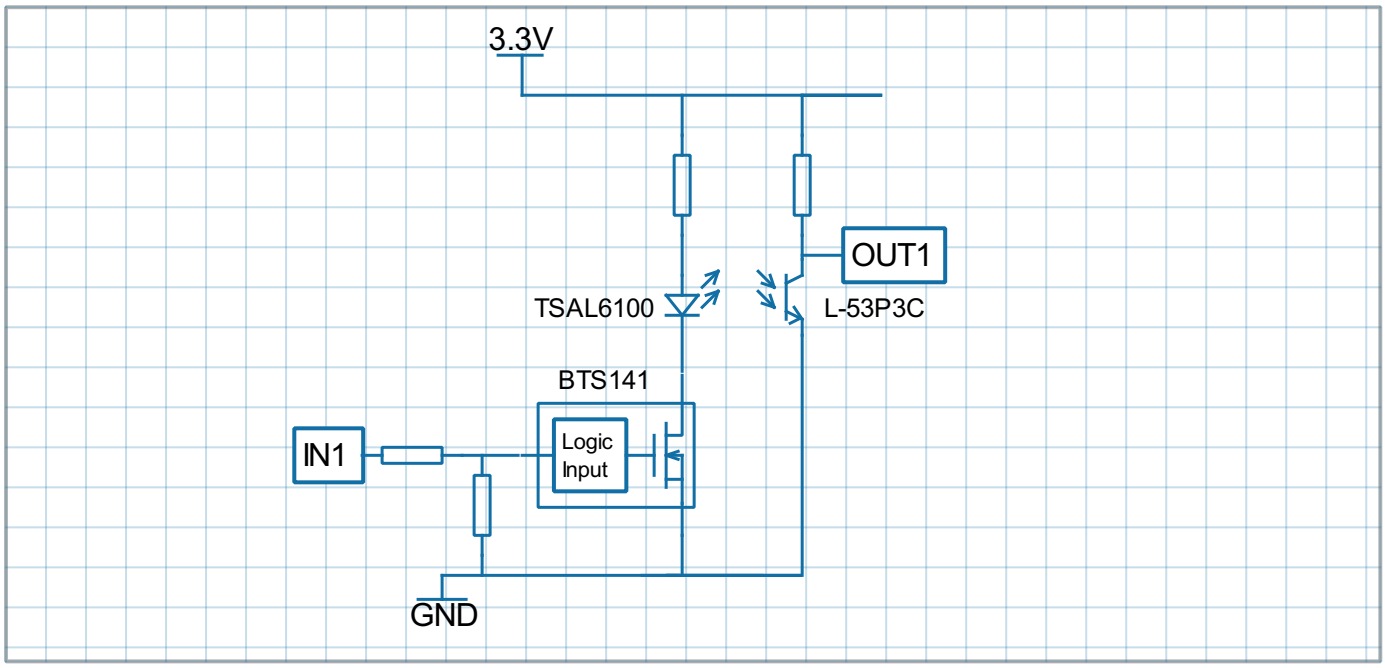
Optical communications system realized using LabJack T7 device and MatDeck

In this document, we will illustrate the example of an electronic circuit which functions as a basic optical communications system. The optical communication system is based on a light emitting diode (LED) which is used as a transmitter, and photo diode which is used as an optical receiver.

Schematics of the electronic circuits

The schematics of the described optical system is displayed below. It should be pointed out that the schematics is created in MatDeck, which is suitable for various professional drawings.



Functionality

- IN1 and OUT1 demonstrate IR communication with DSP filtering

Demo board schematic pin descriptions

- IN1 - input for IR LED driver
- OUT1 - output from IR sensor

Connection to ADC unit

- IN1 is connected to the PWM output.
- OUT1 is connected to Analog inputs 1

Parts:

- For power driver is used BTS 141 which is logic level low side driver.
- Photo-transistor L-53P3C NPN 5MM, 940nm
- IR emitterTSAL6100 20◦ 940nm

Use of LabJack T7

The LabJack T7 device is used in the experiment for several roles. First, the pulse width modulation (PWM) is used to produce a input signal, denoted as IN1 on the schematics above. IN1 signal is

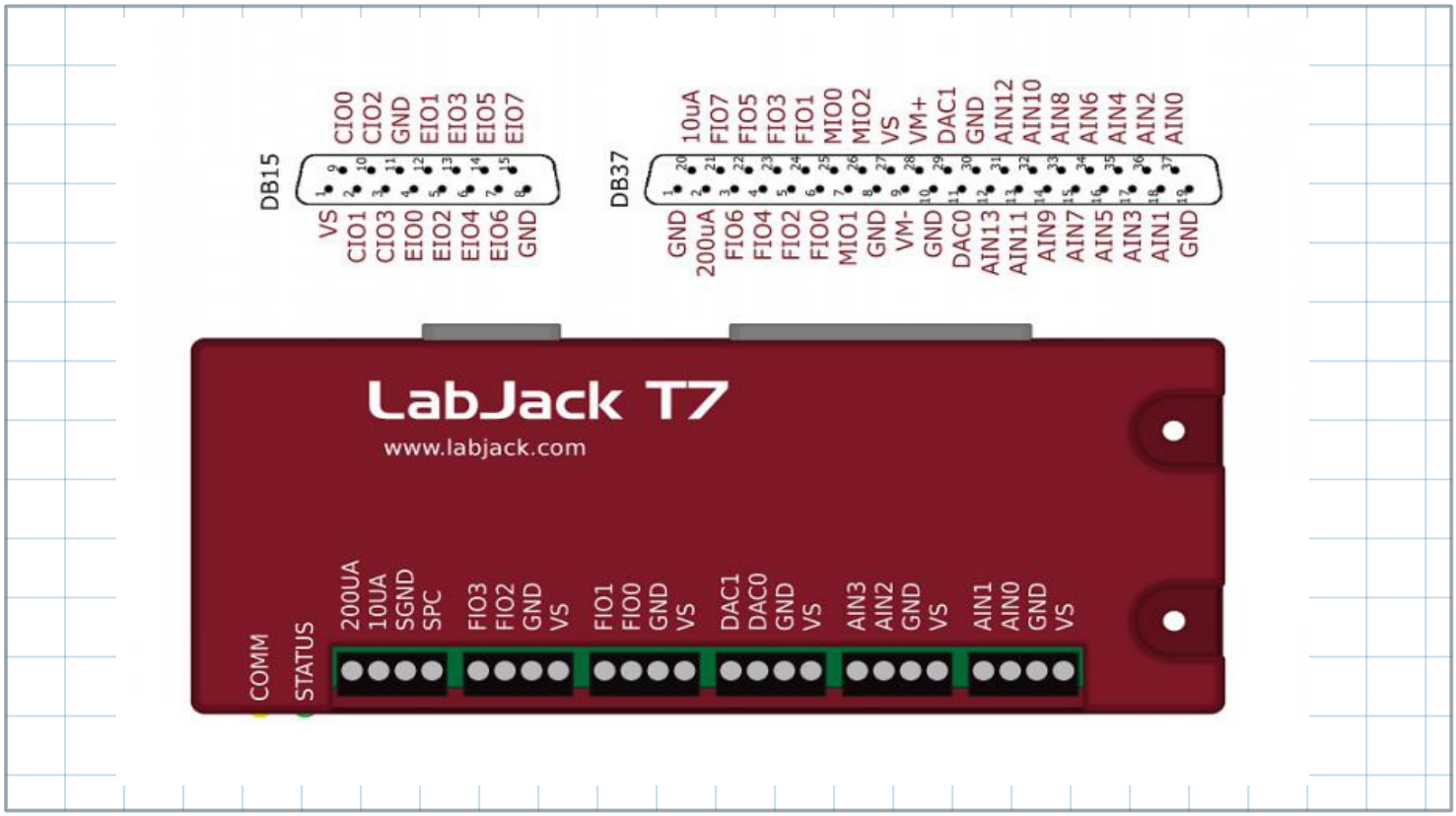
generated using the DIO extended features, PWM output at DIO0(FIO0) and a high-speed counter at the DIO18 output. These extended features are explained in detail in the following links:
https://labjack.com/support/datasheets/t7/digital-io/extended-features
https://labjack.com/support/datasheets/t7/digital-io/extended-features/pwm-out
https://labjack.com/support/datasheets/t7/digital-io/extended-features/high-speed-counter
Second, the output signal is measured using the LabJack T7 device, stream reading at AIN0. Finally, the noise in the communications is simulated using DAC0 to add a signal close to the output.

MatDeck supports LabJack functions which can be used directly inside MatDeck script to configure LabJack devices, and to generate and acquire signals from the electronic circuits as described above. Here, details about the configuration of the selected features in this experiment are explained.

In order to configure and use the device, the LabJack T7 device should be opened in the document:

```
dev := ljdevice_open("any", "any", "any")
devType := ljdevice_type(dev)
```

The function ljdevice_open() returns the handle of the opened device. We can check the type, T4 or T7, of the opened device using the ljdevice_type() function.



PWM Out configuration in MatDeck script

PWM Out at FIO0(DIO0) requires the clock source, thus the clock is first configured. There are three parameters to select for the configuration: clock source, clock divisor and the roll value for the given clock. There are three different clocks supported by T7, the most common is clock0 whose frequency is 80MHz. The clock divisor can be any power of two from 1, 2, up to 256, in this example we select value one. In this example, we use divisor value equal to one. The roll value is determined according to the desired frequency of the PWM Out signal. For example, if the desired frequency is 1kHz, the roll value is 80Hz/ Divisor/1kHz=80000. Before all values are written to the appropriate registers, the clock0 should be disabled.

```
//Disable clock0
```

```
4
5  ljdevice_write(dev,"DIO_EF_CLOCK0_ENABLE", 0)
6  //Setup Clock
7  ljdevice_write(dev,"DIO_EF_CLOCK0_DIVISOR", 1)//DIO_EF_CLOCK#_DIVISOR are
   1,2,4,8,16,32,64,or 256 pre-counter to devide 80MHz
8  ljdevice_write(dev,"DIO_EF_CLOCK0_ROLL_VALUE", 80000)//Devide 80MHz by
   8000 = 1000
```

PWM out at FIO0 (DIO0) is configured by setting 0 at DIO0_EF_INDEX. A duty cycle is set by writing the appropriate value at DIO0_EF_CONFIG_A. If the desired value of the duty cycle is 50%, then DIO0_EF_CONFIG_A will be equal to the half of the roll value, which is 40000.

```
9  //PWM Out at DIO0
10 ljdevice_write(dev,"DIO0_EF_ENABLE", 0)
11 //DIO Index valu 0 - PWM set
12 ljdevice_write(dev,"DIO0_EF_INDEX", 0)
13 //PWM duty cycle
14 ljdevice_write(dev,"DIO0_EF_CONFIG_A", 40000)
```

Configuration of High Speed Counter

High speed counter at DIO18 will be used to check the obtained PWM out frequency.

```
15 //High speed counter at DIO18
16 ljdevice_write(dev,"DIO18_EF_ENABLE", 0)
17 //DIO Index value 7-set high speed counter
18 ljdevice_write(dev,"DIO18_EF_INDEX", 7)
```

Enable Extended Features

Configured Extended Features should be enabled, this is done by writing one to the ENABLE register:

```
19 //Enable and run clock DIO18 and PWM DIO0
20 ljdevice_write(dev,"DIO_EF_CLOCK0_ENABLE", 1)
21 ljdevice_write(dev,"DIO18_EF_ENABLE", 1)
22 ljdevice_write(dev,"DIO0_EF_ENABLE", 1)
23 a := ljdevice_last_error("s") //Check for errors in configuration
```

```
a = "LJ_SUCCESS"
```

Check PWM Out Frequency

The frequency of the PWM Out signal can be checked by reading the value of the counter after one second has passed

```
24 // Wait for one second
25 sleep(1000)
26 // Read the counter
27 counter := ljdevice_read(dev, "DIO18_EF_READ_A")
```

```
counter = 1020    Hz
```

Adding interference

The interference is simulated using a analog output which is produced using DAC0. The output of DAC0 is generated using the stream output, which is configured here.

```
ljdevice_write(dev, "STREAM_OUT0_ENABLE", 0)
// Configure stream -out
// Allocate memory for the stream-out buffer
ljdevice_write(dev, "STREAM_OUT0_TARGET",ljname2address("DAC0"))
ljdevice_write(dev, "STREAM_OUT0_BUFFER_SIZE",512)
ljdevice_write(dev, "STREAM_OUT0_ENABLE", 1)
STREAM_OUT0_LOOP_VALUES := [0, 5, 0, 5, 0, 5, 0, 5, 0, 5]
// Write stream-out
ljdevice_write(dev, "STREAM_OUT0_LOOP_SIZE", 10)
ljdevice_awrite(dev, "STREAM_OUT0_BUFFER_F32", STREAM_OUT0_LOOP_VALUES)
ljdevice_write(dev, "STREAM_OUT0_SET_LOOP", 1)
```

Reading values in the stream mode

The signal at the output, OUT1 at schematics, is read by AIN0 in the stream mode. The stream mode is set by writing in the appropriate registers. Analog inputs have four values to set: resolution index, settling time, range, and single-ended or differential reading. In the stream mode, resolution index and settling time are set for all streamed analog inputs, while range and negative channel are configured separately.

```
// Configure Stream Mode
ljdevice_write(dev, "STREAM_TRIGGER_INDEX", 0)
ljdevice_write(dev, "STREAM_CLOCK_SOURCE", 0)
ljdevice_write(dev, "STREAM_RESOLUTION_INDEX", 0)
ljdevice_write(dev, "STREAM_SETTLING_US", 0)
ljdevice_write(dev, "AIN_ALL_RANGE", 0)
ljdevice_write(dev, "AIN_ALL_NEGATIVE_CH", 199)
```

The function ljdevice_stream_start() is used to prepare the device for reading, the user has to specify scans per read, and scan rate. The function ljdevice_stream_read() is used to obtain data. If multiple channels are streamed, all the data is stored in a single vector, thus reshape is used to separate the channels.

```
ljdevice_stream_start(dev, 10000, ["DIO0"; "AIN0"; "STREAM_OUT0"], 10000)
pwm := ljdevice_stream_read(dev, 20000)
ljdevice_stream_stop(dev)
ch := mat_reshape(pwm, [size(pwm) / 2, 2])
```

Finally, at the end, all the extended features should be disabled and the device should be closed.

```
//Disable clock0, high speed counter, PWM out and close the device
ljdevice_write(dev,"DIO_EF_CLOCK0_ENABLE", 0)
ljdevice_write(dev,"DIO18_EF_ENABLE", 0)
ljdevice_write(dev,"DIO0_EF_ENABLE", 0)
ljdevice_close(dev)
```

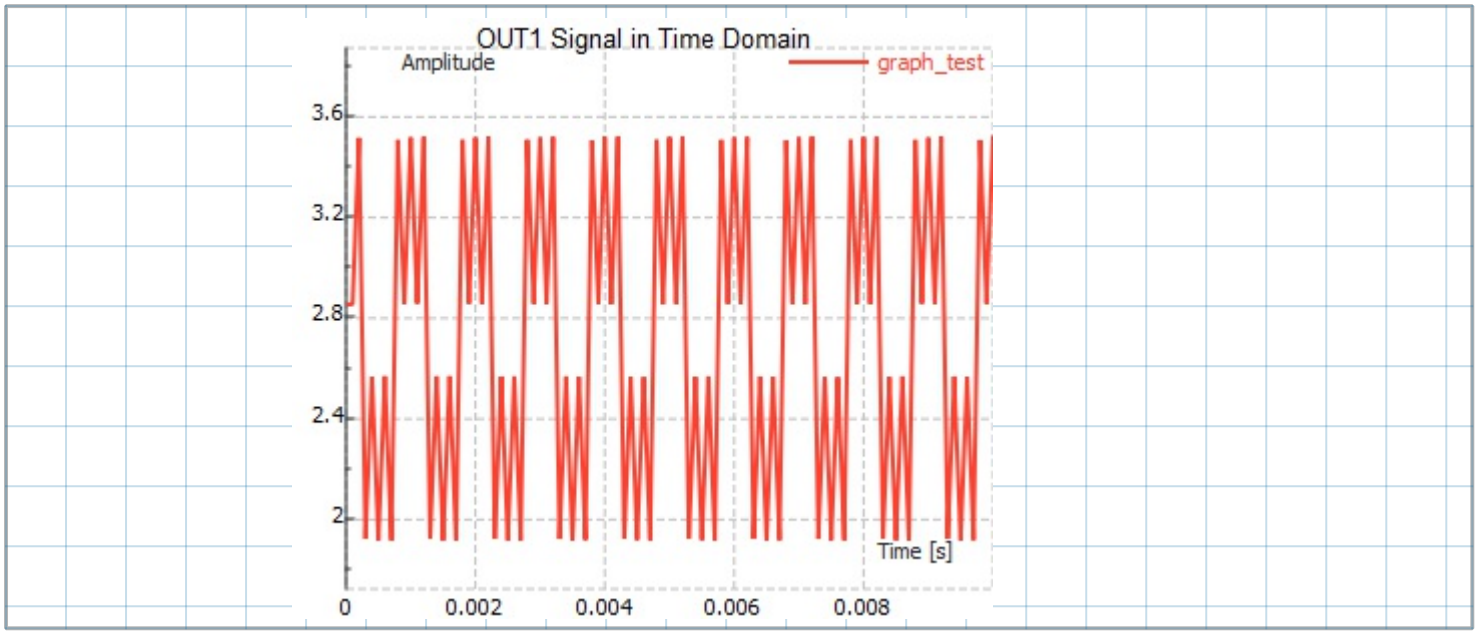
pwm = 20000 elements vector

Signal Processing and Visualization

The obtained signal is visualized using a MatDeck graph. It is evident from looking at the graph that the signal is corrupted with interference.

```
ttime := ynodes(x, 0, 9999 / 10000, 10000)
IN1 :=subset(ch, 0, 0, size(ch) / 2 -1, 0 )
```

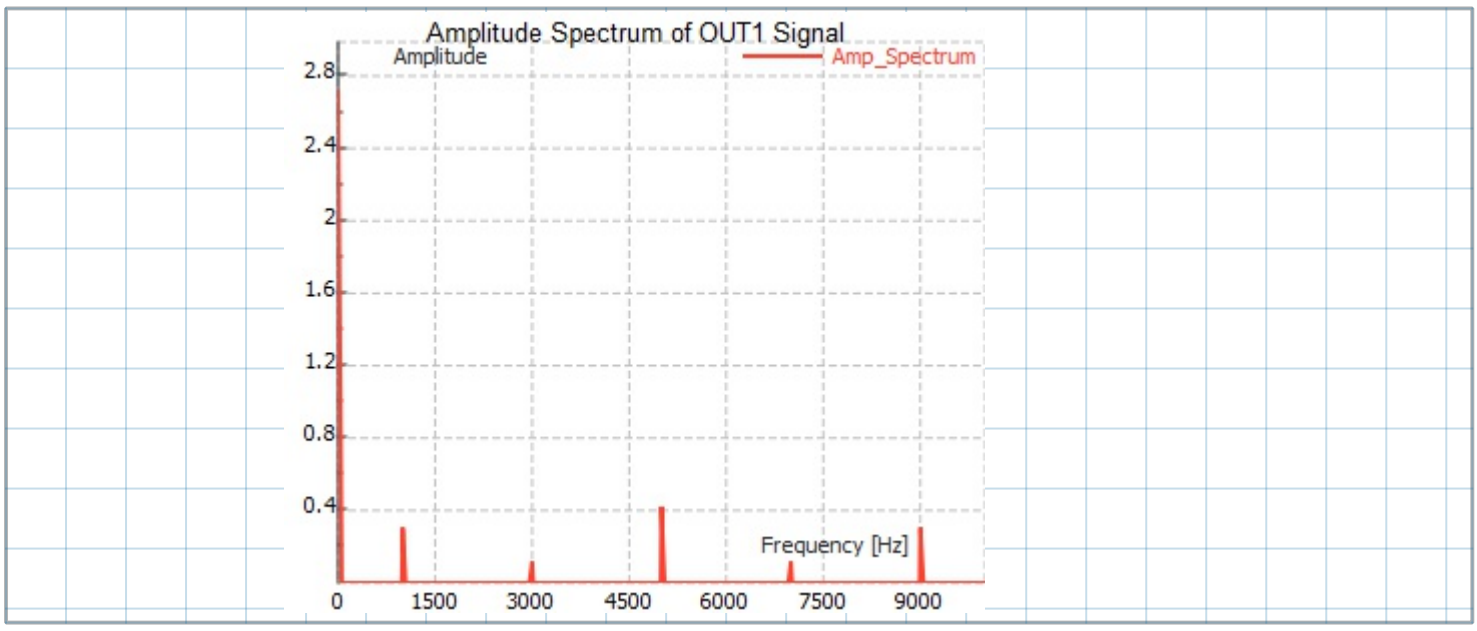
```
57
58 OUT1 :=subset(ch, 0, 1, size(ch) / 2 -1, 1)
59 graph := join_mat_cols(ttime, IN1)
60 graph1 := join_mat_cols(ttime, OUT1 )
61 graph_test := subset(graph1, 0, 0, 100, 1)
```



Signal Processing using MatDeck Digital Filter

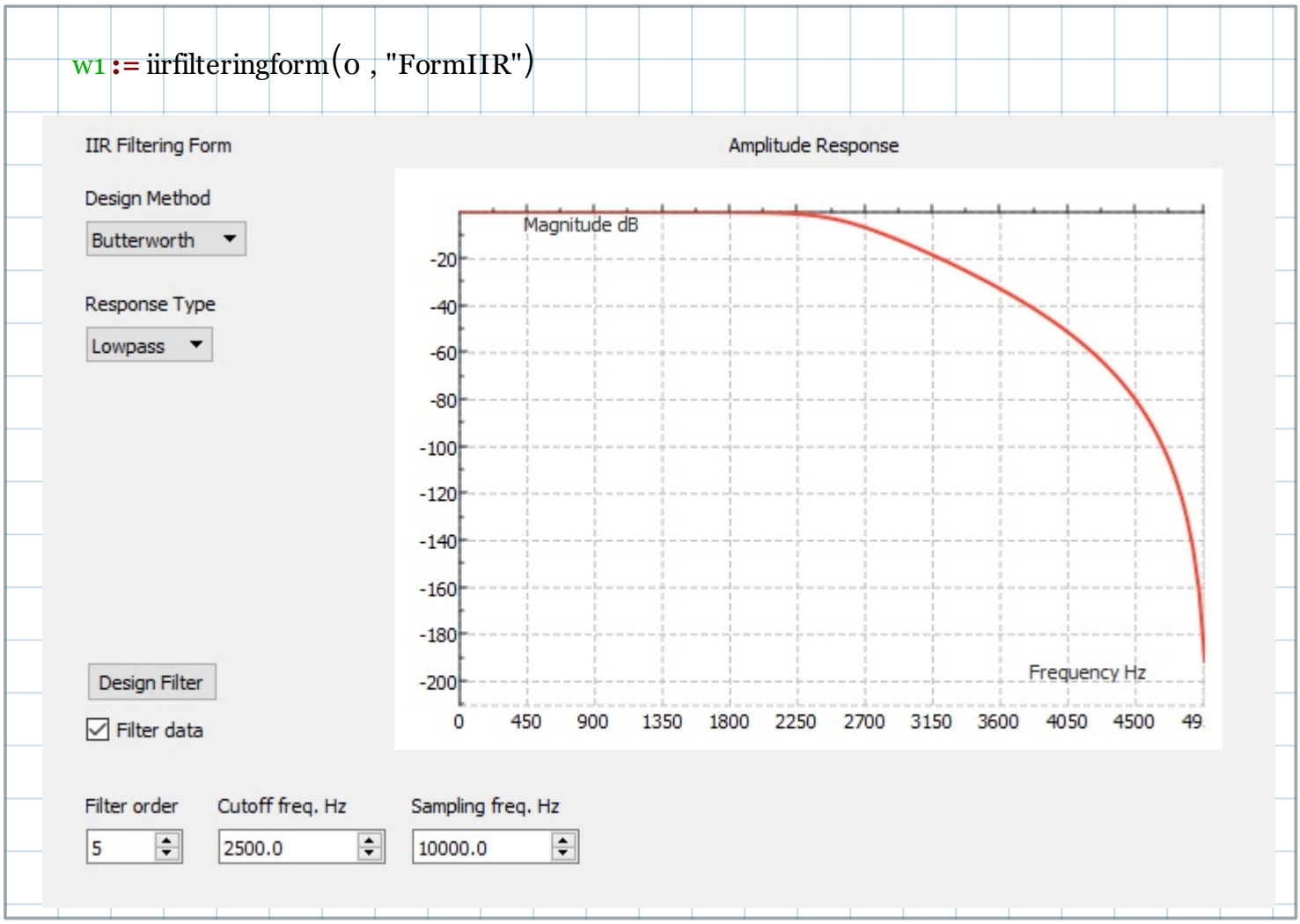
It is obvious from the time domain graph, that it is corrupted with high frequency interference. However, it is possible to check the amplitude spectrum of the signal using the MatDeck function, fft(). The amplitude spectrum shows that the frequency of the interference in Hz. The information is used to design the filter to remove interference from the signal.

```
62 Amp_Spectrum := abs(fft1(OUT1)) / 10000
63 Fre := ynodes(x, 0, 9999, 10000)
64 Amp_Sgraph := join_mat_cols(Fre, Amp_Spectrum)
```



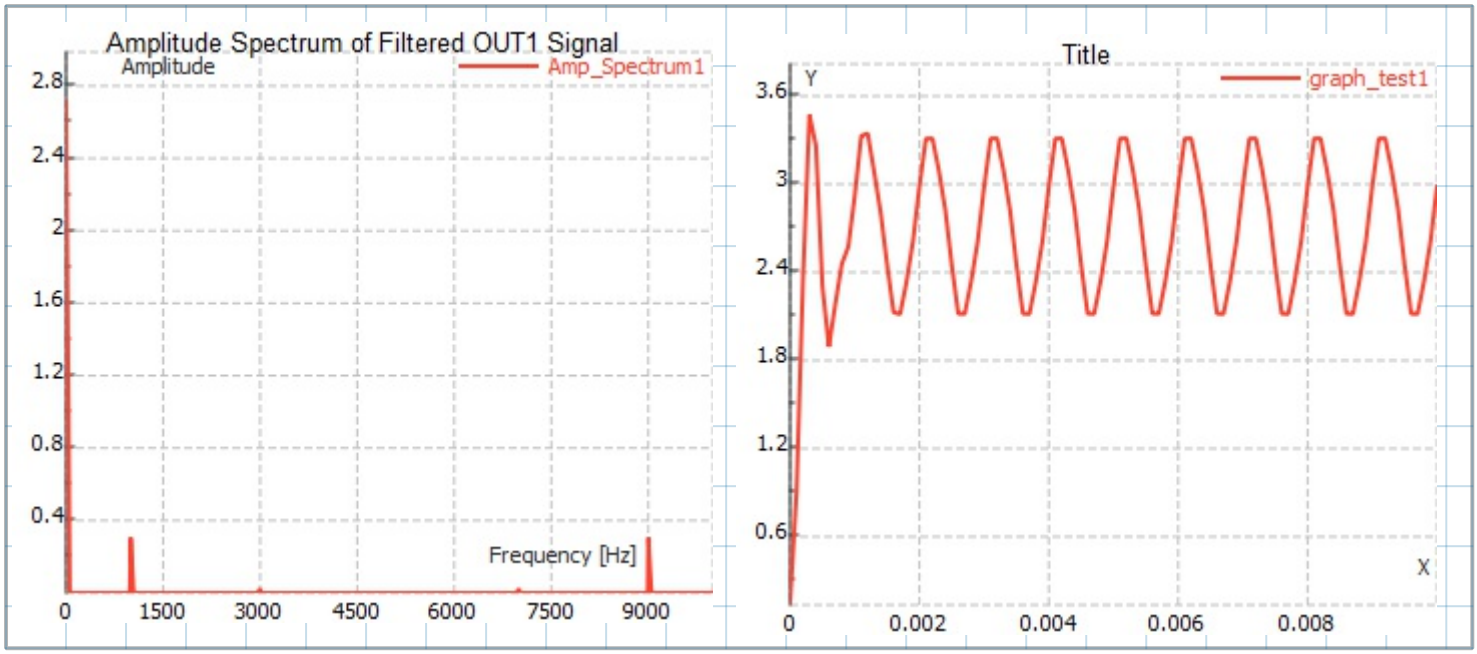
The idea is to use a simple low pass Butterworth filter of the fourth order, with the cutoff frequency equal to 2500Hz. In MatDeck, we can use the IIR form to design the aforementioned filter and to apply the filter to

the signal. In order to use the form, the user has to create the form using the function f:=iirfilteringform(0,"form1"). Here, the first argument is the widget parent or 0, and the second argument is the string form name. The function returns the identification of the widget form. The created IIR Filtering Form is placed in the Canvas at the position where the function embed_widget() is called with the appropriate widget identity. As explained, the IIR filtering form can be used to filter OUT1 signal. In order to perform filtering, the Filter data check box has to be checked. The results are derived by calling the functions iirfilteringresult(w1,OUT1), where f is the identification of the widget, x is the input signal and y is the output signal.



What is left, at the end, is to visualize the time domain and amplitude spectrum of the filtered OUT1 signal.

```
OUT1_filtered :=iirfilteringresult(w1, OUT1)
Amp_Spectrum1 := abs(fft1(OUT1_filtered)) / 10000
Amp_Sgraph1 := join_mat_cols(Fre, Amp_Spectrum1)
Time_OUT1f := join_mat_cols(ttime, OUT1_filtered)
graph_test1 := subset(Time_OUT1f, 0, 0, 100, 1)
```


Amplitude Spectrum of Filtered OUT1 Signal
Amplitude
Amp_Spectrum1
2.8
2.4
2
1.6
1.2
0.8
0.4
Frequency [Hz]
0
1500
3000
4500
6000
7500
9000
Title
Y
graph_test1
3.6
3
2.4
1.8
1.2
0.6
X
0
0.002
0.004
0.006
0.008