Usage of FIFO and Custom Packet with SciCompiler

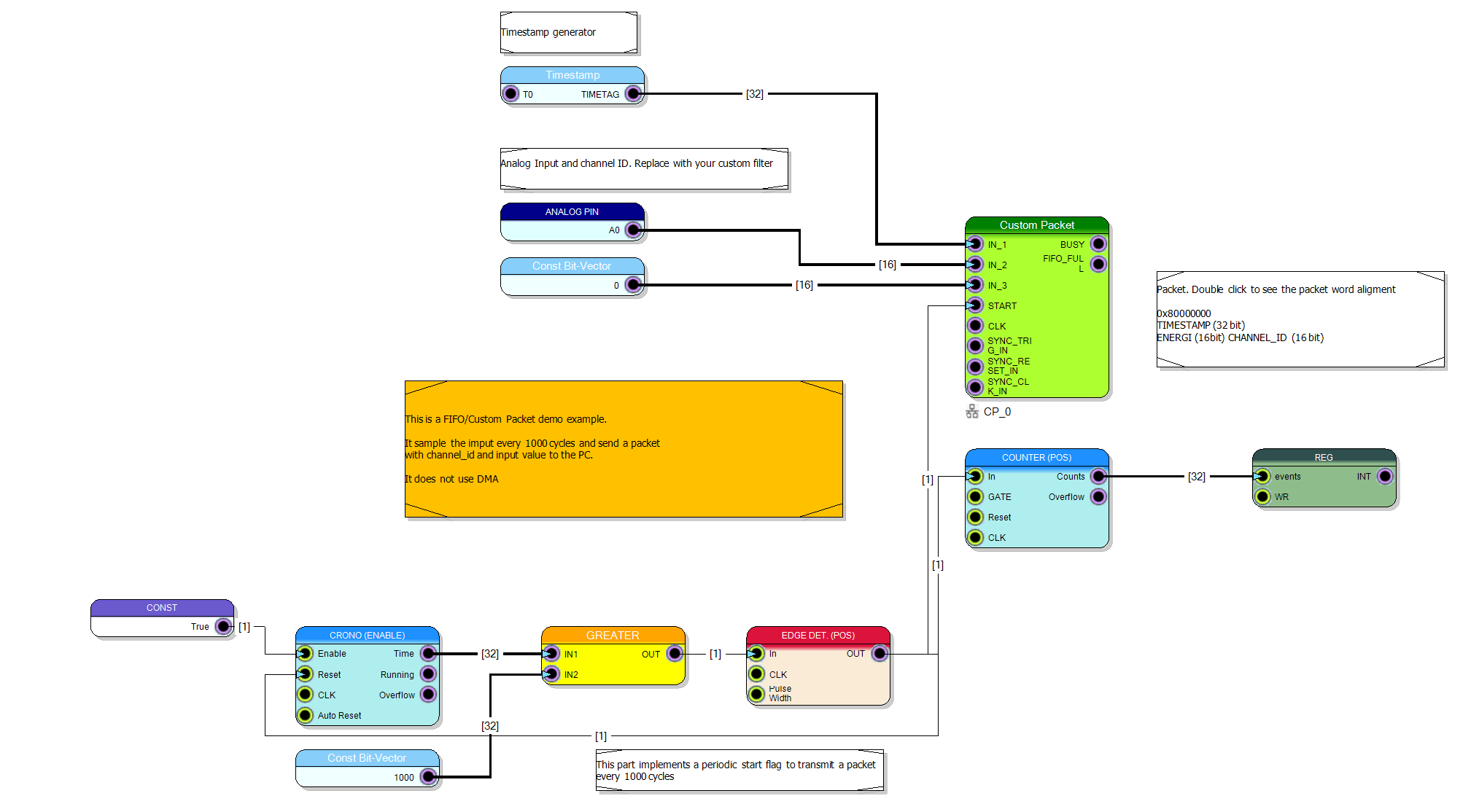
SciCompiler allows to easy implement FIFO communication to transfer formatted date from FPGA to user computer

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

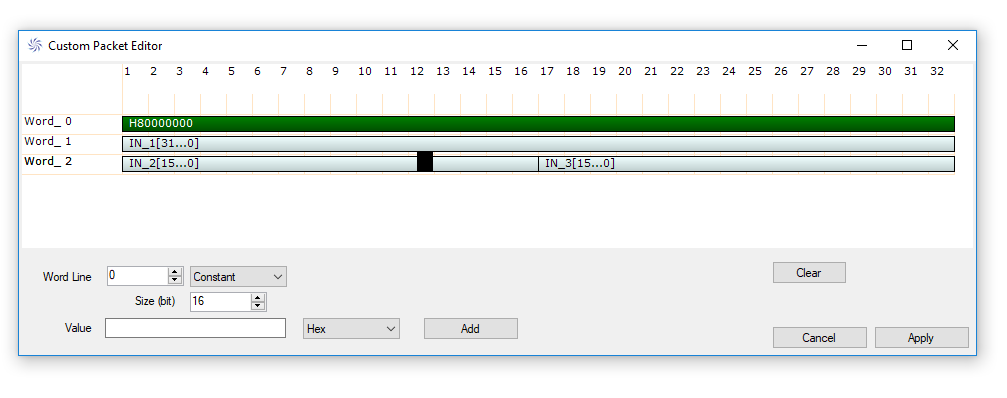
Consider this design as a reference design



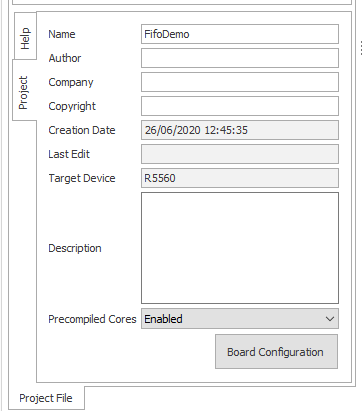
The project can be download from github: <https://github.com/NuclearInstruments/sci_custom_packet>

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

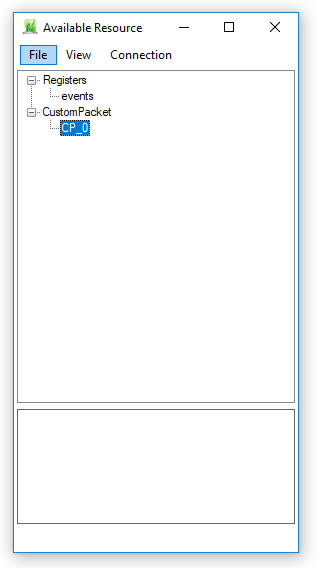
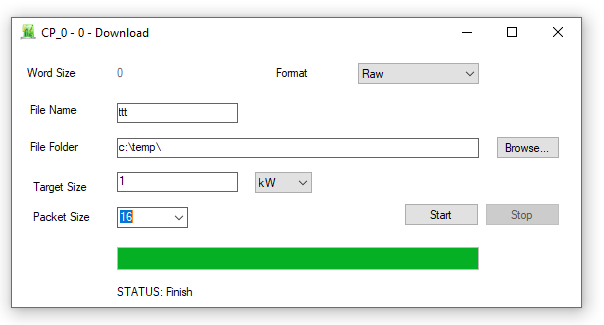


Before compile the project remember to enable the Precompiled Cores in the project properties. It speed up of a factor of 10 the compilation of the FPGA project



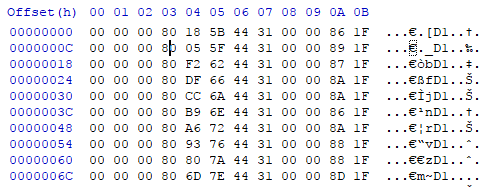
You can easy test the data transferring using the Resource Explorer tool. Compile the project, open Resource Explorer tool from SciCompiler and connect to the hardware.

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



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.

