Usage of FIFO and Custom Packet with SciCompiler

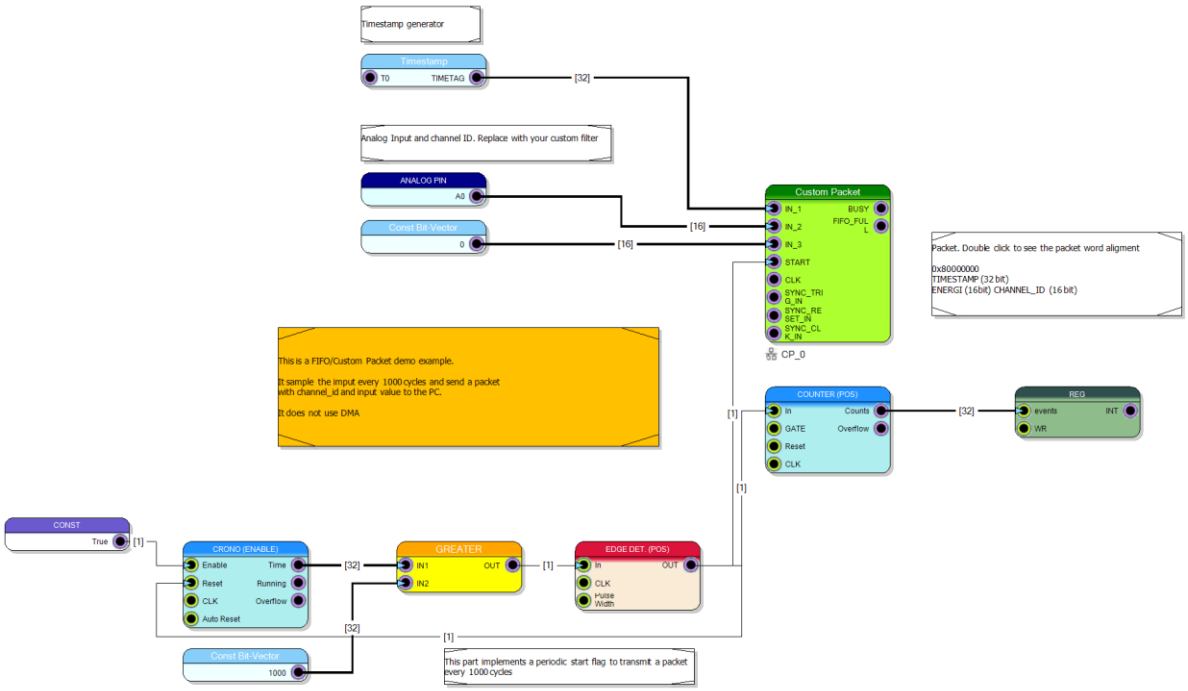
SciCompiler allows to easy implement FIFO communication to transfer formatted date from FPGA to user computer

There are two way to implement this transfer:

- Using pure FIFO and creating a custom protocol with state machines or arbiters
- Using the Packet generator

We strongly suggest to use packet generator because the tool automatically format data and enqueue it in a FIFO buffer as stream of packets. The structure of the packet can be customized by the user using the Custom Packet tool. Either input channels, timestamp, processed data, constant can be inserted in the packet

Consider this design as a reference design

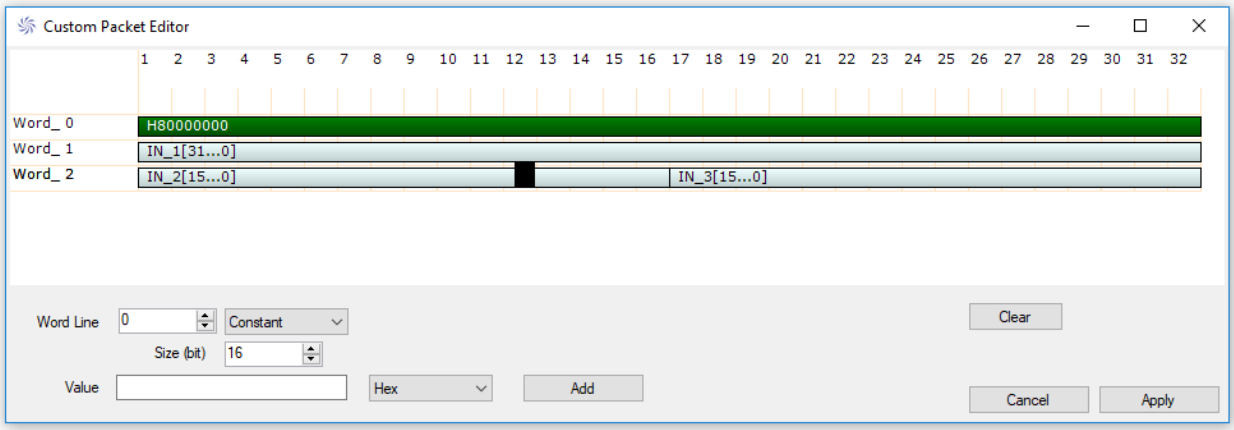


The project can be download from github: https://github.com/NuclearInstruments/sci_custom_packet

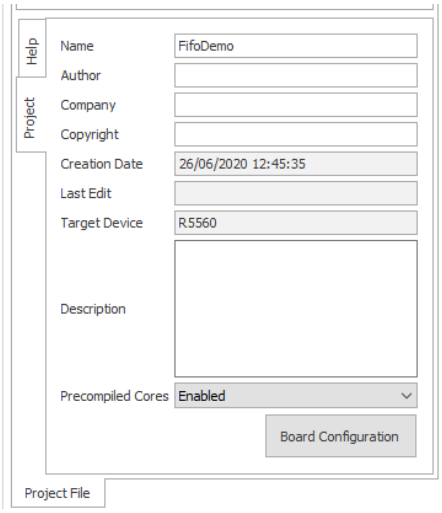
The Custom Packet (Green) acquire data from Timestamp generator, Channel A0 of the instrument, a constant (0) that represents the channel.
The bottom part of the design implements a periodic (1 every 1000 clock cycles) trigger generator and

event counter (counter + register). This circuit toggle the START signal triggering the data transferring from Custom Packet to PC

The Custom Packet Editor (Double Click on Custom Packet) allow to define the packet layout

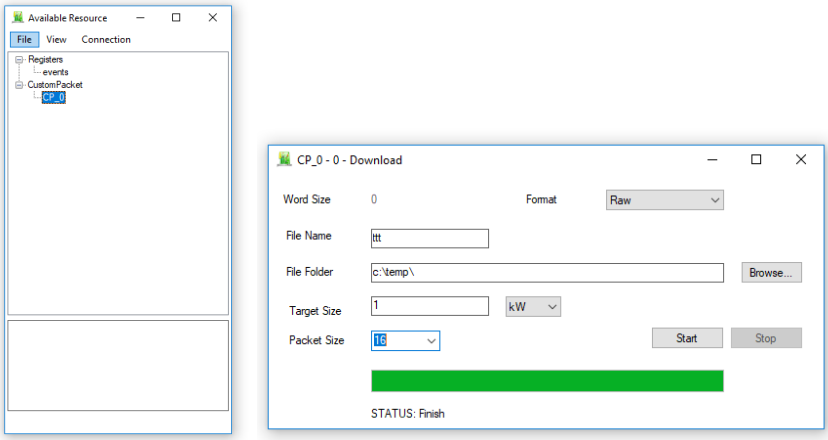


Before compile the project remember to enable the Precompiled Cores in the project properties. It speed up of a factor of 10 the compilation of the FPGA project



You can easy test the data transferring using the Resource Explorer tool. Compile the project, open Resource Explorer tool from SciCompiler and connect to the hardware.

In the resources list select Custom Packet and open the control tool



Insert a path and, raw format and start acquisition

Open the saved file with and Hex editor like HXD and have a look to the content of the file

Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	
00000000	00	00	00	80	18	5B	44	31	00	00	86	1F	...€.[D1..†.
0000000C	00	00	00	80	05	5F	44	31	00	00	89	1F	...€._D1..‰.
00000018	00	00	00	80	F2	62	44	31	00	00	87	1F	...€òbD1..‡.
00000024	00	00	00	80	DF	66	44	31	00	00	8A	1F	...€ßfD1..Š.
00000030	00	00	00	80	CC	6A	44	31	00	00	8A	1F	...€ÌjD1..Š.
0000003C	00	00	00	80	B9	6E	44	31	00	00	86	1F	...€¹nD1..†.
00000048	00	00	00	80	A6	72	44	31	00	00	8A	1F	...€¦rD1..Š.
00000054	00	00	00	80	93	76	44	31	00	00	88	1F	...€"vD1..ˆ.
00000060	00	00	00	80	80	7A	44	31	00	00	88	1F	...€€zD1..ˆ.
0000006C	00	00	00	80	6D	7E	44	31	00	00	8D	1F	...€m~D1....

Green : Key words Red: Timecode Yellow: channel Cyan: Analog Value

SDK

Open the FifoDemo.sln from folder FifoDemo\library\C\lib\VC++ in Visual Studio 2015 or newer

SciCompiler already created for you all necessary code to interface with the Custom Packet, including a circular buffer to download the data and a function to process your signals. You have just to edit few lines of code.

You have to make some customization in FifoDemo_LIB\FifoDemo_LIB.c edit

Costumize the structure `t_FRAME_packet` to correctly handle our data

```
typedef struct
{
    uint16_t ch;
    uint16_t analog;
} t_data;
typedef struct
{
    uint32_t Time_Code;
    t_data *data;
    uint32_t Valid;
```

```
} t_FRAME_packet;
```

Edit the example function `CPACK_CP_0_RECONSTRUCT_DATA`. This function decode raw data in packets structured as collections of `t_FRAME_packet`

```
SCILIB int CPACK_CP_0_RECONSTRUCT_DATA(void *buffer_handle, t_FRAME_packet_collection *decoded_packets)
{
    cbuf_handle_t cbuf;
    cbuf = (cbuf_handle_t)buffer_handle;
    int n_ch = 1;
    //the packet size is equal to
    // n_ch*1 (32 bit word) + header (1x32 bit word) + timestamp  (1x32 bit word)
    int PacketSize = n_ch + 2;
    int in_sync = 0;
    uint64_t event_timecode = 0;
    uint32_t ev_energy = 0;
    uint32_t mpe = 0;
    int ch_index = 0;
    int i = 0,j;
    int k = 0;
    //check if we have elements in the circular buffer
    if (circular_buf_size(cbuf) < PacketSize) return -1;
    //allocate output data packets. Extimate extra space for extra
    //packets
    int possible_packets = (circular_buf_size(cbuf) / PacketSize)+10;
    decoded_packets->packets = (t_FRAME_packet *)malloc(possible_packets * sizeof(t_FRAME_packet));
    if (decoded_packets->packets==NULL) return -2;

    //Allocate memory for multiple channels in the packet
    for (i = 0; i < possible_packets; i++)
    {
        decoded_packets->packets[i].data = (uint32_t *)malloc(n_ch * sizeof(t_data));
        if (decoded_packets->packets[i].data == NULL)
        {
            for (j = 0; j < i; j++)
            {
                if (decoded_packets->packets[i].data !=NULL)
                    free(decoded_packets->packets[i].data);
            }
            if (decoded_packets->packets != NULL)
                free(decoded_packets->packets);

            return -2;
        }
    }
    decoded_packets->allocated_packets = possible_packets;
    decoded_packets->valid_packets = 0;

    //process packets
    while (circular_buf_size(cbuf)> PacketSize)
    {
        circular_buf_get(cbuf, &mpe);

        if (in_sync == 0) {
            //Check for header, if not wait
            //for an header
            if (mpe != 0x80000000)
            {
                continue;
            }
            in_sync = 1;
            ch_index =0;
            continue;
        }
        if (in_sync == 1) {
            //Read timecode (first word)
            decoded_packets->packets[k].Time_Code =  mpe;
            in_sync = 2;
            continue;
        }
        if (in_sync == 2) {
```

```
            //Read packet data (analog + channel)
            //if packet is broken and a new packet early
            //begin, trash packet and decode the new one
            if (mpe == 0x80000000) {
                in_sync = 1;
                ch_index =0;
            }
            else {
                decoded_packets->packets[k].data[ch_index].analog = (mpe>>16) & 0xFFFF;
                decoded_packets->packets[k].data[ch_index].ch = (mpe >> 0) & 0xFFFF;
                ch_index++;
                if (ch_index == n_ch) {
                    in_sync = 0;
                    k++;
                    decoded_packets->valid_packets++;
                }
            }
            continue;
        }
    }

 return 0;

}
```

In the end modify the function to delete unused packets `free_FRAME_packet_collection`

```
SCILIB void free_FRAME_packet_collection (t_FRAME_packet_collection *decoded_packets)
{
    int i;
    for (i = 0; i < decoded_packets->allocated_packets; i++)
    {
        free(decoded_packets->packets[i].data);
    }
    free(decoded_packets->packets);
}
```

Right click on the library project and compile it!

Now modify the Example file Fifodemo\fifodemo_example.c. Replace the automatically generated code with this fully working example

Insert the IP address of your board

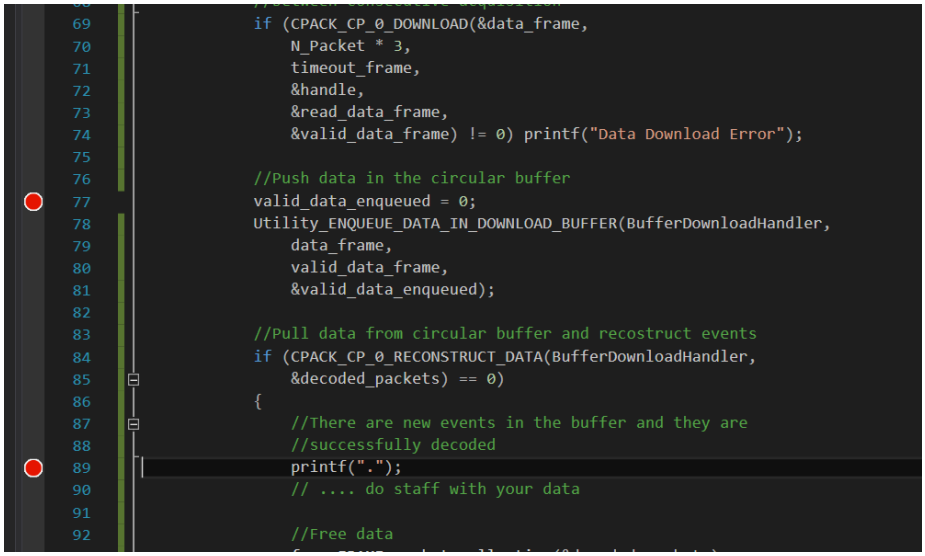
```
#include "Def.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <stdint.h>

#include  "FifoDemo_lib.h"

#define BOARD_IP_ADDRESS "192.168.50.241"



int main(int argc, char* argv[])
{
    NI_HANDLE handle;
    int ret;
    uint32_t    val;


    R_Init();
```

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

     //REMOVE THIS COMMENT TO ENABLE THE EXAMPLE CODE

    uint32_t status_frame = 0;
    uint32_t N_Packet = 100;
    uint32_t data_frame[100000];
    uint32_t read_data_frame;
    uint32_t valid_data_frame;
    uint32_t valid_data_enqueued;

    uint32_t N_Total_Events = 10000;
    uint32_t ReadDataNumber = 0;
    int32_t timeout_frame = 1000;
    t_FRAME_packet_collection decoded_packets;

    //Configuration flag
    int32_t FrameSync = 0;
    int32_t FrameWait = 0;
    int32_t FrameMask = 3;
    int32_t FrameExternalTrigger = 0;
    int32_t FrameOrTrigger = 1;

    void *BufferDownloadHandler = NULL;

    //Create the circular buffer where download raw data
    Utility_ALLOCATE_DOWNLOAD_BUFFER(&BufferDownloadHandler, 1024*1024);

    //Startup the Custom Packet acquisition
    if (CPACK_CP_0_RESET(&handle) != 0) printf("Reset Error");
    if (CPACK_CP_0_START(&handle) != 0) printf("Start Error");

    //chec if is ready
    if (CPACK_CP_0_STATUS(&status_frame, &handle) != 0) printf("Status Error");
    if (status_frame >0)
    {
        //Forever download
        while (1)
        {
            valid_data_frame = 0;

            //Download N_packet raw data. There is no guarantee that the data
            //are packet aligned. The circular buffer make continuty
            //between consecutive acquisition
            if (CPACK_CP_0_DOWNLOAD(&data_frame,
                N_Packet * 3,
                timeout_frame,
                &handle,
                &read_data_frame,
                &valid_data_frame) != 0) printf("Data Download Error");

            //Push data in the circular buffer
            valid_data_enqueued = 0;
            Utility_ENQUEUE_DATA_IN_DOWNLOAD_BUFFER(BufferDownloadHandler,
                data_frame,
                valid_data_frame,
                &valid_data_enqueued);

            //Pull data from circular buffer and recostruct events
            if (CPACK_CP_0_RECONSTRUCT_DATA(BufferDownloadHandler,
                &decoded_packets) == 0)
            {
                //There are new events in the buffer and they are
                //successfully decoded
                printf(".");
                // .... do staff with your data

                //Free data
                free_FRAME_packet_collection(&decoded_packets);
            }
```

```
            ReadDataNumber = ReadDataNumber+ N_Packet;
    }
        printf("Download completed");
    }
    else printf("Status Error");



#else

#endif

    return 0;
}
```

Run the example. You should see a console with several ……. populating the shell.

Just put a breakpoint after `CPACK_CP_0_DOWNLOAD` and `CPACK_CP_0_RECONSTRUCT_DATA`. You will be able to see raw acquired data and decoded packets.



```
69              if (CPACK_CP_0_DOWNLOAD(&data_frame,
70                  N_Packet * 3,
71                  timeout_frame,
72                  &handle,
73                  &read_data_frame,
74                  &valid_data_frame) != 0) printf("Data Download Error");
75
76              //Push data in the circular buffer
77              valid_data_enqueued = 0;
78              Utility_ENQUEUE_DATA_IN_DOWNLOAD_BUFFER(BufferDownloadHandler,
79                  data_frame,
80                  valid_data_frame,
81                  &valid_data_enqueued);
82
83              //Pull data from circular buffer and recostruct events
84              if (CPACK_CP_0_RECONSTRUCT_DATA(BufferDownloadHandler,
85                  &decoded_packets) == 0)
86              {
87                  //There are new events in the buffer and they are
88                  //successfully decoded
89                  printf(".");
90                  // .... do staff with your data
91
92                  //Free data
```