Qubitekk CC1 Handheld Coincidence Counter

Qubitekk

August 14, 2014

Contents

Chapter 1

Principle of Operation

1.1 Counters

The CC1 has three counters in its internal programming, one for each input channel and one for the coincidences. These counters are tied to the input SMA cables, which use the 3.3V LVTTL standard. The channel counters increase by one on the positive edge of the input greater than 0.8 V.

These counters go between 0 and 2,097,152. If a counter goes above 2,097,152, it will be set to -1 until it is cleared. On the LCD screen, this is indicated by the word "overflow".

1.2 Gate Input

The gate input SMA connector operates on a 2.5 V voltage level, however like the channel SMA inputs it registers as high at voltage levels above 0.8 V.

When the gate is enabled, the counters count the intersection of this gate and the input signal, as seen in 1.2.

1.3 Coincidence Window

The coincidence window calculation operates by generating a pulse of the width of the specified length at each positive edge of the detection signals for both channels, and then sends the intersection of these signals to the coincidence counter, as shown in 1.3.

If the coincidence window is set to 0 ns, the transformed signals will be at constant low.

If two pulses on the same channel arrive within the same coincidence window, the

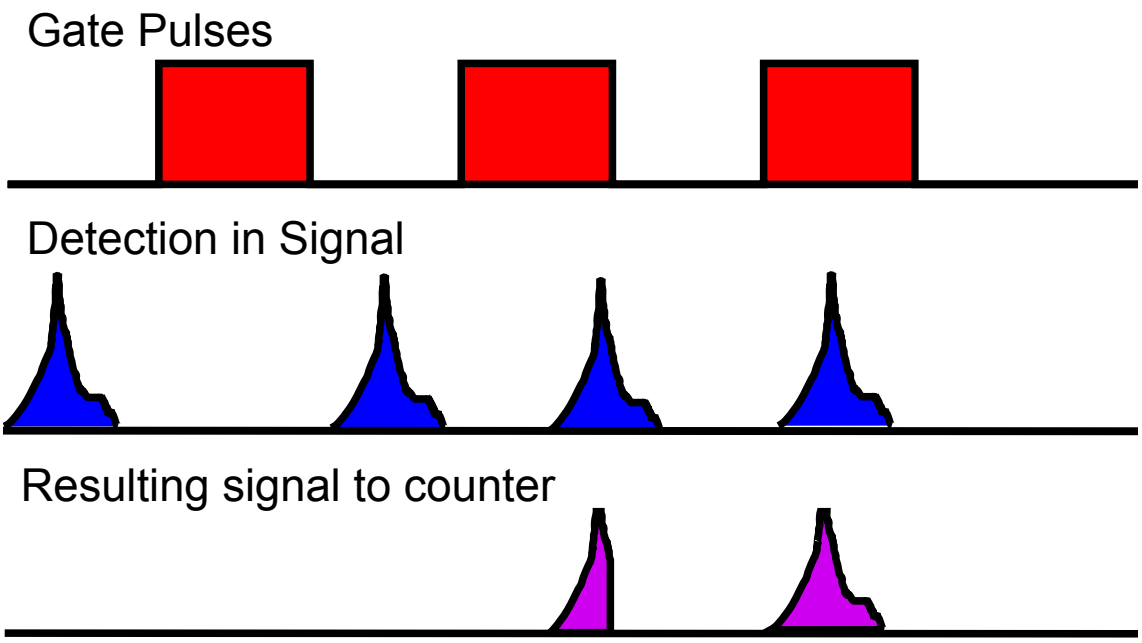


Figure 1.1: When the gate input is enabled, the gate pulses and detection signal are ANDed together, producing the resulting signal to counter.

transformed output will produce a pulse that goes high with the positive edge of the first pulse, and low when the coincidence window has elapsed after the second pulse. If a pulse arrives on the other channel within this time (after the first pulse but before the end of the coincidence window after the second pulse) it will be counted as a single coincidence.

If two pulses are separated by greater than one coincidence window, but less than two coincidence windows, and a pulse arrives on the other channel between these two pulses, then it will be counted as two coincidences instead of one.

Due to programming constraints, the transformed coincidence input will be delayed by up to 9 nanoseconds from the original input pulse.

1.4 Dwell Time

The dwell time is the amount of time which elapses before the clear signal is sent to the counters in count enable mode. When count enable mode is on (indicated by a play button), the counters will count up until this dwell time has elapsed. When count enable mode is off (indicated by the absence button), the counters will be cleared every 0.1 s.

The menu interface allows the dwell time to be set to integer second values of dwell time between 0 an 60 seconds. For dwell times less greater than 60 seconds, or with

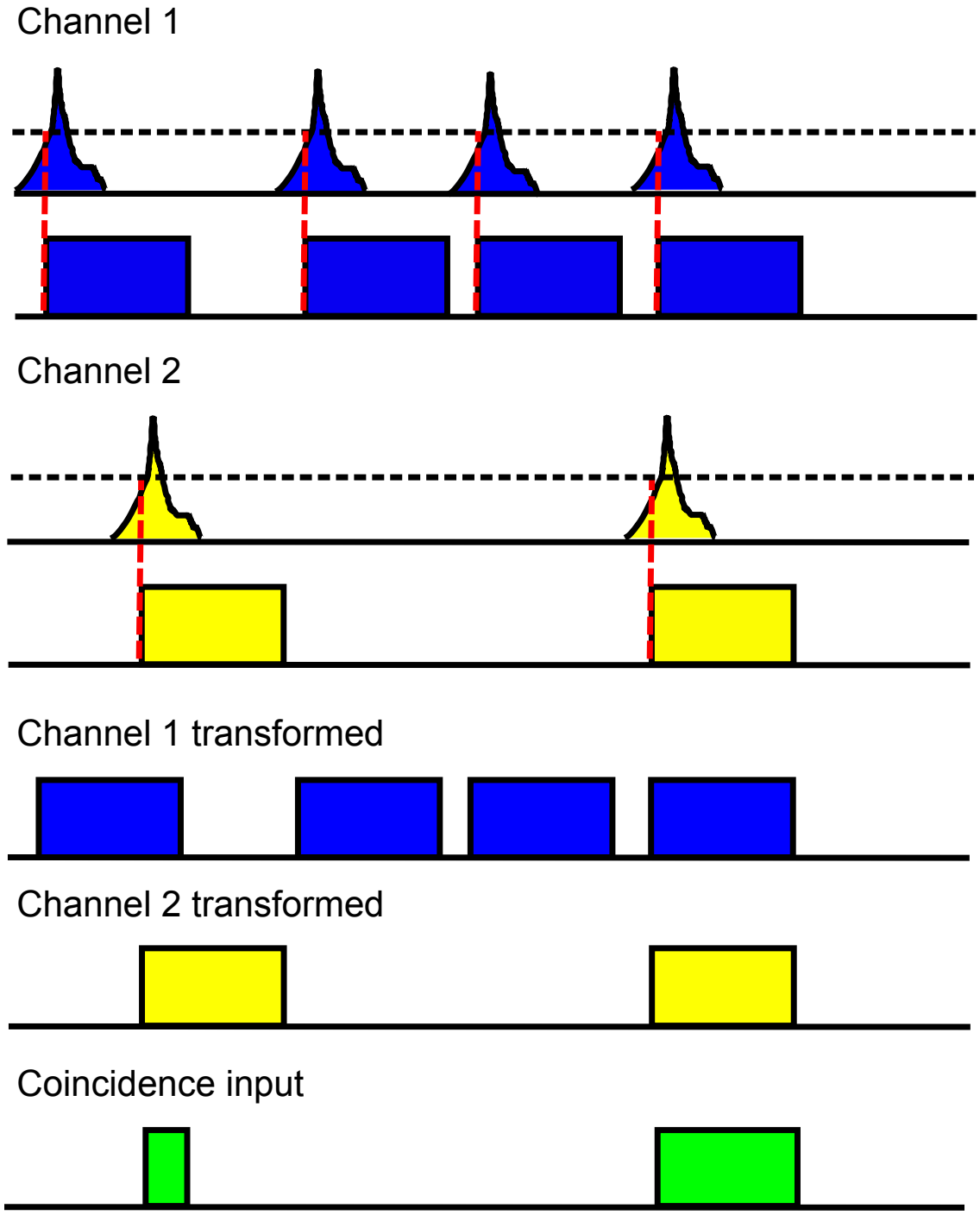


Figure 1.2: To find which pulses fall within the same time window, the inputs on channel 1 and 2 are transformed by creating an output signal which goes high on the positive edge of the input signal, and low after the specified window passes. These two transformed signals are ANDed together, and sent to the counter.

decimal values of seconds, use the computer interface.

The dwell time has a lower resolution of a millisecond. If the command to set the dwell time to 0.0019, it will have the same effect as setting the dwell time to 0.001.

The dwell time cannot be negative. If a negative dwell time is sent to the CC1, it will not be set.

1.5 Coincidence Window

The coincidence window is the maximum amount of time between the positive edge of signals that can elapse for two signals to be considered a coincidence. The coincidence window can be set to 0, 1, 2, 3, 4, 5, 6, or 7 nanoseconds.

If the coincidence window is set to 0, no coincidences will be found, so if measuring the coincidence rate between signals that are less than 1 ns apart, set the coincidence window to 1.

If the coincidence window is set to decimal numbers through the computer interface, the CC1 will always round down to the nearest integer. For example, if the computer interface receives the command to set the coincidence window to 7.6 ns, it will be set to 7 ns.

If the command to set the coincidence window to 8 ns or higher is sent, the coincidence window will not be set.

1.6 Gate Enable

The gate enable setting enables or disables the gating of the channel 1 and channel 2 signals with the gate in signal. By default, gate enable is off. Gate enable can be turned on through either the menu or through the computer interface.

1.7 Count Enable

The count enable setting distinguishes between free running mode, where the counters will only report their values once every 100 milliseconds before being cleared, and count enable mode, where the counters will count up during the specified dwell time before being cleared. Count enable mode can be triggered either by pressing the “ENABLE” button on the front of the unit, or through the computer interface.

1.8 Clear Counts

Clearing the counts resets them to zero. This includes clearing the overflow indicator.

Chapter 2

Unit Interface

2.1 Buttons

2.1.1 Power

If the CC1 is off, pressing the power button briefly will turn it on. If the CC1 is on, pressing and holding the power button for at least 3 seconds will turn it off.

2.1.2 Up and Down

If the CC1's menu is on, the "UP" and "DOWN" buttons will allow you to cycle through the menu options and possible values for the settings. These buttons have no effect when the menu is not on.

2.1.3 Menu

Pressing "MENU" toggles the menu. If the menu is open when the "MENU" button is pressed, the current values of the settings will be applied.

2.1.4 Clear

Pressing "CLEAR" will clear all counters, including the overflow indicator. Pressing this button has no effect when the menu is on.

2.1.5 Enable

Pressing the "ENABLE" button when the menu is off will toggle count enable mode. Pressing the "ENABLE" button when the menu if off, and count enable mode is on will

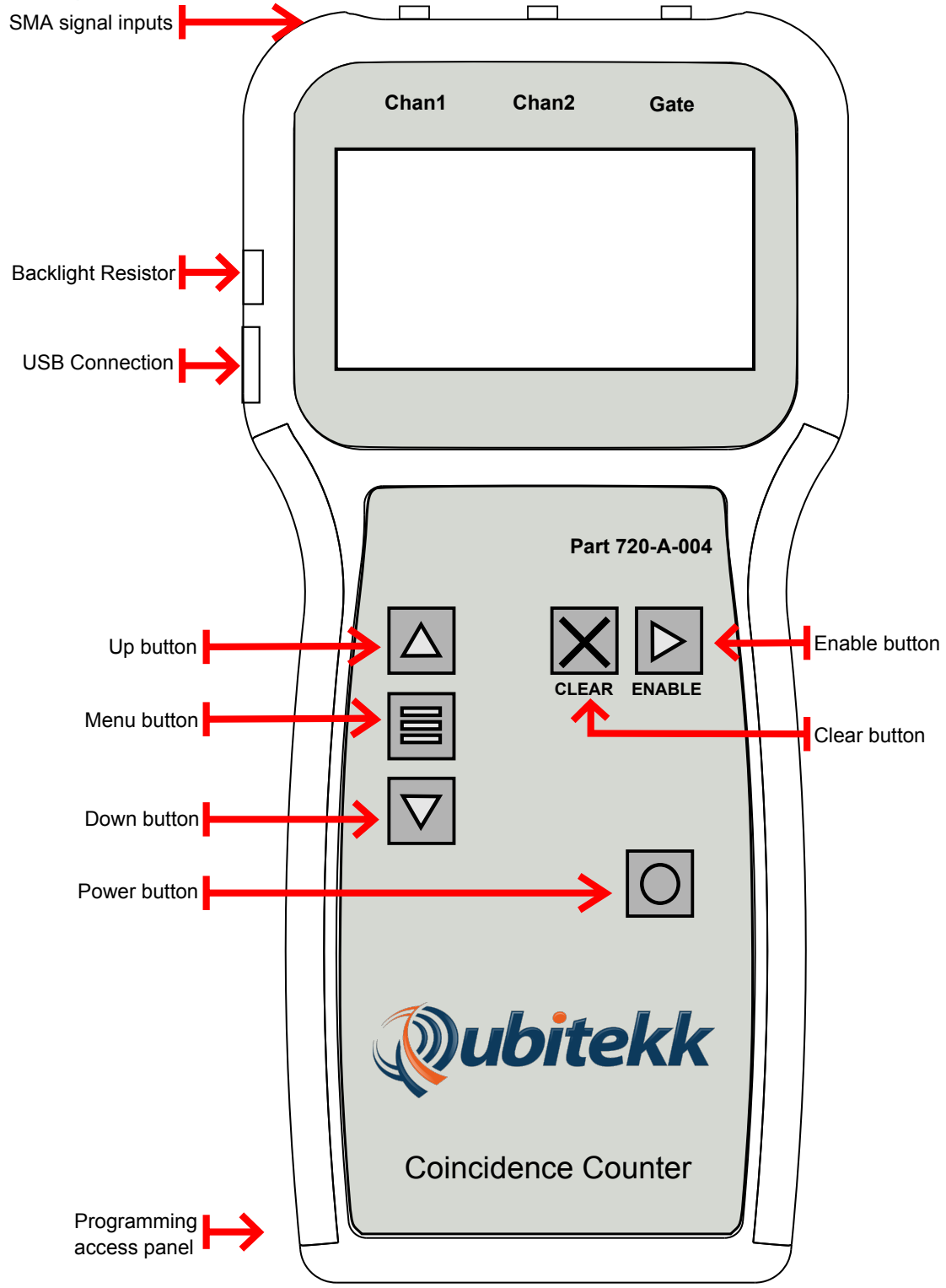


Figure 2.1: CC1 front panel

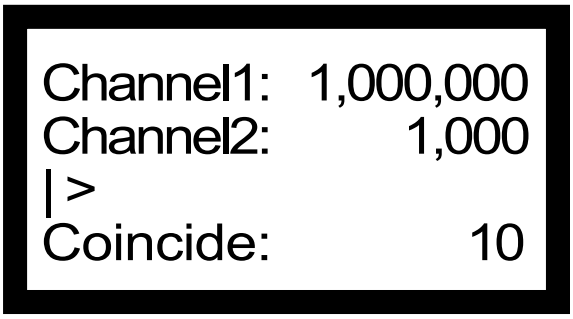


Figure 2.2: The LCD screen with the menu off. The | > characters indicate that the CC1 is in count enable mode.

turn the count enable mode off.

When the menu is on, pressing “ENABLE” will select the menu option next to the flashing arrow. Pressing “ENABLE” with the menu on and an option selected will deselect the option.

2.2 LCD Screen

The LCD screen on the CC1 will display one of two things; either the current values of the counters (2.2), or the menu (2.2). When the LCD screen displays the counter totals, it also indicates whether the CC1 is in count enable mode with the characters | > on the third line.

If “CLEAR” or “ENABLE” is pressed while on the counts screen, the LCD will briefly show the messages “Counts Cleared” or “Count Enable”.

The menu can be toggled with the “MENU” button. When the menu is open, a flashing arrow will appear to indicate the highlighted option. Pressing the “UP” and “DOWN” buttons will move the highlighted option. Pressing the “ENABLE” button will cause this option to be selected. When selected, the arrow will stop flashing, and instead the value of the option will flash. Pressing the “UP” and “DOWN” buttons will cycle through the available values for this option. Pressing “ENABLE” will return to the flashing arrow and the ability to select other options. Pressing “MENU” will exit the menu screen, and send the last values to the settings.

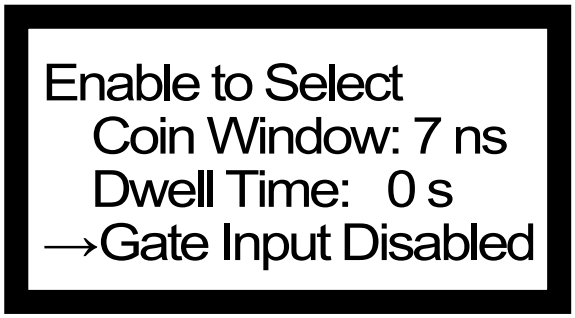


Figure 2.3: The LCD screen with the menu on.

Chapter 3

Serial Interface

The CC1 has an emulated COM port for serial communication, and uses SCPI-compliant commands. These commands have been incorporated into the python open-source InstrumentKit project, available at: `https://github.com/Galvant/InstrumentKit`. Sample code provided from Qubitekk uses Python and this interface. Code for controlling the device using other languages can be developed by writing SCPI commands to the serial interface.

To communicate with the CC1, download a serial terminal emulator program (such as RealTerm or PuTTY on Windows, or minicom on Linux). The interface settings should be:

- Baud Rate: 19200 b/s
- Data bits: 8
- Parity: none
- Stop Bits: 1
- Flow Control: None

Mnemic	Name	Description
* IDN?	Identification Query	Returns the vendor, part number, and firmware version of the device
* RST	Reset Command	Resets to factory settings

Mnemic	Name	Description
:GATE:ON (:OFF)	Gate Enable Command	Turn the gate input on or off
GATE?	Gate Enable Query	Returns 1 for on and 0 for off
:COUNt:ON (:OFF)	Count Enable Command	Turn the count enable mode on or off
COUNt?	Count Enable Query	Returns 1 for on and 0 for off
:WINDow N	Coincidence Window Length Set	Set the coincidence window length to N nanoseconds. N must be between 0 and 7.
WINDow?	Coincidence Window Length Query	Returns the coincidence window length in nanoseconds.
:DWELltime N	Dwell Time Set	Set the dwell time to N seconds. N must be positive.
DWELltime	Dwell Time Query	Returns the dwell time in seconds.
COUNter:C1? (:C2?)(:CO?)	Counter Query	Returns the current value on the counter, either Channel 1 (C1), Channel 2 (C2), or Coincidences (CO)
:CLEAr	Clear Command	Sends the clear signal to the counters

3.1 SCPI Protocol

3.1.1 Mandated SCPI Commands

3.1.2 CC1 Commands

If the command is not recognized, the CC1 will reply with "Unknown Command"

3.2 InstrumentKit Driver

To open a connection to the CC1 using the InstrumentKit driver, first import the library:

```
import instruments as ik
```

Next, open a connection to a serial port.

On Windows, this would be, for example:

```
cc = ik.qubitekk.CC1.open_serial('COM8', 19200,timeout=1)
```

on Linux, it would be:

```
cc = ik.qubitekk.CC1.open_serial('/dev/ttyUSB0', 19200,timeout=1)
```

You can now access all of the features of the CC1 by calling different attributes of the cc object.

For example, to set the coincidence window to 3 ns, use:

```
cc.window = 3
```

or to read the coincidences, use:

```
print(cc.channel[2].count)
```

3.3 Sample Code

Sample code is provided in python interacting with the CC1 using the InstrumentKit and using serial communication. This code uses the Tkinter library for the user interface.

3.3.1 InstrumentKit example

```
#Qubitekk Coincidence Counter example
import matplotlib
matplotlib.use('TkAgg')

from matplotlib.figure import Figure

from numpy import arange, sin, pi
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg,
    NavigationToolbar2TkAgg
# implement the default mpl key bindings
from matplotlib.backend_bases import key_press_handler

import instruments as ik

import Tkinter as tk
import re
#open connection to coincidence counter. If you are using
    Windows, this will be a com port. On linux, it will show up
    in /dev/ttyusb
cc = ik.qubitekk.CC1.open_serial('COM8', 19200,timeout=1)
#i is used to keep track of time
i = 0

def clearcounts(*args):
```

```
    cc.clear_counts()

def getvalues(i):
    #set counts labels
    chan1counts.set(cc.channel[0].count)
    chan2counts.set(cc.channel[1].count)
    coinccounts.set(cc.channel[2].count)
    #add count values to arrays for plotting
    chan1vals.append(cc.channel[0].count)
    chan2vals.append(cc.channel[1].count)
    coincvals.append(cc.channel[2].count)
    t.append(i*0.1)
    i = i+1
    #plot values
    p1, = a.plot(t,coincvals,color="r",linewidth=2.0)
    p2, = a.plot(t,chan1vals,color="b",linewidth=2.0)
    p3, = a.plot(t,chan2vals,color="g",linewidth=2.0)
    a.legend([p1,p2,p3],["Coincidences","Channel 1", "Channel 2"
        ])
    a.set_xlabel('Time (s)')
    a.set_ylabel('Counts (Hz)')

    canvas.show()
    #get the values again in 100 milliseconds
    root.after(100,getvalues,i)

def gateenable():
    if(gateenabled.get()):
        cc.gate_enable = 1
    else:
        cc.gate_enable = 0

def parse(*args):
    cc.dwell_time = float(re.sub("[A-z]", "", dwell_time.get()))
    cc.window = float(re.sub("[A-z]", "", window.get()))

root = tk.Tk()
root.title("Qubitekk Coincidence Counter Control Software")

#set up the Tkinter grid layout
mainframe = tk.Frame(root)
mainframe.padding = "3 3 12 12"
mainframe.grid(column=0, row=0, sticky=(tk.N, tk.W, tk.E, tk.S))
mainframe.columnconfigure(0, weight=1)
```

```
mainframe.rowconfigure(0, weight=1)

#set up the label text
dwelltime = tk.StringVar()
dwelltime.set(cc.dwell_time)

window = tk.StringVar()
window.set(cc.window)

chan1counts = tk.StringVar()
chan1counts.set(cc.channel[0].count)
chan2counts = tk.StringVar()
chan2counts.set(cc.channel[1].count)
coinccounts = tk.StringVar()
coinccounts.set(cc.channel[2].count)

gateenabled = tk.IntVar()

#set up the initial checkbox value for the gate enable
if(cc.gateenable=="enabled"):
    gateenabled.set(1)
else:
    gateenabled.set(0)

#set up the plotting area

f = Figure(figsize=(10,8), dpi=100)
a = f.add_subplot(111,axisbg='black')

t = []
coincvals = []
chan1vals = []
chan2vals = []


# a tk.DrawingArea
canvas = FigureCanvasTkAgg(f, mainframe)
canvas.get_tk_widget().grid(column=3,row=1,rowspan=8,sticky=tk.W
    )

#label initialization
dwelltime_entry = tk.Entry(mainframe, width=7, textvariable=
    dwelltime,font="Verdana 20")
dwelltime_entry.grid(column=2,row=2,sticky=(tk.W,tk.E))
```

```
window_entry = tk.Entry(mainframe, width=7, textvariable=window,
    font="Verdana 20")
window_entry.grid(column=2,row=3,sticky=(tk.W,tk.E))
qubitekklogo = tk.PhotoImage(file="qubitekklogo.gif")

tk.Label(mainframe,image=qubitekklogo).grid(column=1,row=1,
    columnspan=2)

tk.Label(mainframe, text="Dwell Time:",font="Verdana 20").grid(
    column=1,row=2,sticky=tk.W)
tk.Label(mainframe, text="Window size:",font="Verdana 20").grid(
    column=1,row=3,sticky=tk.W)

tk.Checkbutton(mainframe,font="Verdana 20",variable =
    gateenabled, command=gateenable).grid(column=2,row=4)
tk.Label(mainframe,text="Gate Enable: ",font="Verdana 20").grid(
    column=1,row=4,sticky=tk.W)


tk.Label(mainframe, text="Channel 1: ",font="Verdana 20").grid(
    column=1,row=5,sticky=tk.W)
tk.Label(mainframe, text="Channel 2: ",font="Verdana 20").grid(
    column=1,row=6,sticky=tk.W)
tk.Label(mainframe, text="Coincidences: ",font="Verdana 20").
    grid(column=1,row=7,sticky=tk.W)



tk.Label(mainframe, textvariable=chan1counts,font="Verdana 34",
    fg="white",bg="black").grid(column=2,row=5,sticky=tk.W)
tk.Label(mainframe, textvariable=chan2counts,font="Verdana 34",
    fg="white",bg="black").grid(column=2,row=6,sticky=tk.W)
tk.Label(mainframe, textvariable=coinccounts,font="Verdana 34",
    fg="white",bg="black").grid(column=2,row=7,sticky=tk.W)

tk.Button(mainframe,text="Clear Counts",font="Verdana 24",
    command=clearcounts).grid(column=2,row=8,sticky = tk.W)

for child in mainframe.winfo_children(): child.grid_configure(
    padx=5, pady=5)
#when the enter key is pressed, send the current values in the
    entries to the dwelltime and window to the coincidence
    counter
root.bind('<Return>',parse)
```

```
#in 100 milliseconds, get the counts values off of the
    coincidence counter
root.after(100,getvalues,i)
#start the GUI
root.mainloop()
```

3.3.2 Serial example

```
#Qubitekk Coincidence Counter example
import matplotlib
matplotlib.use('TkAgg')

from matplotlib.figure import Figure

from numpy import arange, sin, pi
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg,
    NavigationToolbar2TkAgg
# implement the default mpl key bindings
from matplotlib.backend_bases import key_press_handler

import serial

import Tkinter as tk
import re
#open connection to coincidence counter. If you are using
    Windows, this will be a com port. On linux, it will show up
    in /dev/ttyusb
cc = serial.Serial('COM8', 19200,timeout=1)
#i is used to keep track of time
i = 0

def query(ser,command):
    ser.write(command)
    return ser.readline()

def clearcounts(*args):
    cc.write("CLEA\n")

def getvalues(i):
    #set counts labels
    chan1counts.set(int(query(cc,"COUN:C1?\n")))
    chan2counts.set(int(query(cc,"COUN:C2?\n")))
    coinccounts.set(int(query(cc,"COUN:CO?\n")))
    #add count values to arrays for plotting
```

```
        chan1vals.append(query(cc,"COUN:C1?\n"))
        chan2vals.append(query(cc,"COUN:C2?\n"))
        coincvals.append(query(cc,"COUN:CO?\n"))
        t.append(i*0.1)
        i = i+1
        #plot values
        p1, = a.plot(t,coincvals,color="r",linewidth=2.0)
        p2, = a.plot(t,chan1vals,color="b",linewidth=2.0)
        p3, = a.plot(t,chan2vals,color="g",linewidth=2.0)
        a.legend([p1,p2,p3],["Coincidences","Channel 1", "Channel 2"
            ])
        a.set_xlabel('Time (s)')
        a.set_ylabel('Counts (Hz)')

        canvas.show()
        #get the values again in 100 milliseconds
        root.after(100,getvalues,i)

def gateenable():
        if(gateenabled.get()):
                ser.write(":GATE:ON\n")
        else:
                ser.write(":GATE:OFF\n")

def parse(*args):
        dwelltime = float(re.sub("[A-z]", "", dwelltime.get()))
        ser.write(":DWEL "+str(dwelltime)+"\n")
        window = float(re.sub("[A-z]", "", window.get()))
        ser.write(":WIND "+str(window))

root = tk.Tk()
root.title("Qubitekk Coincidence Counter Control Software")

#set up the Tkinter grid layout
mainframe = tk.Frame(root)
mainframe.padding = "3 3 12 12"
mainframe.grid(column=0, row=0, sticky=(tk.N, tk.W, tk.E, tk.S))
mainframe.columnconfigure(0, weight=1)
mainframe.rowconfigure(0, weight=1)

#set up the label text
dwelltime = tk.StringVar()
dwelltime.set(query(cc,"DWEL?\n"))
window = tk.StringVar()
```

```
window.set(query(cc,"WIND?\n"))

chan1counts = tk.StringVar()
chan1counts.set(int(query(cc,"COUN:C1?\n")))
chan2counts = tk.StringVar()
chan2counts.set(int(query(cc,"COUN:C2?\n")))
coinccounts = tk.StringVar()
coinccounts.set(int(query(cc,"COUN:CO?\n")))

gateenabled = tk.IntVar()

#set up the initial checkbox value for the gate enable
if(query(cc,"GATE?\n")):
    gateenabled.set(1)
else:
    gateenabled.set(0)

#set up the plotting area

f = Figure(figsize=(10,8), dpi=100)
a = f.add_subplot(111,axisbg='black')

t = []
coincvals = []
chan1vals = []
chan2vals = []


# a tk.DrawingArea
canvas = FigureCanvasTkAgg(f, mainframe)
canvas.get_tk_widget().grid(column=3,row=1,rowspan=8,sticky=tk.W
    )

#label initialization
dwelltime_entry = tk.Entry(mainframe, width=7, textvariable=
    dwelltime,font="Verdana 20")
dwelltime_entry.grid(column=2,row=2,sticky=(tk.W,tk.E))
window_entry = tk.Entry(mainframe, width=7, textvariable=window,
    font="Verdana 20")
window_entry.grid(column=2,row=3,sticky=(tk.W,tk.E))
qubitekklogo = tk.PhotoImage(file="qubitekklogo.gif")

tk.Label(mainframe,image=qubitekklogo).grid(column=1,row=1,
    columnspan=2)
```

```
tk.Label(mainframe, text="Dwell Time:",font="Verdana 20").grid(
    column=1,row=2,sticky=tk.W)
tk.Label(mainframe, text="Window size:",font="Verdana 20").grid(
    column=1,row=3,sticky=tk.W)

tk.Checkbutton(mainframe,font="Verdana 20",variable =
    gateenabled, command=gateenable).grid(column=2,row=4)
tk.Label(mainframe,text="Gate Enable: ",font="Verdana 20").grid(
    column=1,row=4,sticky=tk.W)


tk.Label(mainframe, text="Channel 1: ",font="Verdana 20").grid(
    column=1,row=5,sticky=tk.W)
tk.Label(mainframe, text="Channel 2: ",font="Verdana 20").grid(
    column=1,row=6,sticky=tk.W)
tk.Label(mainframe, text="Coincidences: ",font="Verdana 20").
    grid(column=1,row=7,sticky=tk.W)



tk.Label(mainframe, textvariable=chan1counts,font="Verdana 34",
    fg="white",bg="black").grid(column=2,row=5,sticky=tk.W)
tk.Label(mainframe, textvariable=chan2counts,font="Verdana 34",
    fg="white",bg="black").grid(column=2,row=6,sticky=tk.W)
tk.Label(mainframe, textvariable=coinccounts,font="Verdana 34",
    fg="white",bg="black").grid(column=2,row=7,sticky=tk.W)

tk.Button(mainframe,text="Clear Counts",font="Verdana 24",
    command=clearcounts).grid(column=2,row=8,sticky = tk.W)

for child in mainframe.winfo_children(): child.grid_configure(
    padx=5, pady=5)
#when the enter key is pressed, send the current values in the
    entries to the dwelltime and window to the coincidence
    counter
root.bind('<Return>',parse)
#in 100 milliseconds, get the counts values off of the
    coincidence counter
root.after(100,getvalues,i)
#start the GUI
root.mainloop()
```

Appendix A

Uploading Custom Firmware

A.1 Digi Rabbit 3400

The CC1 uses a Digi Rabbit 3400 microprocessor. The user manual for this unit can be found at `http://ftp1.digi.com/support/documentation/019-0122_P.pdf`. The Digi Rabbit 3400 can be programmed using the Dynamic C language. A reference manual on the Dynamic C language can be found online at `http://ftp1.digi.com/support/documentation/019-0113_N.pdf`.

The integrated development environment and compiler for Dynamic C is available for free download from Digi with registration. Dynamic C Version 9 is required, which can be downloaded at `http://www.digi.com/products/wireless-wired-embedded-solutions/software-microprocessors-accessories/software/dynamicc#docs`.

Dynamic C is very similar to ANSI C, with extra functions for interaction with the digital i/o, analog inputs, and PWM outputs. It can also incorporate assembly to speed up processing.

On the CC1, the Rabbit can be accessed through J3 8 pin header through the bottom of the board. The programming cable used is the 20-101-1201, which can be purchased from Mouser: `http://www.mouser.com/ProductDetail/Rabbit-Semiconductor/20-101-1201/`

The firmware source code for the Rabbit can be found in the Qubitekk Coincidence Counter repository at ...(url in the future)

A.2 Terasic DE0-Nano

The FPGA in the CC1 is an Altera Cyclone IV E on a Terasic DE0-Nano board. This FPGA can be programmed using the the Quartus II IDE/ compiler, version 13+. The Quartus II Web Edition is available from Altera for free download with registration at `https://www.altera.com/download/sw/dnl-sw-index.jsp`. The user manual for this

board can be found online at: `http://www.terasic.com.tw/cgi-bin/page/archive_download.pl?Language=English&No=593&FID=75023fa36c9bf8639384f942e65a46f3`

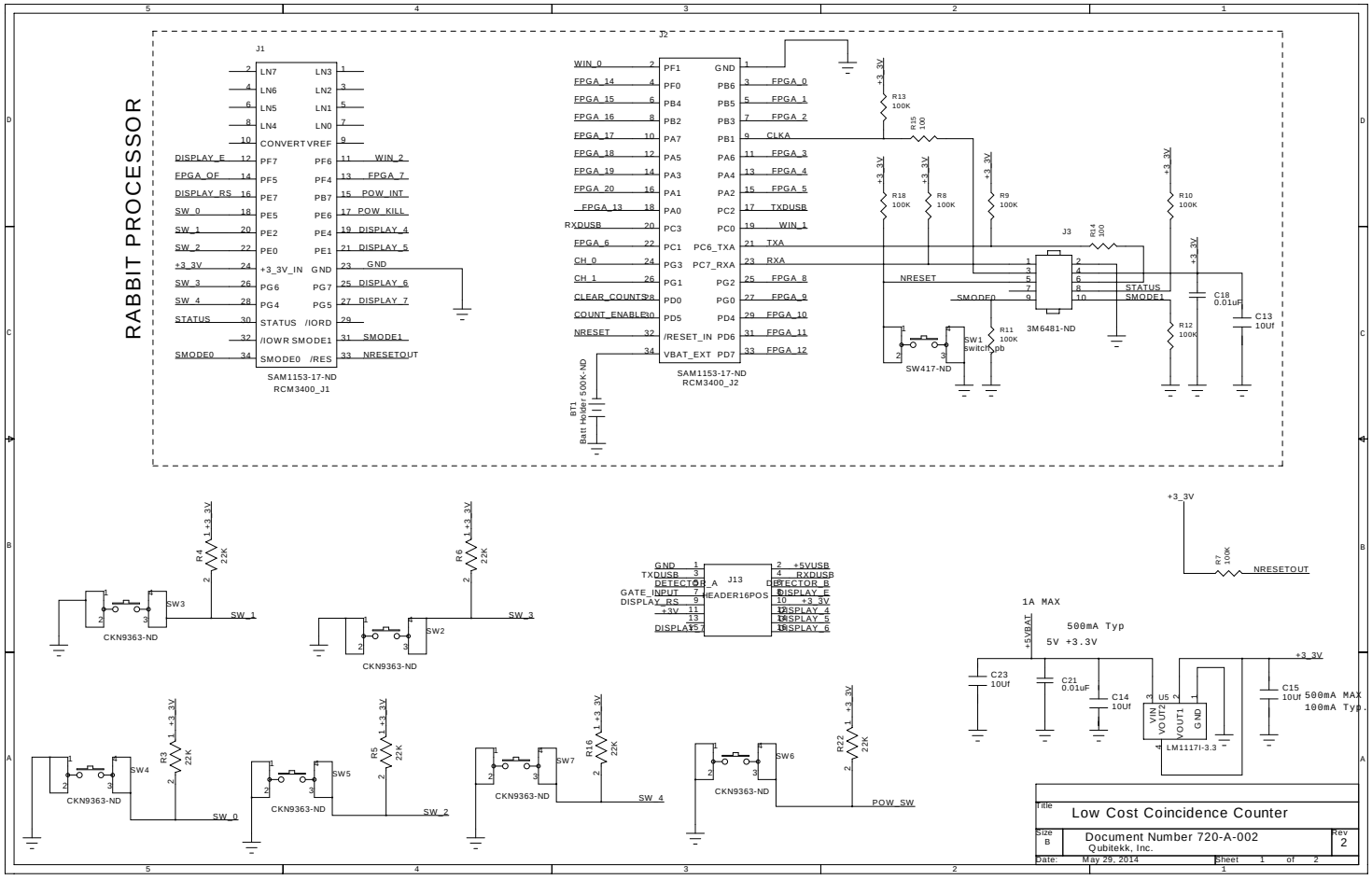
The Quartus project file, including the verilog modules, pin assignments, board support files and board schematic files are all available from in the Qubitekk Coincidence Counter repository at ...(url in the future).

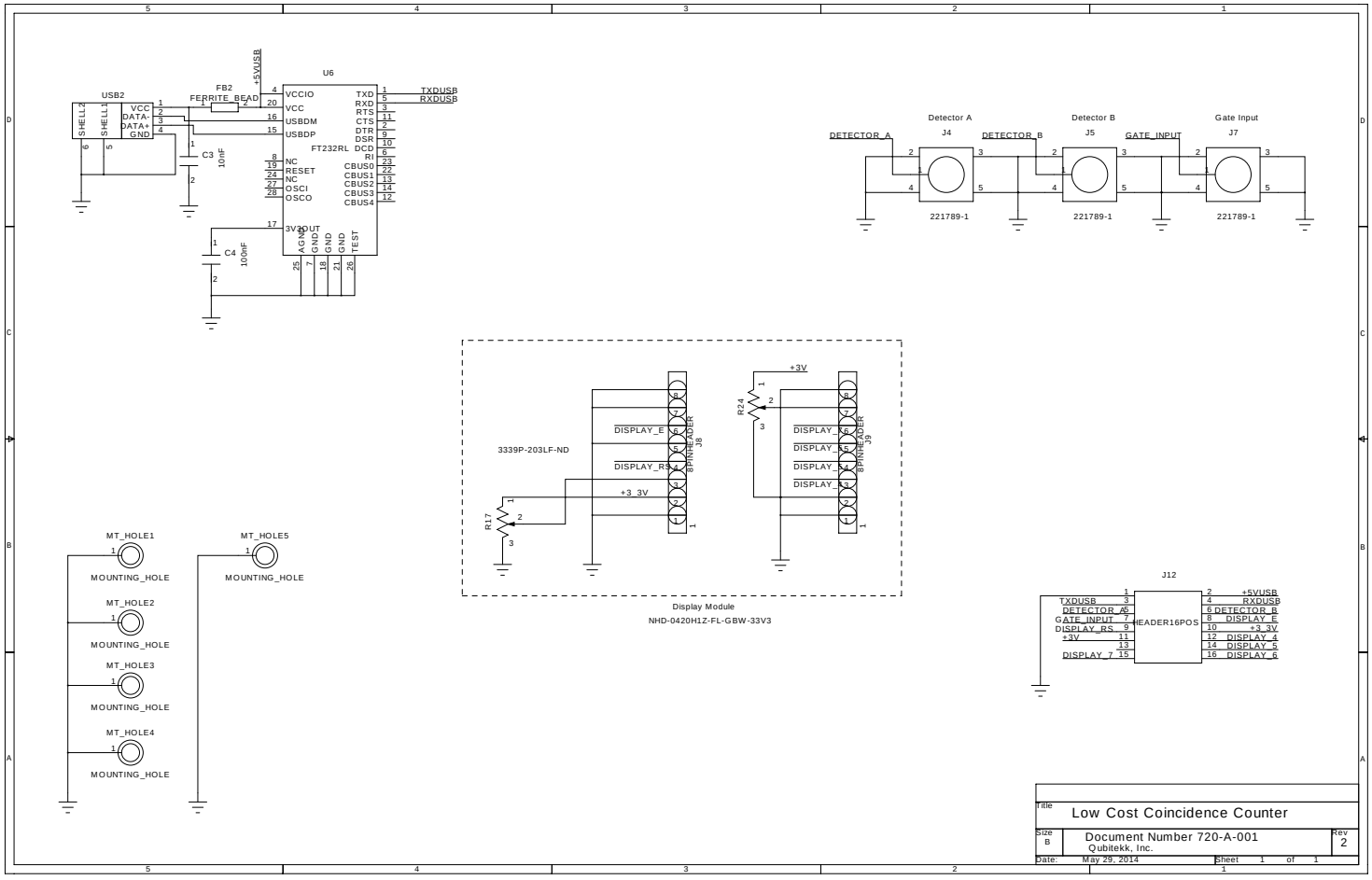
The Cyclone can be programmed through the USB-blaster port, accessible through the bottom panel of the CC1. To program the DE0-Nano and have it persist in memory after power cycling, a JTAG Indirect Configuration File (.jic) must be created. This can be done by converting a compiled SRAM Object File (.sof) under File-Convert Programming Files. In this menu, the configuration device should be set to EPCS64. This configuration device may change in future revisions of the DE0-Nano, check with your user manual. Add the SOF by selecting "SOF Data" and clicking on the "Add File" button to the right. The EP4C22 device should be added to the Flash Loader.

To program this device, add the .jic file that was generated to the programmer window.

Appendix B

Electrical Schematics


RABBIT PROCESSOR
J1
SAM1153-17-ND
RCM3400_J1
J2
SAM1153-17-ND
RCM3400_J2
J3
3M6481-ND
SW417-ND
BT1
Batt Holder 500K-ND
CKN9363-ND
J13
HEADER16POS
LM1117I-3.3
1A MAX
500mA Typ
5V +3.3V
500mA MAX
100mA Typ.
Low Cost Coincidence Counter
Document Number 720-A-002
Qubitekk, Inc.
Rev 2
May 29, 2014
Sheet 1 of 2


U6
USB2
FB2
FERRITE_BEAD
+5VUSB
VCCIO
VCC
USBDM
USBDP
FT232RL
TXD
RXD
RTS
CTS
DTR
DSR
DCD
RI
CBUS0
CBUS1
CBUS2
CBUS3
CBUS4
TXDUSB
RXDUSB
C3
10nF
C4
100nF
3V3OUT
AGND
GND
TEST
Detector A
Detector B
Gate Input
DETECTOR_A
DETECTOR_B
GATE_INPUT
J4
J5
J7
221789-1
3339P-203LF-ND
R17
R24
+3V
+3_3V
DISPLAY_E
DISPLAY_RS
DISPLAY_4
DISPLAY_5
DISPLAY_6
DISPLAY_7
8PIN HEADER
J8
J9
Display Module
NHD-0420H1Z-FL-GBW-33V3
J12
HEADER16POS
MT_HOLE1
MT_HOLE2
MT_HOLE3
MT_HOLE4
MT_HOLE5
MOUNTING_HOLE
Title
Low Cost Coincidence Counter
Size
B
Document Number 720-A-001
Qubitekk, Inc.
Rev
2
Date:
May 29, 2014
Sheet
1
of
1