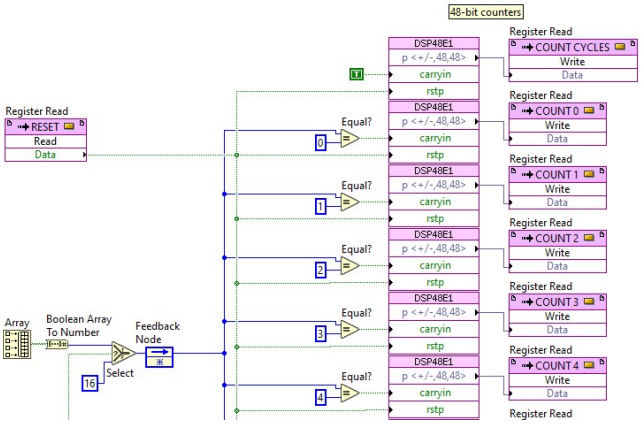

FIG. 27. The first five counters of the LabVIEW counting program. Whenever the coincidence latch (pictured in figure 26, and partly visible here) detects a coincidence, a 4-bit coincidence number is passed for a single clock cycle to all 16 counting channels, 5 of which are visible here. Each channel checks the equality of the sent number with a distinct pre-programmed value. In the event of a match, the corresponding counter, implemented using a DSP48E1 slice, is incremented by one. The outputs of each counter are written to a corresponding register after every cycle to be retrieved later for the front-end. This retrieval process is pictured in figure 28. Also pictured is the cycle counter, which increments on every cycle without regard to any coincidence. The full configuration for the DSP slice used here is available in figure 33 in appendix B.

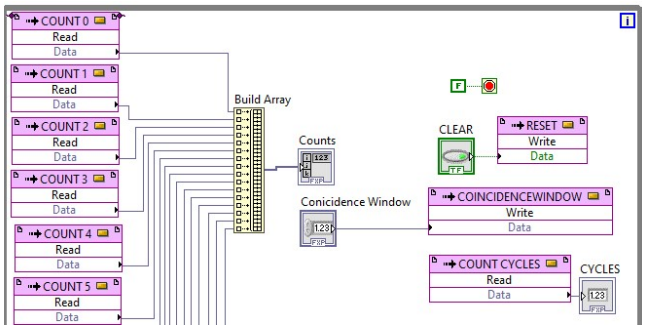

FIG. 28. Picture of the low-frequency loop used to copy register values to and from the high-frequency SCTL, featuring a basic while loop, 16 coincidence count register reads (the first 6 of which are pictured), which are aggregated into a single array available on the front panel, a coincidence window control and register write, a CLEAR/RESET control and register write, and finally a cycle counter readout. The while loop operates continuously and is not frequency-limited.

IV. RESULTS & VALIDATION

A. Dark Counts

As mentioned earlier, SPADs have a baseline spurious count rate which is present even in absolute darkness. These dark counts should be detected by a successful CCU implementation, and the measured dark count rate should be on approximately the same order of magnitude as the listed maximum dark count rate provided by the manufacturer. We do, indeed, detect dark counts when our implementation is powered on, at a rate of between 360 and 410 counts per second, which is comfortably close to the quoted upper bound dark count rate of 500 counts per second.[7] A nominal dark count rate is not listed, but the fact that the measured rate is the same order of magnitude as the stated upper bound is an excellent sign.

B. Minimum Guaranteed-Distinguishable Pulse Delay

The completed program is able to compile with a driving clock frequency for the main SCTL of up to 280 MHz. In theory, this corresponds to a sample interval of approximately 3.5 ns, or equivalently, a minimum guaranteed-distinguishable delay (which we will now abbreviate as MGDD) of twice that value, or about 7 ns. In other words, theoretically, our implementation is guaranteed to be able to distinguish two pulses that are more than approximately 7 ns apart.

This theoretical value seems to correlate well with the achieved performance of the Beck research team, who claim a 7 ns to 8 ns timing resolution.[5] While an exact definition of timing resolution is not given by the Beck team in this paper, it is reasonable to assume MGDD was the intended meaning. The Han research team, meanwhile, achieved an MGDD of 0.46 ns.[6]

MGDD can be measured by feeding two precision square waves to the CCU, and performing a scan of phase shift vs. coincidence probability with the CCU. The MGDD will be the difference in time between the smallest phase shift which guarantees no coincidences are counted, and the largest phase shift which guarantees all pulses are counted as a coincidence. Performing this scan at a variety of coincidence window values improves its reliability.

We perform precisely this test, using another myRIO FPGA to generate the precision square waves. Unfortunately, our square wave generator has extremely similar limitations to our CCU in that it has a maximum clock frequency of 290 MHz, and therefore a phase shift and pulse length resolution of only about 3.5 ns. Therefore, our measurements are not nearly as precise as the Han team, which had a 10 ps timing resolution for their square waves,[6] but they do at least establish a solid lower bound on our true performance, and provide a viable sanity check to the theoretical MGDD. The myRIO-to-myRIO test setup is pictured in figure 29.

Figure 30 shows a set of 11 scans of phase shift vs. percent chance of coincidence, each for a distinct coincidence window. Each scan ranged from a phase shift of about 3 ns to about 70 ns, incrementing by the minimum phase shift our test signal generator is capable of. Each data point is the result of a full second of feeding two square waves with this phase shift to two channels of the CCU. Both square waves had a frequency of 1 MHz for all samples. As would be expected, the transition point moves right as the coincidence window increases. Importantly, the transition from 100% coincidence to 0% coincidence is completed, without fail, within a span of two samples, indicating that the MGDD is, at most, 6.9 ns, or about 15 times that achieved by Han et al. This is exactly in line with the theoretical prediciton for the achieved clock frequency, and indicates the CCU is operating as expected.

FIG. 29. Picture of the myRIO-to-myRIO configuration used to characterize the CCU performance. DIO lines from test signal generator are connected directly across to DIO lines for CCU FPGA. DGND lines are also connected.

C. Maximum Pulse Frequency

The maximum coincidence count rate of the CCU can be characterized by performing a simple frequency sweep with two identical 50% duty cycle square waves fed to two of the FPGA pulse input channels. The coincidences detected per second should equal the frequency of the square waves. The point at which this equality fails is the maximum coincidence count rate. Han et al. achieve a maximum coincidence count count rate of about 400 MHz.

Figure 31 shows the graph of a frequency vs coincidence rate sweep from about 300 kHz to 280 MHz. 1000 samples were collected, with each sample corresponding to a different minimum decrement of the test signal period. Values were obtained by allowing the CCU and test signal to run for one second at the given target frequency, and then dividing the total observed coincidences by the total observed clock cycles, multiplying by a scaling factor to convert from coincidences per cycle into coincidences per second. The test signal frequency and coincidence counting rate are identical until 32.2 MHz, after which the coincidence counts drop to zero. In other words, the maximum coincidence count rate for our implementation is 32.2 MHz, about one 12th of that of Han et al. Once again, the precision limitations for the test signal generator severely limit the accuracy to which we can test the maximum coincidence count rate. This is especially pronounced

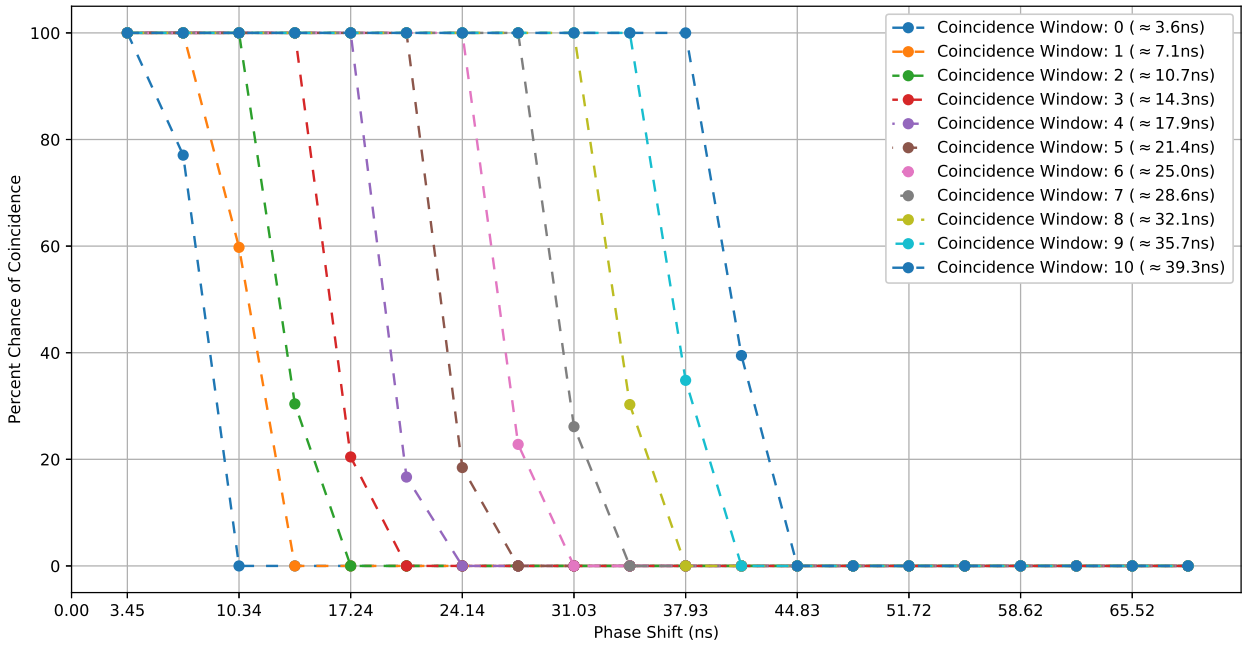

FIG. 30. 11 plots of probability of coincidence vs. phase shift for 11 different coincidence windows.

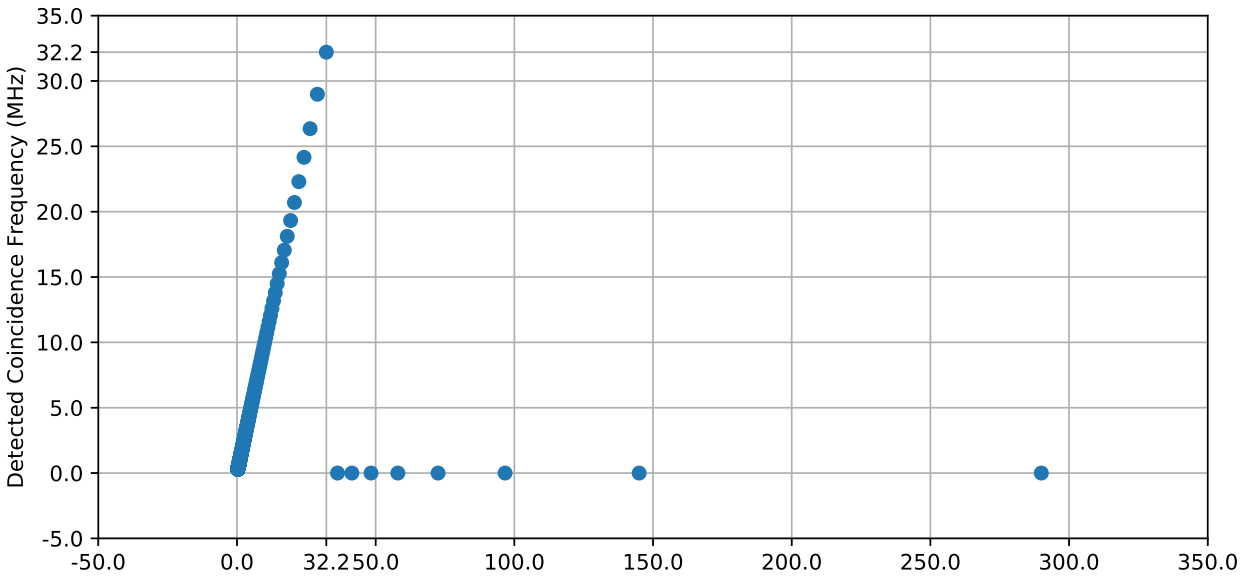

FIG. 31. A plot of test signal frequency vs. coincidence detection rate for the CCU.

near 290 MHz, as the achievable target frequencies get more spread out. A more precise test signal generator might significantly increase the estimated upper bound coincidence counting rate. At the very least, however, we have a lower bound for the upper bound performance.

V. CONCLUSION

We have successfully implemented and demonstrated a National Instruments version of Mark Beck's CCU implementation, overcoming a great deal of technical challenges mainly imposed by the LabVIEW FPGA programming environment. While this implementation is not state of the art, it achieves comparable minimum guaranteed-distinguishable delay to that of Beck et al., and is therefore adequate for implementing quantum experiment curriculum. We have also

presented the full electrical and construction details of our implementation, so that the reader is able to completely recreate the implementation if so desired.

In future work we hope to make the transition to a metal enclosure, acquire more accurate test data with higher-precision signal generators, implement and demonstrate some of the CCU experiments authored by the Beck research lab, and investigate ways to further improve on the limitations imposed by the LabVIEW FPGA programming environment.

It is our hope that this work enables other research teams who are heavily invested in the NI ecosystem to pursue FPGA-based CCU and the associated undergraduate quantum experiment curriculum created by Mark Beck.

ACKNOWLEDGMENTS

I would like to thank Andres La Rosa for his continued support and guidance. Through thick and thin, Andres has kept me involved in the lab, kept me in the loop, and done his best to help me succeed. His unrelenting support has been a rock for me during hard times.

On behalf of the Nano Optics Group, I would also like to thank Mark Beck and his research team for their instrumental work on bringing low-cost, FPGA-based CCUs to undergraduates, without which this research would not have been possible, or even conceived.

Special thanks are due to the 2020 Portland State University Faculty Development Grant, which enabled us to purchase the SPCM-AQ4C used in this research.

Appendix A: Pulse Shaping DSP Slice Configuration

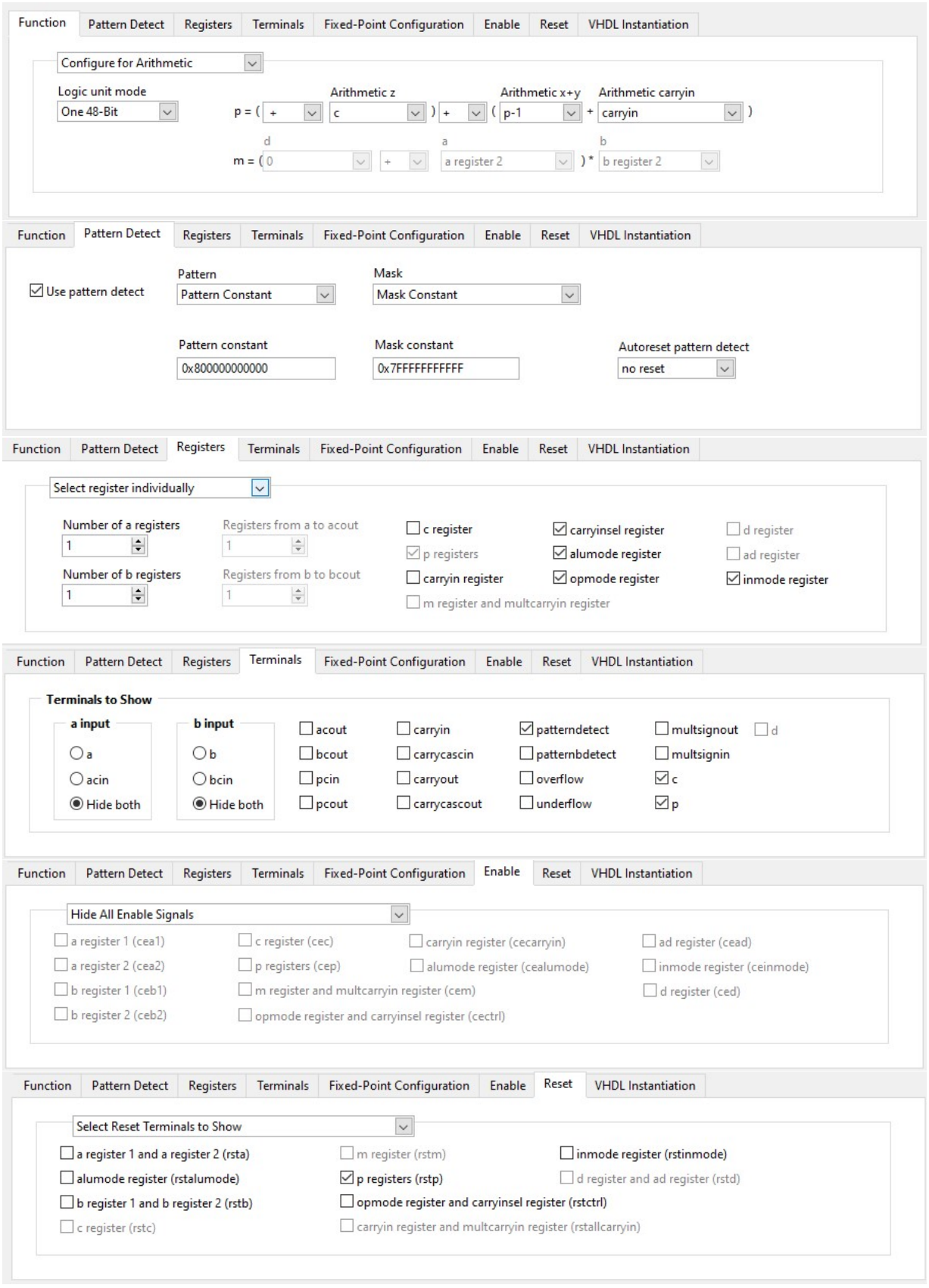

FIG. 32. Full configuration for the DSP slice used for pulse shaping.

Appendix B: Counter DSP Slice Configuration

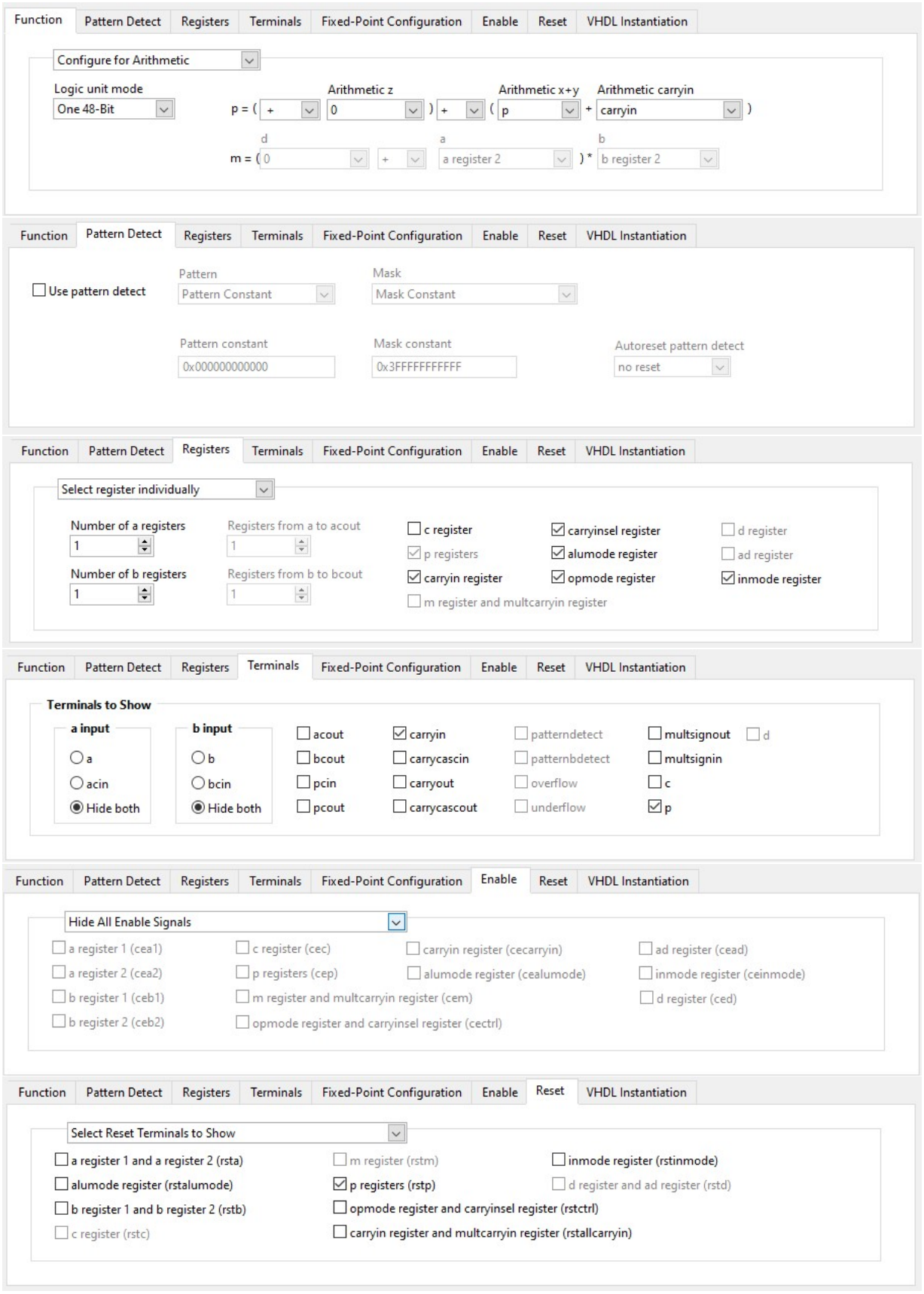

FIG. 33. Full configuration for the DSP slice used for all counters.

[1]M. Beck, "*Modern Undergraduate Quantum Mechanics Experiments*," (2018), http://people.reed.edu/~beckm/QM/.

[2]D. Branning, S. Bhandari, and M. Beck, "Low-cost coincidence counting electronics for undergraduate quantum optics," American Journal of Physics **77,** (2009).

[3]M. Beck, "Coincidence counting units," (2018), http://people.reed.edu/~beckm/QM/circuit/circuit.html.

[4]"Dataflow and the enable chain in FPGA VIs (FPGA module)," (2018), https://zone.ni.com/reference/en-XX/help/371599P-01/lvfpgaconcepts/fpga_sctl_and_enablechain/.

[5]J. W. Lord and M. Beck, "Coincidence counting unit using the Altera DE2," Available for download at http://people.reed.edu/~beckm/QM/circuit/circuit.html.

[6]B. Park, Y. Kim, Y. Cho, and S. Han, "Arbitrary configurable 20-channel coincidence counting unit for multi-qubit quantum experiment," Electronics **10** (2021).

[7]"SPCM-AQ4C datasheet," (2020), https://www.excelitas.com/file-download/download/public/61201?filename=Excelitas_SPCM-AQ4C_datasheet.pdf.

[8]"SPCM-AQ4C-IO datasheet," (2020), https://media.digikey.com/pdf/Data\%20Sheets/Excelitas\%20PDFs/SPCM-AQ4C-IO_UG.pdf.

[9]J. Serrano, "Introduction to FPGA design," in *CERN Accelerator School: Course on Digital Signal Processing* (2008) pp. 231–247.

[10]S. Nassif, "Design for variability in DSM technologies," (IEEE, 2000).

[11]J. Cong, B. Lie, S. Neuendorffer, and J. Noguera, "High-level synthesis for FPGAs: From prototyping to deployment," (IEEE, 2011) pp. 473–491.

[12]F. Zappa, A. Tosi, and S. Cova, "Principles and features of single-photon avalanche diode arrays," Sensors and Actuators A: Physical **140**, 103–112 (2007).

[13]M. Kinch, "A theoretical model for the HgCdTe electron avalanche photodiode," Journal of Electronic Materials **37** (2008).

[14]A. Ruggeri, P. Ciccarella, F. Villa, F. Zappa, and A. Tosi, "Integrated circuit for subnanosecond gating of InGaAs/InP SPAD," IEEE Journal of Quantum Electronics **51** (2015).

[15]Y. Xu, P. Xiang, and X. Xie, "Comprehensive understanding of dark count mechanisms of single-photon avalanche diodes fabricated in deep sub-micron CMOS technologies," Solid-State Electronics **129** (2017).

[16]L. Neri, S. Tudisco, F. Musemeci, A. Scordino, G. Fallica, M. Mazzillo, and M. Zimbone, "Dead time of single photon avalanche diodes," in *Nuclear Physics B - Proceedings Supplements*, Vol. 215 (2011) pp. 291–293.

[17]"User guide and specifications: NI myRIO-1900," https://www.ni.com/pdf/manuals/376047c.pdf.

[18]"Timed loop (single-cycle)," (2018), https://zone.ni.com/reference/en-XX/help/371599P-01/lvfpga/fpga_timed_loop/.

[19]"Optimizing FPGA VIs using pipelining (FPGA module)," (2018), https://zone.ni.com/reference/en-XX/help/371599P-01/lvfpgaconcepts/fpga_pipelining/.

[20]"Zynq-7000 SoC: DC and AC switching characteristics," (2020), https://www.xilinx.com/support/documentation/data_sheets/ds187-XC7Z010-XC7Z020-Data-Sheet.pdf.