Wave dump using digitizer component

Abstract

The digitizer block in SciCompiler works as a programmable size FIFO to store waveform. It is possible to define, compiling time, the maximum number of channels, while at runtime it is possible to select which channels effectively dump.

In respect to the oscilloscope, this modules allows a faster way to dump waveform and have the possibility to partition the total amount of samples in the FIFO on multiple channels.

1 Structure of the digitizer block

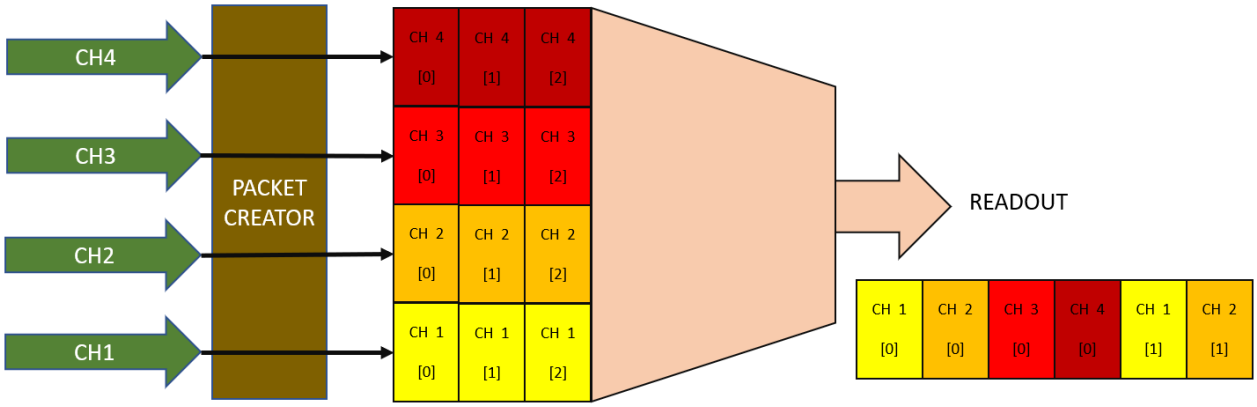
The digitizer block is designed as following: a packet creator block serializes the enabled channels in a common FIFO and transfer data to the PC.

Every time a start signal is triggered, the timestamp, the hits and user data are captured, and enqueued in the output FIFO.

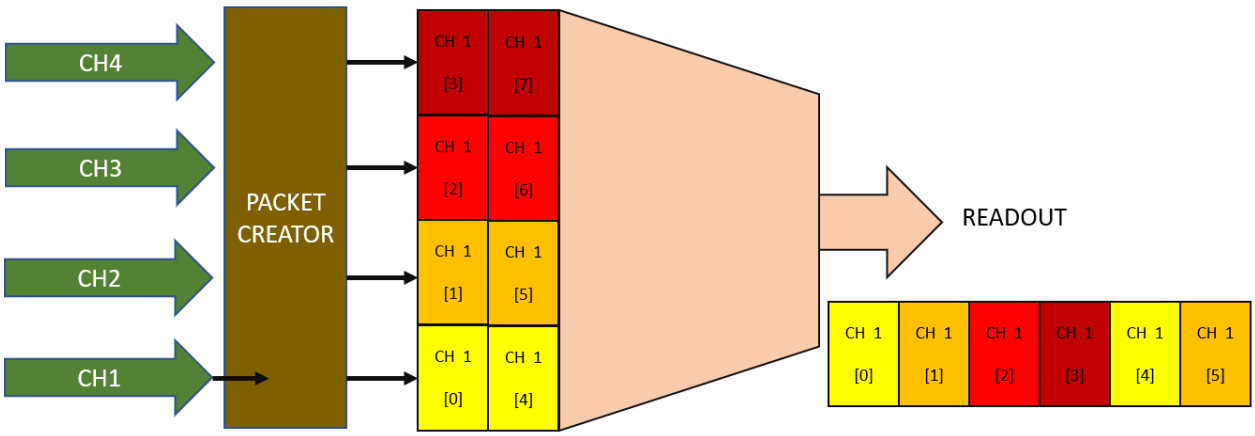
The packet creator is designed to optimize the usage of the output FIFO. It is possible to select the number of channels to download a runtime in order to do no waste FIFO area on not used channels.

It is only possible the specify the number of channels (N) starting from the channel 0. So If N = 2, CH0 and CH1 will be dumped, if N = 4, CH0, CH1, CH2, CH3 will be dumped.

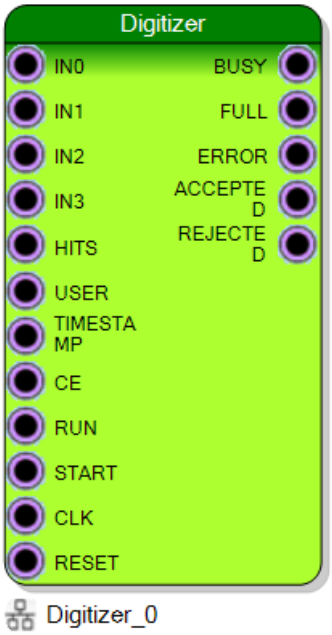
Four channel data acquisition



Single channel data acquisition



2 Digitizer component



IN	VECT →	Size: 16

Waveform data, one for each channel enabled

HITS	VECT →	Size: 64

General purpose 64bit register. Typically used to store which channels fired the trigger, but can be used for any other indication. It is captured on the rising edge of the start

USER	VECT →	Size: 32

General purpose 32bit register. It is captured on the rising edge of the start

TIMESTAMP	VECT →	Size: 64

Connect to the board or system timestamp generator in order to synchronize acquisition to a global timing

CE	VECT →	Size: 1

Enable data point storage

START	VECT →	Size: 1

Input signal enabling the digitization process

BUSY	VECT →	Size: 1

Output signal indicating when HIGH that the digitization process is occurring

FULL	VECT →	Size: 1

Output signal indicating when HIGH that the internal memory is full and can not acquire other waveform

ERROR	VECT →	Size: 1

Settings are not compliant to the configuration

ACCEPTED	VECT →	Size: 1

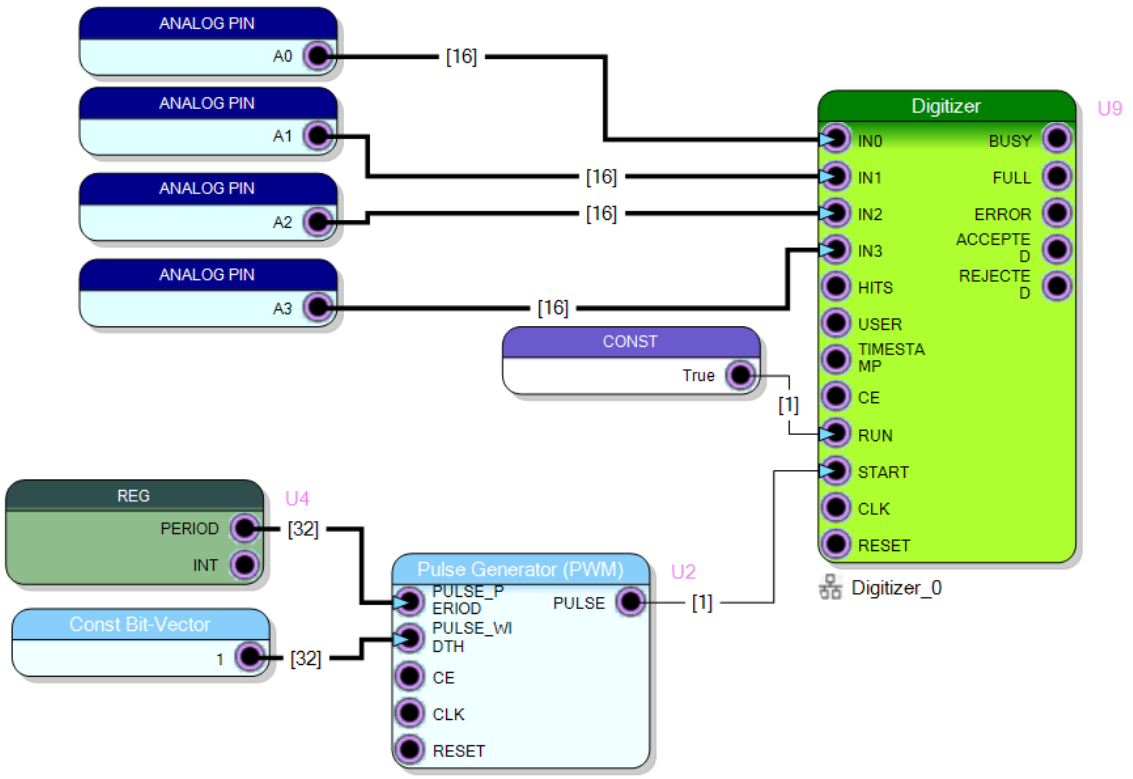
Last START has been accepted by the digitizer block and the full waveform will be captured and transmitted

REJECTED	VECT →	Size: 1

Last START has been rejected

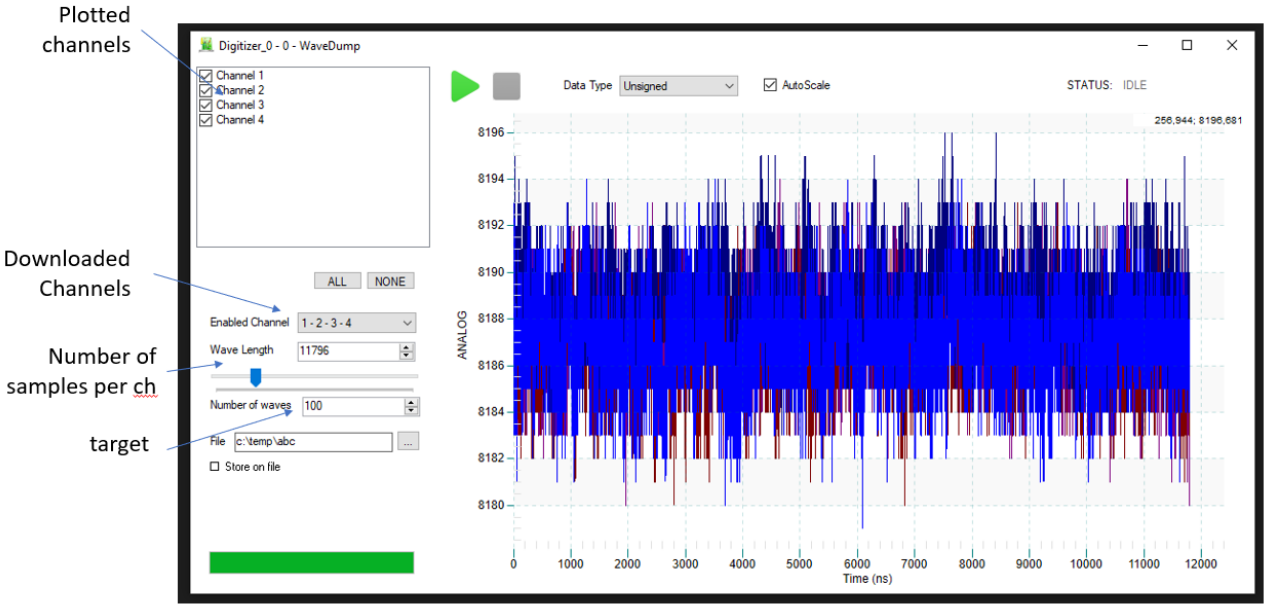
3 Example design

In the following example the digitizer is configured to acquire 16 bits analog data from 4 channels. Timestamp, hits, and user data are not connected indeed they will be zero. Run is fixed to TRUE, while the start is triggered by the pulse generator with a programmable period.



3 Data acquisition using Resource Explorer

In the following example the digitizer is configured to acquire 16 bit analog data from 4 channels.

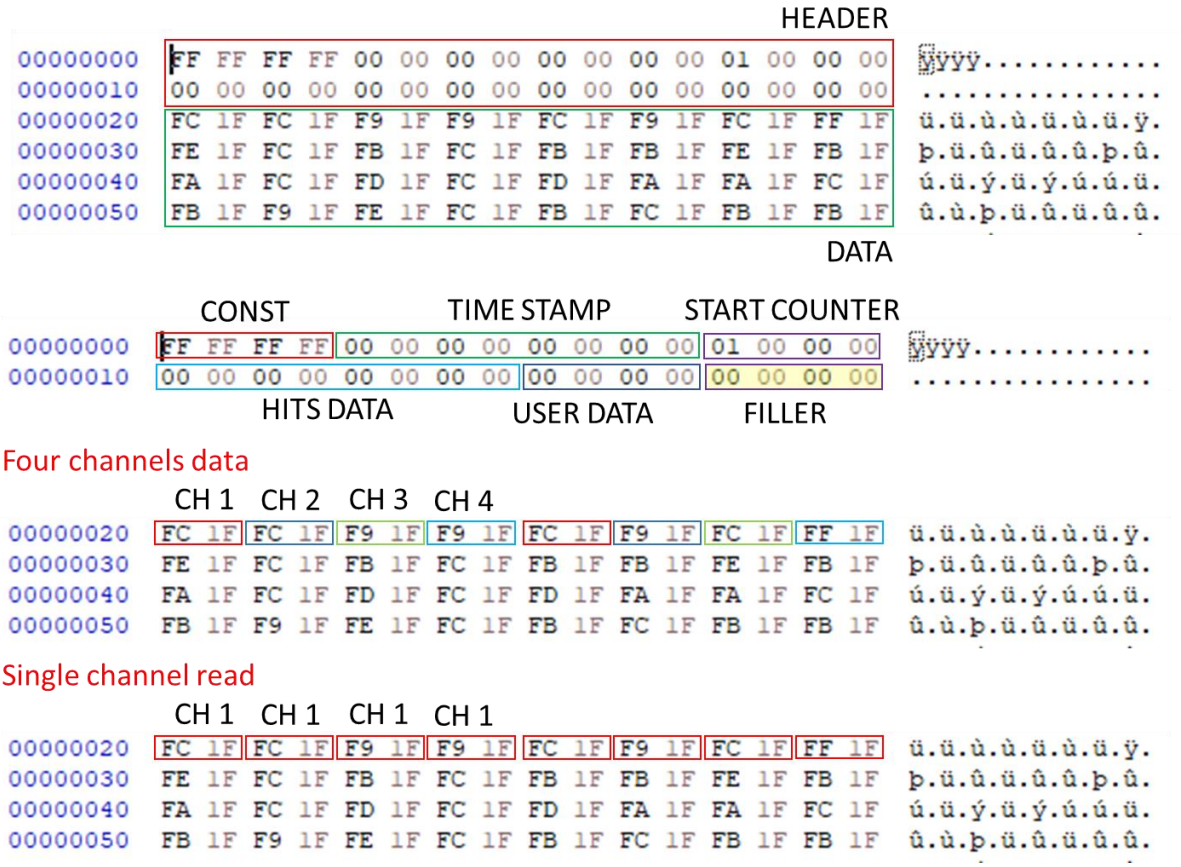


4 Data acquisition using C++

```
	//How many waves acquire (20)
	uint32_t TargetWaveNumber = 20;
	//Enable channels 1,2,3,4
	uint32_t ChannelsEnable = 4;
	//How many samples per wave (1000)
	uint32_t WaveformLen = 4000;

	size_list = (ChannelsEnable*WaveformLen + 10);
	data_list = malloc(size_list * sizeof(uint32_t));
	TargetDataNumber = size_list * TargetWaveNumber/2;



	R_Init();


	if(R_ConnectDevice(BOARD_IP_ADDRESS, 8888, &handle) != 0) { printf("Unable to connect
to the board!\n"); return (-1); };

	//Se the pulse generator to generate 1 pulse every 10000 clock cycles
	REG_PERIOD_SET(10000, &handle);

	//Set Digitizer wave Len
	LISTMODULE_Digitizer_0_SetLen(&handle, WaveformLen);

	//Set Digitizer enabled channels and start acquisition
	LISTMODULE_Digitizer_0_START(&handle, ChannelsEnable);

	//Dump Data and write on file
	fopen_s(&fp,"c:\\temp\\data.hex", "wb");
	printf("Start download\n");
	while (TargetDataNumber > 0) {
		if (LISTMODULE_Digitizer_0_DOWNLOAD(data_list, size_list, timeout_list,
&handle, &read_data_list, &valid_data_list) != 0) printf("Get Data Error");
		if (valid_data_list > 0) {
			fwrite(data_list, 4, valid_data_list, fp);
			printf("."); fflush(stdout);
		}
		TargetDataNumber -= valid_data_list;
	}

	fclose(fp);
```

5 RAW data format

The file dumped from the script above will save the file as a list of events. Every event is triggered by a L->H commutation of the START input.



FILLER

The filler field size is variable and depends on the MAXIMUM number of channels selected at compiling time and does not depends by the effective number of channels configured to be readout. If in SciCompiler you configure the Digitizer to have four input, the filler will be 1 independently from what is set by the API `LISTMODULE_Digitizer_0_SetLen`

Number of channels	Physical FIFO width (in DWORD)	Filler size (in DWORD)
1	1	0
2	1	0
4	2	1
8	4	1
16	8	1
32	16	9
64	32	25

6 Python decode software

IP configured for 4 channels, acquisition enabled for 4 channels

```
import numpy as np

import matplotlib.pyplot as plt

f = open("data_4chc.hex", "r")
a = np.fromfile(f, dtype=np.uint32)

wave_len = 8000                         ## Number of samples per channels

filler_size = 1                         ## IP CONFIGURED FOR 4 CH
                                        ## Filler is 1

state = 0
idx=0
ch0 = []
ch1 = []
ch2 = []
ch3 = []
for x in a:
    if (state==0):                      ## CHECK HEADER
        if (x==0xFFFFFFFF):
            state = 1
            start_counter = 0
            hits = 0
            user =0
            ts = 0
            icnt = 0
            filler_cnt =filler_size
            wavec = wave_len
        else:
            print ("DECODE ERROR")
    elif (state==1):                    ## GET TS LSB
        ts = x
        state = 2
    elif (state==2):                    ## GET TS MSB
        ts += x << 32
        state = 3
    elif (state==3):                    ## GET START COUNTER
        start_counter = x
        state = 4
    elif (state==4):                    ## GET HITS LSB
        hits = x
        state = 5
    elif (state==5):                    ## GET HITS MSB
        hits += x << 32
        state = 6
```

```
    elif (state==6):                    ## GET USER FIELD
        user = x
        if (filler_cnt>0):
            state = 7
        else:
            state = 8
    elif (state==7):                    ## FILLER
        filler_cnt = filler_cnt - 1
        if (filler_cnt==0):
            state = 8
    elif (state==8):                    ## RECEIVER DATA AD DECODE CHs
        if icnt == 0:
            ch0.append(x&0xFFFF)
            ch1.append((x>>16)&0xFFFF)
            icnt = 1
        elif icnt == 1:
            ch2.append(x&0xFFFF)
            ch3.append((x>>16)&0xFFFF)
            icnt = 0
            if wavec == 0:
               state = 0
               idx=idx +1
            else:
                wavec = wavec-1




fig, axes = plt.subplots(nrows=2, ncols=1)

print("Total waveform in the file: " + str(idx))

axes[0].plot(ch0)
axes[1].plot(ch1)

plt.show()
```

IP configured for 4 channels, acquisition enabled for 1 channel

```
import numpy as np

import matplotlib.pyplot as plt

f = open("data_1ch.hex", "r")
a = np.fromfile(f, dtype=np.uint32)
```

```
state = 0
wave_len = 3000/2                       ## NUMBERF OF SAMPLE
                                        ## MUST BE DIVIDED BY 2
                                        ## WHEN ONLY 1 CHANNEL IS ENABLE
idx=0
ch0 = []
filler_size = 1
for x in a:
    if (state==0):
        if (x==0xFFFFFFFF):
            state = 1
            start_counter = 0
            hits = 0
            user =0
            ts = 0
            filler_cnt =filler_size
            wavec = wave_len
        else:
            print ("DECODE ERROR")
    elif (state==1):
        ts = x
        state = 2
    elif (state==2):
        ts += x << 32
        state = 3
    elif (state==3):
        start_counter = x
        state = 4
    elif (state==4):
        hits = x
        state = 5
    elif (state==5):
        hits += x << 32
        state = 6
    elif (state==6):
        user = x
        if (filler_cnt>0):
            state = 7
        else:
            state = 8
    elif (state==7):
        filler_cnt = filler_cnt - 1
        if (filler_cnt==0):
            state = 8
        state = 8
    elif (state==8):
```

```
        if wavec == 1:
            state = 0
        else:
            ch0.append(x&0xFFFF)
            ch0.append((x>>16)&0xFFFF)
            wavec = wavec-1

    idx=idx +1
fig, axes = plt.subplots(nrows=1, ncols=1)



axes.plot(ch0)


plt.show()
```

IP configured for 2 channels, acquisition enabled for 2 channel

```
import numpy as np

import matplotlib.pyplot as plt

f = open("data_2ch.hex", "r")
a = np.fromfile(f, dtype=np.uint32)
state = 0
wave_len = 3000
idx=0
ch0 = []
ch1 = []
filler_size = 0                         ## FILLER MUST BE 0
                                        ## BECAUSE IP IS CONFIGURED
                                        ## FOR 2 CHANNELS
for x in a:
    if (state==0):
        if (x==0xFFFFFFFF):
            state = 1
            start_counter = 0
            hits = 0
            user =0
            ts = 0
            filler_cnt =filler_size
            wavec = wave_len
        else:
            print ("DECODE ERROR")
    elif (state==1):
```

```
        ts = x
        state = 2
    elif (state==2):
        ts += x << 32
        state = 3
    elif (state==3):
        start_counter = x
        state = 4
    elif (state==4):
        hits = x
        state = 5
    elif (state==5):
        hits += x << 32
        state = 6
    elif (state==6):
        user = x
        if (filler_cnt>0):
            state = 7
        else:
            state = 8
    elif (state==7):
        filler_cnt = filler_cnt - 1
        if (filler_cnt==0):
            state = 8
        state = 8
    elif (state==8):
        if wavec == 0:
            state = 0
        else:
            ch0.append(x&0xFFFF)
            ch1.append((x>>16)&0xFFFF)
            wavec = wavec-1

    idx=idx +1
fig, axes = plt.subplots(nrows=2, ncols=1)



axes[0].plot(ch0)
axes[1].plot(ch1)

plt.show()
```