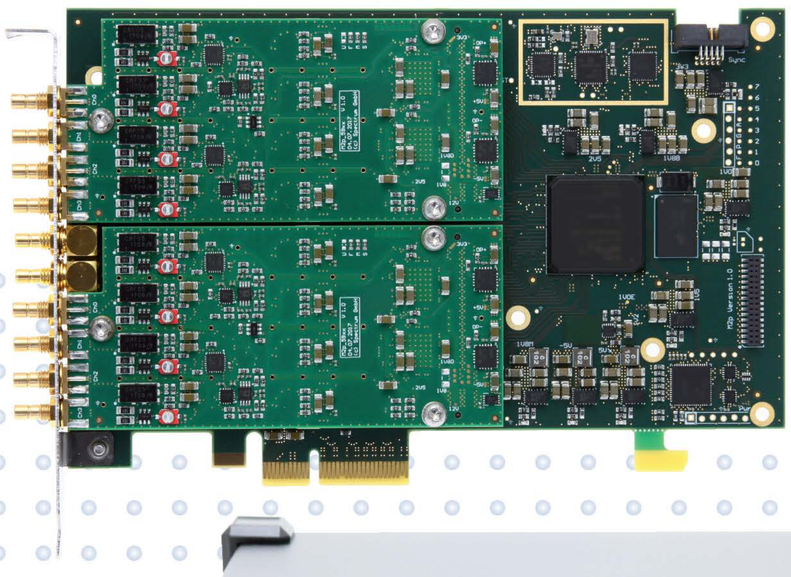
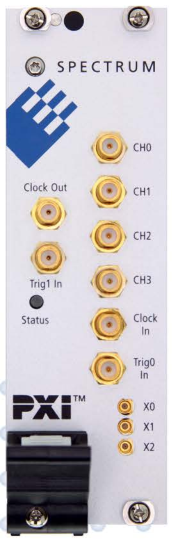
SPECTRUM
INSTRUMENTATION

Perfect fit – modular designed solutions





SCAPP Package

Spectrum CUDA Access for Parallel Programming

for M4i, M2p and M5i cards

English version

October 17, 2019

Digitizers | Transient Recorders | Arbitrary Waveform Generators | Digital Waveform Acquisition Cards

for PCI Express, PXI Express and LXI / Ethernet

SCAPP Option

Spectrum's CUDA Access for Parallel Processing

Modern GPUs (Graphic Processing Units) are designed to handle a large number of parallel operations. While a CPU offers only a few cores for parallel calculations, a GPU can offer thousands of cores. This computing capabilities can be used for calculations using the Nvidia CUDA interface. Since bus bandwidth and CPU power are often a bottleneck in calculations, CUDA Remote Direct Memory Access (RDMA) can be used to directly transfer data from/to a Spectrum Digitizer/Generator to/from a GPU card for processing, thus avoiding the transfer of raw data to the host memory and benefiting from the computational power of the GPU.

SCAPP
Spectrum's CUDA Access – Parallel Processing

The SCAPP Package

The SCAPP package opens up a CUDA-based GPU for parallel measurement data processing by giving a method of direct data transfer from Spectrum card to GPU accompanied by some easy-to-use programming examples that show the usage of the parallel processing. The examples also show how the direct data transfer is set up and handled and how the GPU measurement results are transferred back into the PC system for further analysis, storage or display.

Requirements

Motherboard with two free PCIe slots for Spectrum Digitizer and Nvidia GPU card. Please make sure that both slots fit the maximum PCIe lane and generation of the card to receive optimum performance

- Linux based operating system. RDMA on Windows is NOT supported.
- Spectrum Digitizer or Arbitrary Generator with enabled SCAPP option
- Nvidia Quadro or Tesla card with CUDA compute capability 5.0 or higher. GeForce cards are NOT supported, even if their CUDA compute capability is 5.0 or higher.
- A list of the GPUs and their CUDA compute capabilities can be found here: https://developer.nvidia.com/cuda-gpus
- Nvidia Toolkit
- Signed NDA for access to Spectrum kernel driver sources and SCAPP examples

SCAPP and Windows: Due to restrictions of the Nvidia driver it is not possible to do remote data transfer (RDMA) between a PCIe card and a Nvidia GPU using any Windows operating system. The calculation on the GPU itself can be used under Windows and Linux but from a Windows host, the data has to be transferred using one DMA transfer from Digitizer to PC and a second DMA transfer from PC to GPU.

Supported Spectrum Products

All M5i, M4i and M2p products are supported by the SCAPP option. Please note that the integrated RDMA is a function of the kernel driver and is therefore available for all products of these series of PCIe cards.

Preparation

Install CUDA Toolkit

Download the Nvidia CUDA Toolkit and install it as described here:
http://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html

Build the Nvidia kernel driver module

Download the Nvidia driver from here: http://www.nvidia.com/Download/index.aspx

Extract it on a command line using.

```
 ./NVIDIA-Linux-x86_64-<version>.run -x
```

Navigate to the output directory:

```
cd <output directory>/kernel/
```

Build the Nvidia kernel driver module:

```
make module
```

This will create the nvidia.ko kernel module and a Module.symvers file that is needed in the next step.

Compiling a CUDA RDMA enabled kernel driver module

To enable CUDA RDMA capability in the kernel driver module it needs to be linked to the proprietary Nvidia kernel driver module.

In Makefile uncomment the line that contains „NVIDIA_DRV_SRC :=“ by removing the # at the beginning of the line. Change the path to the directory of the Module.symvers file of the Nvidia kernel driver module from the previous step. Then compile the Spectrum kernel module as described in the Readme file. The Makefile will add the necessary steps to link against the Nvidia kernel module.

After the kernel module has been loaded successfully you can run dmesg to check if RDMA support has been enabled correctly. The Spectrum kernel driver module is called spcm4 and should output „Driver supports CUDA RDMA“:

```
[    2.185902] spcm4: Spectrum 64 bit driver loaded. Version 1.20 build 13926.
[    2.185903] spcm4: Driver supports CUDA RDMA.
[    2.185935] spcm4_cards 0000:02:00.0: enabling device (0000 -> 0002)
[    2.186069] spcm4: /dev/spcm0: M4i.44xx-x8
```

To check if the SCAPP option is available on the card you can read the SPC_PCIFEATURES register from your program:

Register		Value	Direction	Description
SPC_PCIFEATURES		2120	read	PCI feature register. Holds the installed features and options as a bitfield. The read value must be masked out with one of the masks below to get information about one certain feature.
	SPCM_FEAT_SCAPP	20000h	Support for the SCAPP option allowing CUDA RDMA access to/from supported graphics cards for GPU calculations (M4i and M2p)	

Programming

Setup of Spectrum Digitizer

Setup of the Spectrum Digitizer for RDMA differs only slightly from a setup without RDMA. Like in the C/C++ examples the spcm_dwSetParam_i32 () function and its friends are used to program the Spectrum Digitizer.

This manual only explains the additional information needed to run the SCAPP examples. Please refer to the standard hardware manual for a detailed description of the functions and setup registers of the driver and the hardware.

To use the direct data transfer from Digitzer to GPU the same function for data transfer setup spcm_dwDefTransfer_i64 () is used as already explained in the hardware manual:

Definition of the transfer buffer

Before any data transfer can start it is necessary to define the transfer buffer with all its details. The definition of the buffer is done with the spcm_dwDefTransfer function as explained in an earlier chapter.

```
uint32 _stdcall spcm_dwDefTransfer_i64 (// Defines the transfer buffer by using 64 bit unsigned integer values
    drv_handle  hDevice,                // handle to an already opened device
    uint32      dwBufType,              // type of the buffer to define as listed below under SPCM_BUF_XXXX
    uint32      dwDirection,            // the transfer direction as defined below
    uint32      dwNotifySize,           // number of bytes after which an event is sent (0=end of transfer)
    void*       pvDataBuffer,           // pointer to the data buffer
    uint64      qwBrdOffs,              // offset for transfer in board memory
    uint64      qwTransferLen);         // buffer length
```

This function is used to define buffers for standard sample data transfer as well as for extra data transfer for additional ABA or timestamp information. The dwBufType parameter must be the following:

SPCM_BUF_DATA	1000	Buffer is used for transfer of standard sample data

The dwDirection parameter defines the direction of the following data transfer:

SPCM_DIR_PCTOCARD	0	Transfer is done from PC memory to on-board memory of card
SPCM_DIR_CARDTOPC	1	Transfer is done from card on-board memory to PC memory.
SPCM_DIR_CARDTOGPU	2	RDMA transfer from card memory to GPU memory, SCAPP option needed, Linux only
SPCM_DIR_GPUTOCARD	3	RDMA transfer from GPU memory to card memory, SCAPP option needed, Linux only

The direction parameter SPCM_DIR_CARDTOGPU sets up the DMA engine on the Spectrum Digitizer to transfer the data from the Digitizer into the buffer on the GPU instead of transferring it to the host memory. If CARDTOGPU is used it is required to use a buffer on the GPU which can be allocated using the CUDA function cudaMalloc ().

The corresponding direction value for transfer from the GPU memory to a Spectrum AWG is SPCM_DIR_GPUTOCARD.

GPU

CUDA uses different functions to allocate memory:

- cudaMalloc () allocates a buffer on the GPU
- cudaMallocHost () allocates a buffer on the host (in the PC memory), similar to the malloc () function in standard C.

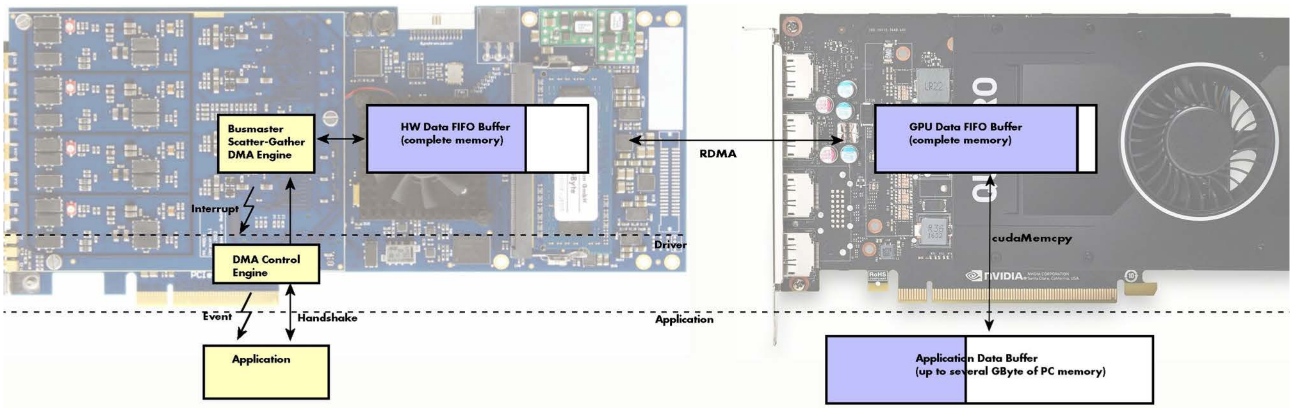
Data can be copied between a buffer on the GPU and a buffer on the host using the cudaMemcpy () function. This function uses a interface similar to the standard C memcpy () function, but with an additional parameter that defines the direction of the data transfer.

The CU_POINTER_ATTRIBUTE_SYNC_MEMOPS flag needs to be enabled if a GPU buffer should be used for RDMA:

```
unsigned int dwFlag = 1;
cuPointerSetAttribute (&dwFlag, CU_POINTER_ATTRIBUTE_SYNC_MEMOPS, (CUdeviceptr)pvDMABuffer_gpu);
```

Spectrum's convenience function pvGetRDMABuffer () which is part of the RDMA examples checks if the requested CUDA device matches the necessary requirements. Then it allocates a buffer on the GPU, sets it up for RDMA transfer and returns it to the user program.

Data flow and Control flow


Busmaster Scatter-Gather DMA Engine
HW Data FIFO Buffer (complete memory)
RDMA
GPU Data FIFO Buffer (complete memory)
Interrupt
DMA Control Engine
Driver
cudaMemcpy
Event
Handshake
Application
Application
Application Data Buffer (up to several GByte of PC memory)

Buffer handshaking for DMA transfer from Spectrum Digitizer to GPU (or vice versa) using RDMA uses the same registers as the transfer from Digitizer to host (or host to Generator) as described in the „Buffer handling“ section of the hardware manual.

The user program will be notified by an interrupt after a notify size of bytes or more are available in the GPU memory. The program can then check the precise amount of data using the SPC_DATA_AVAIL_USER_LEN register, and the position of the data inside the buffer using SPC_DATA_AVAIL_USER_POS. After the data has been processed on the GPU and the result copied to the host the user program will use SPC_DATA_AVAIL_CARD_LEN to mark the memory as free again.

Examples

There are several examples provided by Spectrum. Generally two different categories of examples are available:

- Examples which names start with the prefix „rdma_“ require RDMA support and are therefore currently only available on Linux
- Examples which names start with the prefix „cuda_“ do not use RDMA and are therefore usable Linux and also on Windows

Usefull Links

- Nvidia Toolkit: https://developer.nvidia.com/cuda-toolkit
- Nvidia CUDA Toolkit Documentation: http://docs.nvidia.com/cuda/index.html
- CUDA Runtime API: http://docs.nvidia.com/cuda/cuda-runtime-api/index.html
- cuFFT: http://docs.nvidia.com/cuda/cufft/index.html