characterization

August 26, 2019

```
In [1]: using StatsBase
        using Statistics
        using CSV
        using DSP
        using Measurements
        using CRC
        using Optim
        using Tables
```

1 Delayed Signal Computations

1.1 DAC Rise Time

```
In [2]: rtdata = CSV.File("./data/Tek004.csv"; header=false)
        rtweights = FrequencyWeights(rtdata.Column2)
        rt = mean(rtdata.Column1, rtweights)
        rtσ = std(rtdata.Column1, rtweights, corrected=true)

Out[2]: 1.1492767433389036e-9

In [3]: rtσ /= 4 # For a 1 Vpp square wave edge
        rt /= 4

Out[3]: 1.290866506382043e-7

In [4]: (rt ± rtσ) * 1e9
```

Out[4]:

129.09 ± 0.29

1.2 Delay Error

The steps in the error plots are from single-digits on the scope, so the results should taken with a grain of salt.

```
In [5]: dlimit=35;
        ddata = CSV.File("./delay_linearity.csv"; skipto=2, limit=dlimit,
            header=[:min, :max, :swords, :na, :clock, :signal, :rt]);
        rtcorrection = vcat(repeat([rt], 13), zeros(dlimit-13));
        minerr = 1e-3.*ddata.min .- (ddata.swords .* 20e-6) .+ 20e-6 - rtcorrection;
        maxerr = 1e-3.*ddata.max .- (ddata.swords .* 20e-6) - rtcorrection;
```

In [6]:
```
last(ddata.swords)
```

Out[6]:
```
9000
```

Clock is at 50 kHz.

1.3 Estimated Internal Propagation Delay

Roughly estimated from risetime-corrected delay data.

In [7]:
```
pt = mean(map(mean, [minerr[1:13], maxerr[1:13]])) ./ 1e-9 # ns
```

Out[7]:
```
40.14411859255405
```

1.4 In-Out Difference Mean

In [8]:
```
meandata = CSV.File("./data/diffmean2.csv"; header=false);
meanweights = FrequencyWeights(meandata.Column2);
diffμ = mean(meandata.Column1, meanweights);
diffμσ = std(meandata.Column1, meanweights, corrected=true);
(diffμ ± diffμσ) * 1e3 # mV
```

Out[8]:
-1.023 ± 0.095

1.5 In-Out Difference ACRMS (Standard Deviation)

In [9]:
```
acrmsdata = CSV.File("./data/diffacrms2.csv"; header=false);
acrmsweights = FrequencyWeights(acrmsdata.Column2);
diffσ = mean(acrmsdata.Column1, acrmsweights);
diffσσ = std(acrmsdata.Column1, acrmsweights, corrected=true);
(diffσ ± diffσσ) * 1e3 # mV (RMS)
```

Out[9]:
1.035 ± 0.058

2 Output Plots

- Input: 50 Hz sine wave.
- Clock: 50 kHz.
- Delay: 1000 swords.
- Measurement Bandwidth: 100 MHz.

Ignore the naming of the file. The signal is actually at *50 Hz*.

In [10]:
```
diff500 = CSV.File("./data/500HzDiff100MHz.csv", skipto=10, header=[:t, :v])
```

```
Out[10]: CSV.File("./data/500HzDiff100MHz.csv"):
         Size: 1250000 x 2
         Tables.Schema:
          :t  Float64
          :v  Float64
```

```
In [11]: # Resample/decimate for faster processing
         #ddiff500t = resample(diff500.t, 1 // 10);
         #ddiff500v = resample(diff500.v, 1 // 10);

         ddiff500t = diff500.t;
         ddiff500v = diff500.v;
```

- Input: 5 kHz sine wave.
- Clock: 5 MHz.
- Delay: 1000 swords.
- Measurement Bandwidth: 250 MHz.

Ignore the naming of the file. The signal is actually at *5 kHz*.

```
In [12]: diff5M = CSV.File("./data/5MHzDiff250MHz.csv", skipto=10, header=[:t, :v])
```

```
Out[12]: CSV.File("./data/5MHzDiff250MHz.csv"):
         Size: 625000 x 2
         Tables.Schema:
          :t  Float64
          :v  Float64
```

```
In [13]: # Resample/decimate for faster processing
         #ddiff5Mt = resample(diff5M.t, 1 // 10);
         #ddiff5Mv = resample(diff5M.v, 1 // 10);

         ddiff5Mt = diff5M.t;
         ddiff5Mv = diff5M.v;
```

3 Initial History Programming

```
In [14]: little_endian(x) = UInt8.([x & 0x00FF, x >> 8]);
```

```
In [15]: prog_words = [
             0x0000, 0x07FF, 0x0FFF, 0x07FF,
             0x0000, 0x07FF, 0x0FFF, 0x07FF,
             0x0000, 0x07FF, 0x0FFF, 0x07FF,
             0x0000, 0x07FF, 0x0FFF, 0x07FF
         ];
         waveform_file = "./prog_debug.bin";
         write(waveform_file, prog_words);
         crc32 = crc(CRC_32);
         prog_crc = crc32(vcat(little_endian.(prog_words)...));
         display(prog_crc);
```

3.1 Initial History Programming Instructions

- Open *Realterm*.
- On the *Display* tab, check the *newLine mode* box.
- On the *Port* tab, set the *Baud* rate to *115200*.
- Select the *Port* that the *Arduino Due* is plugged in to, such as `6 = \USBSER000`.
- Press enter in the *Port* field to open the port. You should see text from the Arduino Due.
- On the *Send* tab, select the first text field.
- Type `ph N`, where `N` is the number of words that you want to program.
- Click the *Send ASCII* button. (Do not include a line feed in this command, as it will be interpreted as data).
- In the *Dump File to Port* section, click the ellipses (. . .) box.
- Select the history data file that you want to program.
- Under *Delays*, set the first entry (*Character Delay*) to `1`.
- Click *Send File*. You should see a hexdump of the data from the Arduino Due as it is transmitted.
- At the end of the transmission, there will be a line that says `CRC: 0x....` Check that this matches the `CRC_32` of the sent data (little-endian).
- Send the `go` command to the Arduino Due to switch to the external input and clocks.

4 Filtered Output Plots

4.1 Signal A

- Input: 5 kHz sine wave.
- Clock: 5 MHz.
- Delay: 1000 swords.
- Measurement Bandwidth: 250 MHz.

```
In [16]: ain = CSV.File("./data/5-5-in.csv",
             skipto=10, header=[:t, :v]); # Input signal
         aout = CSV.File("./data/5-5-unfiltered.csv",
             skipto=10, header=[:t, :v]); # Unfiltered output
         arc = CSV.File("./data/5-5-106kHz-RC.csv",
             skipto=10, header=[:t, :v]); # 106 kHz RC filter (single-pole)
         abessel4 = CSV.File("./data/5-5-bessel4.csv",
             skipto=10, header=[:t, :v]); # ~121 kHz Bessel filter (four-pole)
```

```
In [17]: abessel4vpp = maximum(abessel4.v) - minimum(abessel4.v);
```

4.2 Signal B

- Input: 20 kHz symmetric triangle wave.
- Clock: 5 MHz.
- Delay: 1000 swords.

- Measurement Bandwidth: 250 MHz.

```
In [18]: bin = CSV.File("./data/5-20-in.csv",
             skipto=10, header=[:t, :v]); # Input signal. ZI: 6249.5
         bout = CSV.File("./data/5-20-unfiltered.csv",
             skipto=10, header=[:t, :v]); # Unfiltered output
         brc = CSV.File("./data/5-20-RC.csv",
             skipto=10, header=[:t, :v]); # 106 kHz RC filter (single-pole)
         bbessel4 = CSV.File("./data/5-20-bessel4.csv",
             skipto=10, header=[:t, :v]); # ~121 kHz Bessel filter (4-pole). ZI: -0.5
```

```
In [19]: bbessel4vpp = maximum(bbessel4.v) - minimum(bbessel4.v);
```

4.3 Nelder-Mead Optimization of Filter Output Gain and Phase

```
In [20]: sΔt = 3.2e-10;
         span = 0.5;
         margin = (1 - span) / 2;
         function slidesamples(x, n, m)
             xlen = length(x);
             a = mod(Int64(floor(n)), xlen);
             b = mod(Int64(floor(n) + floor(m)), xlen);
             x[a:b]
         end
         stdslide(x; m=margin) = slidesamples(x, m*length(x), span*length(x));
         function modifywave(y, n, k, off)
             off .+ (k .* slidesamples(y, n * length(y), span*length(y)))
         end
         function waveerr(x, y, s)
             xlen = length(x);
             rms(stdslide(x) .- modifywave(y, s...))
         end
```

```
Out[20]: waveerr (generic function with 1 method)
```

The default algorithm that `optimize` will use here is Nelder-Mead.

```
In [21]: arcoptim = optimize(s -> waveerr(ain.v, arc.v, s), [margin, 1., 0.]);
```

```
In [22]: abessel4optim = optimize(s -> waveerr(ain.v, abessel4.v, s),
              [margin, 1.0 / abessel4vpp, 0.]);
```

```
In [23]: arco = modifywave(arc.v, arcoptim.minimizer...);
```

```
In [24]: abessel4o = modifywave(abessel4.v, abessel4optim.minimizer...);
```

```
In [25]: CSV.write("paout.csv",
              (aint=stdslide(ain.t) .* 1e6,
                  ainv=stdslide(ain.v) .* 1e3,
```

```
                aoutt=stdslide(aout.t) .* 1e6,
                aoutv=stdslide(aout.v) .* 1e3,
                abessel4t=stdslide(abessel4.t) .* 1e6,
                abessel4v=stdslide(abessel4.v) .* 1e3 * abessel4optim.minimizer[2]))
```

Out[25]: "paout.csv"

In [26]:
```
arcodiff = stdslide(ain.v) .- arco;
abessel4odiff = stdslide(ain.v) .- abessel4o;
```

In [27]:
```
abessel4odiffμ = mean(abessel4odiff)
```

Out[27]: -7.533094976133311e-6

In [28]:
```
abessel4odiffσ = std(abessel4odiff)
```

Out[28]: 0.0029591421680602945