INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2021, VOL. 67, NO. 4, PP. 565-570
Manuscript received August 27, 2021; revised November, 2021 DOI: 10.24425/ijet.2021.137847

SoPC-based DMA for PCI Express DAQ Cards

Krzysztof Mroczek

Abstract—**This paper presents low-cost, configurable PCI Express (PCIe) direct memory access (DMA) interface for implementation on Intel Cyclone V FPGAs. The DMA engine was designed to support DAQ tasks including pre-triggering acquisition for transient analysis and multichannel transmission. Performance of the interface has been evaluated on Terasic OVSK board (PCIe Gen2 x4). Target configuration of this interface is based on the Avalon-MM Hard IP for Cyclone V PCIe core and Jungo WinDriver x64 for Windows. A sample speed of 1200 MB/s has been reported for DMA writes to PCIe memory.**

***Keywords*—DAQ, DMA, SoC, PCI Express, FPGA, transient analysis**

I. Introduction

PCI Express is long-standing computer bus offering multi Gb/s transmission speed. It was introduced by Intel as a third generation computer bus. PCIe uses serial point-to-point connection between devices. PCIe link consists of two ports, one port per a device, and their interconnections called lanes. A lane is two differential signal pairs, each of the pairs is used for transmission in one direction. In PC motherboards, PCIe rev. 2.x and 3.x are commonly implemented with the number of lanes from 1 (x1) to 32 (x32). Specification [1] defines three layers: transaction layer (TL), data link layer (DL) and physical layer (PL). PCIe uses packets to carry data between components. Typically, user application operates on transaction layer packets (TLPs). Host memory is directly available for DMA access initiated by a peripheral device. DMA writes are performed by posting memory write packets (MWP). Memory read is a split transaction. Requester device sends memory read packet (MRP) addressed to a completer device. After the completer has received and decoded TLP, it prepares data and sends them to the requester in one or multiple completion TLPs. PCIe 2.x link operates with a speed of 5GT/s per lane and uses the 8b/10b transmission code. PCIe 3.x link operates with a speed of 8GT/s per lane and uses the 128b/130b code. For PCIe Gen2, maximum throughput is 0.5 GB/s per lane.

In this paper low-cost PCIe DMA interface for FPGA-based DAQ cards is proposed. The described DMA engine supports pre-trigger acquisition for signal transient analysis and multichannel transmission. In articles e.g. [11-13], PCIe designs are reported based on custom DMA projects. This paper examines different approach, the use of available free IP cores for designing a functional PCIe DMA interface for DAQ systems. The proposed solution is based on Intel SoPC resources. In section II, architecture and components of the interface are described. Section III introduces projects used for prototyping this interface. Section IV discusses achieved performance.

Described DMA interface is a building block of a DAQ card. In our previous works, two general purpose DAQ cards were used for testing of methods applied in the IRTM NIALM system [2-3]. A new FPGA-based DAQ hardware will provide more flexibility and improve the system.

II. Components and Architecture of DMA Interface

A. *SoPC resources*

In the project presented in this paper, description of hardware system in FPGA has been developed using the Platform Designer (PD) from Intel Quartus Prime software and free IP cores. The PD is a high-level system integration tool for SoPC (System on a Programmable Chip). PD project includes components, interfaces and interconnections in graphical representation of a system [6]. Two interfaces from Avalon family were used in datapaths. The Avalon-ST (streaming) interface is intended for unidirectional point-to-point links between a source of data (ST source) and a destination (ST Sink). ST connections can be used for implementation of address-less transmission channels, including packets and multiplexed streams. The Avalon-MM (memory mapped) allows connecting a component with master interface to one or more slaves. The master initiates address-based read or write transfer to the destination slave. Address bus width can be selected at up to 64 bits, data width at up to 1024 bytes. Avalon-MM enables single and burst transfers.

PD schema of a SoPC is created and edited with GUI tool. Components from Quartus or user-defined are added to the schema from the IP Catalog. HDL description of a SoPC is automatically generated from PD project for synthesis and simulation. Generated HDL is available to be included in Quartus project. Important elements in PD are interconnections and bridges. PD supports 1-1, 1-N, N-M master-slave connections. Interconnections are generated automatically from schema, while bridges are added by the designer. PD generates interconnections with slave-side arbitration. If multiple masters are connected to the same slave, PD inserts arbitration logic. Round-robin algorithm is the default for arbitration. Arbitration selects one master from among the requesting and grants it access to the slave. PD generates parallel master-slave paths enabling concurrent transfers.

PCIe protocol was implemented by applying *Avalon-MM Cyclone V Hard IP for PCI Express* (PCIE_AVMM) core [7]. PCIE_AVMM is compatible with Cyclone V chips comprising a hardened PCIe protocol stack. PCIE_AVMM supports PCIe Gen1 and Gen2 x1, x2 or x4. PCIE_AVMM connects PCIe hard

K. Mroczek is with Institute of Radioelectronics and Multimedia Technology (IRTM), Warsaw University of Technology, Poland (e-mail: krzysztof.mroczek@pw.edu.pl).

block (PCIE_ST) to the application layer via the Avalon-MM bridge (Fig. 1). The PCIE_ST implements the PCIe protocol stack compliant with PCI Express Base Specification v. 2.1/3.0. It includes Data Link Layer, Physical Layer, and connections to LVDS transceivers. Using this IP can decrease overall development time. User application in FPGA can handle data transmission on Avalon-MM interfaces. Building, decoding of TLPs, and other protocol activities are implemented in the bridge. MWPs and MRPs initiated by host software are translated in the bridge to the Avalon-MM transfers. These transfers are performed on the *Rxm_BARn* master interfaces. Up to 6 *Rxm_BARn* can be configured. When PCIE_AVMM receives valid MWP, it initiates a write transfer on the activated *Rxm_BARn* interface. Receiving of MRP initiates read transfer, followed by sending back data in completion TLP. The *Txs* interface allows implementation of bus master DMA. The *Txs* is bursting Avalon-MM slave interface that translates transfers initiated by a peripheral device to MWP or MRP packets. Address translation from Avalon-MM to PCIe depends on the Avalon-MM address width setting, configurable to 32 or 64 bits. For 64-bit, no address translation is performed. In this case, the Avalon addresses are directly mapped to PCIe memory.

B. *PCI Express DMA interface*

The architecture of PCIe DMA interface is shown in Fig. 1. Logical connections between Avalon-MM master and slave interfaces are marked with dots. The PCIE_AVMM controller is configured for PCIe Gen2 x4 and 64-bit address space. The differential 100 MHz REFCLK signal from PCIe connector is the main clock source. Internal clocks are generated from this clock. PCIE_AVMM external interfaces and the DDR3 controller are synchronized to the 125 MHz *coreclkout* (clk1). DMA components are synchronized to the 100 MHz clock (clk2). Software access to registers in components is performed on the *Rxm_BAR2* master interface, connected to the B1 bridge. Avalon-MM master ports of DMA modules are connected to the *Txs* and the DDR3_C slaves via the B2 - B4 bridges. The bridges are implemented by applying *Avalon-MM Clock Crossing Bridge* component from PD. Applying these bridges helps to meet timing requirements for implementations in Cyclone V FPGAs. IRQ requests from system components are connected to the *RxmIrq* input of PCIE_AVMM. The PCIE_AVMM internal registers are accessed by the CRA interface.
DMA memory space is divided into two regions, defined in PD:
- PCIe memory; addresses up to 64 GB,
- Board memory; addresses above 64 GB.

Memory space can be extended by adding next regions of FPGA resources.

The Stream DMA subsystem consists of dual channel DMA. The DMA writes data to PCIe or board memories. Each of DMA channels is handled by its own mSGDMA module, sourced from the PD IP Catalog [5]. Features of the mSGDMA IP are outlined in II.C.
The DAQ_C module splits samples into the DMA channels. Functions of this module are described in II.D. Samples from a DAQ hardware are written to one of the DMA channels via two Avalon-ST interfaces, according to the selected buffering mode. To support a signal analysis in time window before and after a trigger event, pre-trigger buffering mode was developed. The mSGDMAs can have optional master interfaces for descriptors access. Depending on configuration, DMA descriptors are written to the controller FIFO by software or stored in memory (PCIe or board) as a linked list of records.

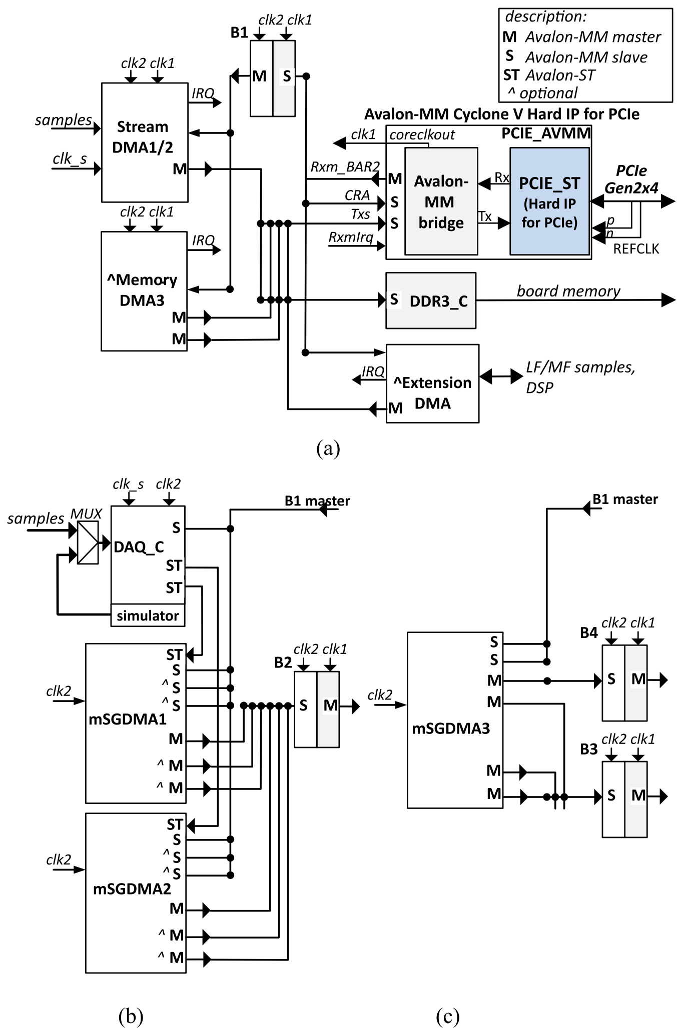


Fig. 1 Architecture of PCIe DMA interface (a). Stream DMA1/2 (b). Memory DMA3 (c)

The Memory DMA subsystem has one mSGDMA3 controller, configured with two Avalon-MM ports (one for write, one for read) and prefetcher extension. This DMA allows moving data between memories located on the board and on PCIe. Write and read master ports of the mSGDMA3 are connected to the B3 and B4 bridges. Additional DMA components can be added to extend the functionality for multifrequency or multichannel data acquisition.

PD is a robust tool for designing high throughput connections between multiple DMA masters and slaves. When the DMA masters perform transfers targeted to both *Txs* and DDR3, data are transmitted simultaneously in separate paths. Thus, throughput is maximized.

C. *The mSGDMA*

The mSGDMA is configurable DMA controller enabling stream to memory, memory to stream and memory to memory transfers. The mSGDMA 1 and 2 modules are configured with the Avalon-ST input interface. The controller has one data mover module and the dispatcher module (Fig. 2(a)). The dispatcher receives and decodes descriptors, which are programming instructions for the data mover. Each descriptor describes a single transfer, defining the source and destination addresses, data length, options. Descriptors can be queued in advance in the dispatcher FIFO. The data mover fetches data from the internal buffer, written prior by the Avalon-ST port, and moves them to the destination memory in burst transfers on the Avalon-MM interface. Many parameters are available for static configuration. Packet support and extended feature set are the most important. Extended feature set allows 64-bit addressing, prefetcher extension, and events signaling.

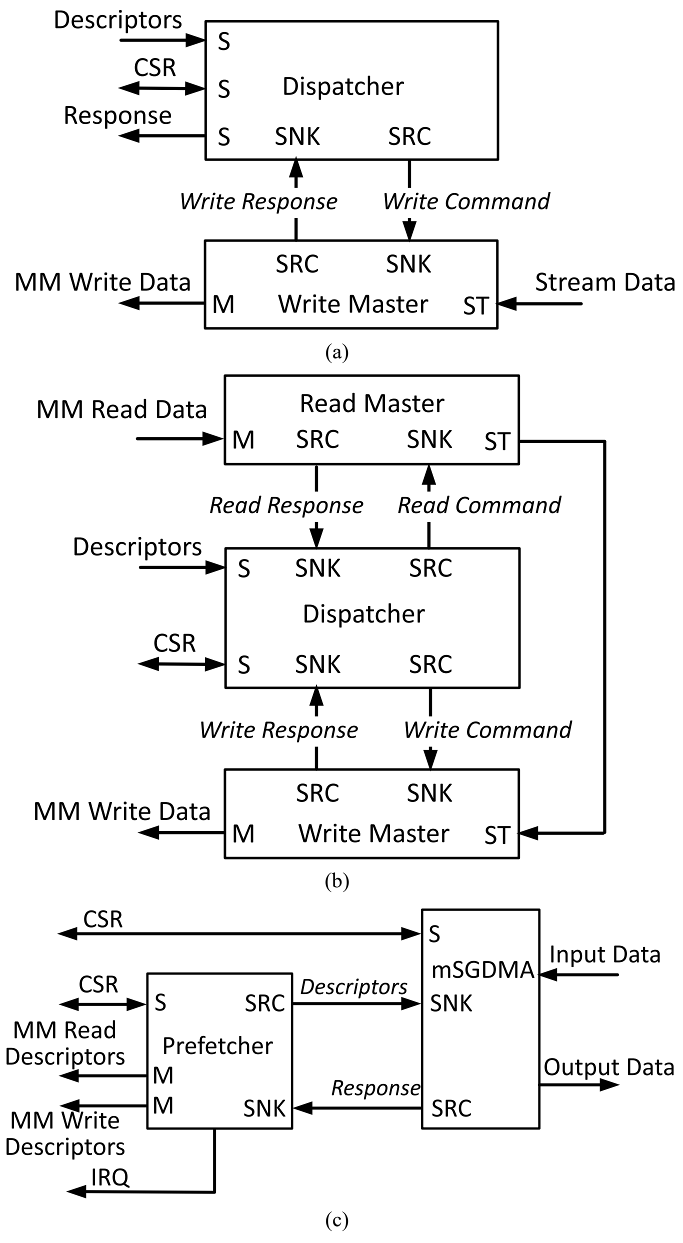


Fig.2. (a) DMA_EXT configuration; (b) with Avalon-MM input; (c) DMA_EXTPREF configuration. Source: [5].

The DAQ_C module writes samples to the mSGDMA internal FIFO in packets. A packet begins with *SOP* (Start of Packet) signal asserted and ends with *EOP* (End of Packet) signal asserted. Packet mode enables termination of DMA transfer by a hardware signal. If *EOP* is asserted during writing a block of data, this block will be the last one in the current descriptor processed. After a descriptor is finished, the transfer status (number data of bytes, events) is written to the response FIFO.

Two configurations of the mSGDMA have been applied in described project, outlined below.

DMA_EXT: without prefetcher module (*Enable Pre-Fetching module = Disable*).

Descriptors are written by software to the dispatcher FIFO. Status of transfer is written to the response FIFO. The response slave port is used for reading this FIFO. Software must read status for every completed descriptor. IRQs can be generated according to unmasked conditions.

DMA_EXTPREF: with prefetcher module (*Enable Pre-Fetching module = Enable*).

This configuration allows transfers to non-contiguous memory buffers, typical in Windows and Linux. Software prepares linked list of descriptors located in memory. Additional prefetcher module fetches descriptors from the list. After fetching, the prefetcher writes a descriptor to the dispatcher FIFO. After transfer is finished, the dispatcher writes response data. Next, the prefetcher reads the response FIFO and performs descriptor write-back. The prefetcher has two Avalon-MM master ports, one for descriptors fetches, second for descriptor writes. After descriptor update, IRQ can be generated according to unmasked conditions. The Owned by Hardware (*ObH*) bit in the control field of a descriptor is used for synchronization between the controller and software. If the *ObH* is set to 1, the controller has access to the descriptor. Software can update the descriptor after the controller has changed the *ObH* to 0.

One of essential functions of a DAQ hardware is support for continuous acquisition. In this mode, DMA most often works with a ring buffer. Ring buffer can be located on the DMA side or in software memory. The mSGDMA can be used in continuous mode. Two approaches can be applied in this DMA engine, described as follows. First is queuing of descriptors in controllers to form ring buffer. In the case of the DMA_EXT, software writes a descriptor to the dispatcher FIFO after completing of descriptor processing. With the prefetcher configuration, equivalent function can be achieved for descriptor list. The second is programming of the mSGDMA *Park Write Mode*. The park mode was used in the pre-trigger transmission, what is outlined in section III.

D. *The DAQ adapter*

The DAQ_C is a custom module written in Verilog. The main task of the DAQ_C is splitting incoming samples into one of two outputs, according to programmed mode. Samples from data source are passed to input port and written to the dual-clock FIFO (FIFOIN). Input side is synchronized to the sample clock, output to the *clk2* system clock. Data bus is extended to 132 bits. 128-bit sample block is written to the FIFOIN together with 5 bits of tags. 4 tags are used to signal events, synchronous in time with samples. The meaning of the tags is as follows:

EvPretrig – trigger for signaling change on data from pre- to post- trigger.
EvError – error.
EvApp, *EvAppEOP* – application defined events without and with generating of EOP request.
Additional *EvOverrun* tag signals that samples have been overwritten due to lack of space in the FIFOIN.

Data are read from the FIFOIN in 133-bit blocks, containing samples and tags. Samples are passed to the Avalon-ST ports and written to selected DMA channel. Tags are used as input to control logic. The mSGDMA buffers incoming samples before generating of Avalon-MM bursts. Therefore, the DAQ_C writes samples without forming bursts. The DMA state machine is shown in Fig. 3 and described below. Every DMA channel has 32-bit sample counter (wCnt1, wCnt2) and buffer size register (rDMA1, rDMA2). Sample counters are initialized by writing buffer size of the performed DMA transfer(s), described by one or more descriptors. Data are written to DMA1 if the FSM is in the SD1 state, and to DMA2 if the FSM is in the SD2 state.

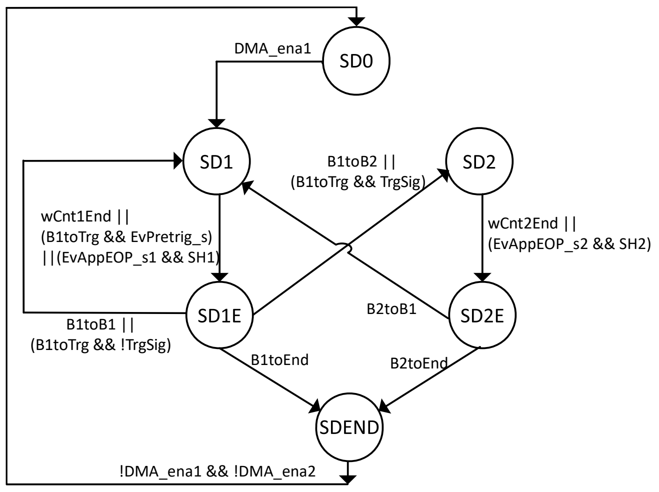


Fig. 3. Stream DMA state machine

The DAQ_C supports three acquisition modes.
1. Single DMA1 (B1toB2 = 0; B1toTrg = 0).
After unblocking by software, the FSM enters the SD1 state. Data are written to the mSGDMA1. The wCnt1 counts bytes written to the DMA1 channel. After the wCnt1 has expired, i.e. after the DMA1 phase is completed, the FSM enters the SD1E state. When the cyclic bit is set (B1toB1 = true), the wCnt1 is reloaded to initial value and the sequence is started again. Otherwise, the process is finished.

2. Dual DMA1/2 (B1toB2 = 1; B1toTrg = 0).
First, data are written to the DMA1. After the DMA1 phase is completed, the FSM enters the SD1E and SD2 states. Data are written to the mSGDMA2. The transmission is finished after the wCnt2 has expired. According to the cyclic bit, the DMA1 – DMA2 sequence is repeated by entering the SD1 state or the process is finished.

3. Pre-trigger dual-channel (B1toB2 = 1; B1toTrg = 1).
Pre-trigger samples are written to the mSGDMA1. The FSM cyclically enters the SD1 - SD1E states. When data block marked with the *EvPretrig* tag has been written on the DMA1 Avalon-ST port, the FSM enters the SD1E state and then goes to the SD2 state. Samples after a trigger are written to the mSGDMA2. The last value of the wCnt1 and the number of finished buffers are stored in events registers. After the wCnt2 has expired, the sequence is started again or the process is finished.
The SH1 and SH2 bits control the termination of current DMA phase triggered by the *EvAppEOP*. To prevent overwriting of samples in continuous modes, software should unblock subsequent cycles by write to control register (it isn't shown in conditions in Fig. 3 for the simplicity of the picture).

The DAQ_C contains status registers for events signaled by the tags and the *EvOverrun*. When an event is signaled during writing of a sample block, current values of the sample register and the number of finished DMA buffers are stored in the status registers.

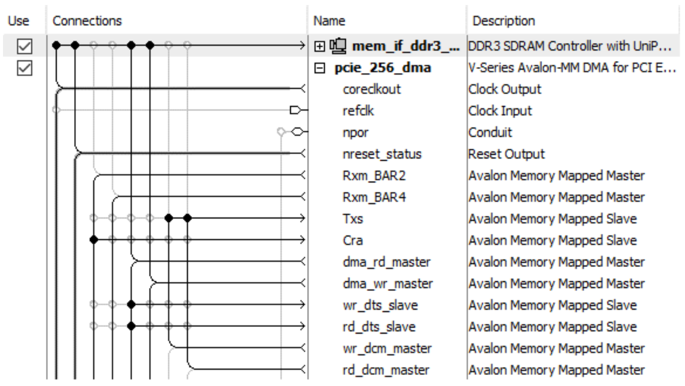
The DAQ_C implements data and events validation. The validation acknowledges that data read in consumer side have been prior written by DMA. It can be carried out in two ways. The first method is including events signals and EOP in data bus of the ST interfaces. 8 bits in 128-bit data block are reserved for events. To determine address of the marked block, the event register is read. Next, the events byte is read from calculated address and data are validated. To preserve throughput, a known data pattern can be inserted synchronously with an event. The second method is connecting the events signals to the Avalon-ST error port of the mSGDMA. When a DMA transfer is completed, if events were set during the transfer, the descriptor status field is updated during descriptor write-back. As it is shown in Fig. 1, the descriptor master ports and DMA data port are connected to the B2 bridge. Descriptor write-back is initiated after a transfer on data interface is finished. The Avalon transfers are queued in the B2 and performed at the master side in request order. To validate data, target side reads event registers and status field of descriptor.

III. PROTOTYPING ON THE BOARD

The PCIe DMA interface described above has been developed using the PD tool. Programming instructions on register level were simulated in ModelSim simulator. Extensive tests were performed on board to evaluate the DMA engine. Design has been prototyped in two projects implemented on Terasic OVSK board (PCIe Gen2 x4) [10].

The first test was based on a PCIE_DDR3 example for this board. In this project, different IP core is used as PCIe interface module. As it is shown in Fig. 4, the *pcie_256_dma* module is an instance of the *V-series Avalon-MM DMA PCIe* core (PCIE_AVMMDMA) [8]. The PCIE_AVMMDMA is configured with an internal DMA controller. This internal DMA moves data between Avalon-MM and PCIe in transfers programmed by descriptors. DMA transfers are initiated on the Avalon-MM master ports, what is different compared to the PCIE_AVMM *Txs* slave connection. The PCIE_AVMMDMA fetches data on the *dma_wr_master* interface and writes them to PCIe destination. The stream DMA1/2 subsystem (Fig. 1) was added to the base project. The mSGDMA1 and 2 modules were configured in the DMA_EXT mode. The DMA moves samples from data sources to DDR3 memory.

Test software was developed for Windows 10 x64. It uses Altera PCI API driver and C library, available at no cost. This driver doesn't support interrupts. Software uses pooling in a thread for handling events and interacting with hardware. The first test procedure is based on descriptors queuing in advance. In this case, at the beginning, an assumed number of descriptors is queued in both mSGDMA controllers. Next, acquisition mode is set in the DAQ_C and data flow with programmed speed is started. The operation thread is periodically woken. When it is running, it reads events from hardware and response queues of the mSGDMAs. If any DMA transfer is finished, the thread calls the *PCIE_DmaRead* API to copy data from DDR3 to the program buffer. After data copying, finished transfer descriptor is written to the dispatcher FIFO in form of ring queue. The pre-trigger mode was evaluated additionally in the second configuration. For this case, to perform continuous DMA writes, the mSGDMA1 is programmed in the *Park Write* mode. After trigger has occurred, the mSGDMA1 is reprogrammed to next trigger sequence. During post-trigger state, after processing a descriptor, the finished descriptor is written to the dispatcher FIFO of the mSGDMA2.

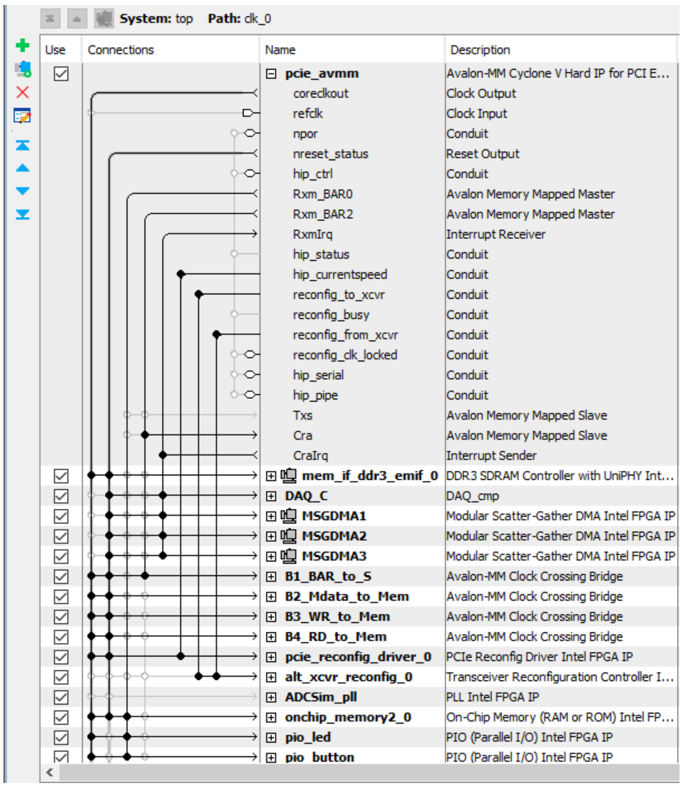


Use	Connections	Name	Description
☑		⊞ mem_if_ddr3_...	DDR3 SDRAM Controller with UniP...
☑		⊟ pcie_256_dma	V-Series Avalon-MM DMA for PCI E...
		coreclkout	Clock Output
		refclk	Clock Input
		npor	Conduit
		nreset_status	Reset Output
		Rxm_BAR2	Avalon Memory Mapped Master
		Rxm_BAR4	Avalon Memory Mapped Master
		Txs	Avalon Memory Mapped Slave
		Cra	Avalon Memory Mapped Slave
		dma_rd_master	Avalon Memory Mapped Master
		dma_wr_master	Avalon Memory Mapped Master
		wr_dts_slave	Avalon Memory Mapped Slave
		rd_dts_slave	Avalon Memory Mapped Slave
		wr_dcm_master	Avalon Memory Mapped Master
		rd_dcm_master	Avalon Memory Mapped Master

Fig. 4. Interfaces of PCIE_AVMMDMA

Maximum estimated data rate was in range 770 - 950 MB/s. The evaluated throughput is based on overrun criterion. Lack of interrupts and memory-oriented DMA are limitations of this design.

The second project is implementation of the architecture shown in Fig. 1. Its structure is shown in Fig. 5. The DMA stream subsystem is connected to the PCIE_AVMM and the DDR3 controllers. The software layer for a hardware is based on Jungo WinDriver x64 [9]. Test software was written in C to evaluate pre-trigger and continuous modes. Acquisition mode with programmed sample rate was set in the DAQ_C. The DMA engine writes data to the destination memory in PCIe or DDR3. WinDriver supports continuous and S-G DMA buffers. For both memory options, the DMA_EXT and DMA_EXTPREF configurations of the mSGDMA have been applied accordingly. Interrupts from the system components were used as sources for MSI generation. The interrupts were handled in user-mode ISR registered by WinDriver. No data corruption errors were reported during the tests. Overrun criterion was applied for estimation of maximum throughputs. Maximum data rate set in samples simulator, which didn't cause noticeable overrun errors, was noted as a boundary value. The reported throughput results are shown in table 1 in rows 1 - 4. Results for DMA3 in row 5 are based on Windows *QueryPerformanceCounter* function. Data rate of about 1200 MB/s has been reported for DMA writes to PCIe memory and 1593 MB/s for the writes to DDR3.



System: top Path: clk_0

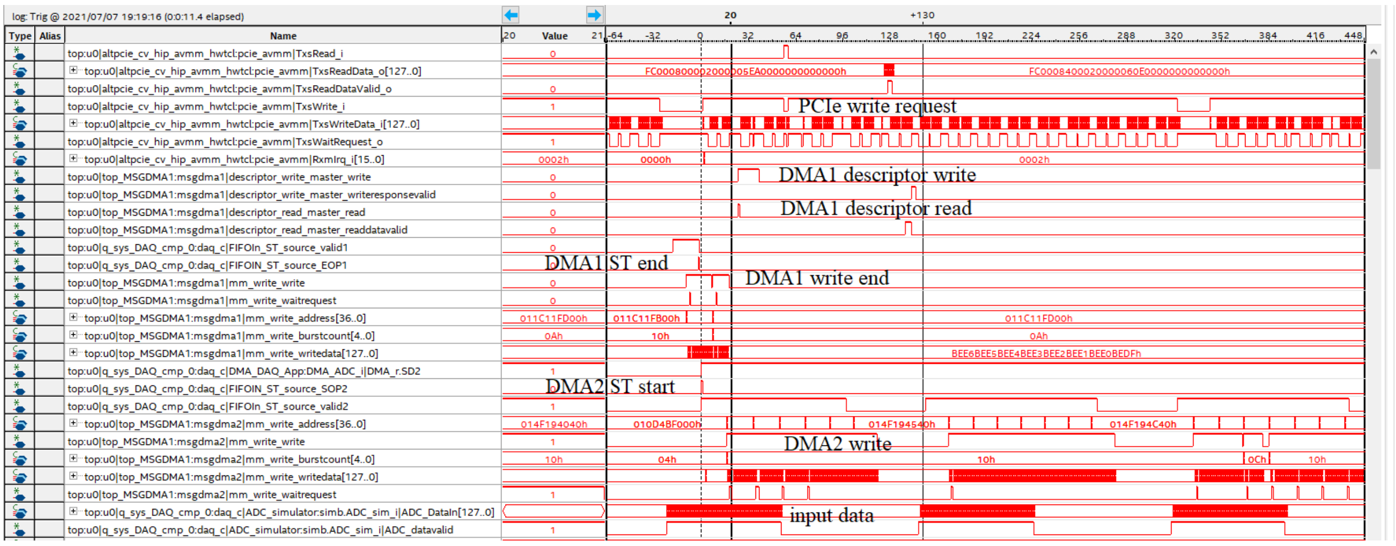
Use	Connections	Name	Description
☑		⊟ pcie_avmm	Avalon-MM Cyclone V Hard IP for PCI E...
		coreclkout	Clock Output
		refclk	Clock Input
		npor	Conduit
		nreset_status	Reset Output
		hip_ctrl	Conduit
		Rxm_BAR0	Avalon Memory Mapped Master
		Rxm_BAR2	Avalon Memory Mapped Master
		RxmIrq	Interrupt Receiver
		hip_status	Conduit
		hip_currentspeed	Conduit
		reconfig_to_xcvr	Conduit
		reconfig_busy	Conduit
		reconfig_from_xcvr	Conduit
		reconfig_clk_locked	Conduit
		hip_serial	Conduit
		hip_pipe	Conduit
		Txs	Avalon Memory Mapped Slave
		Cra	Avalon Memory Mapped Slave
		CraIrq	Interrupt Sender
☑		⊞ mem_if_ddr3_emif_0	DDR3 SDRAM Controller with UniPHY Int...
☑		⊞ DAQ_C	DAQ_cmp
☑		⊞ MSGDMA1	Modular Scatter-Gather DMA Intel FPGA IP
☑		⊞ MSGDMA2	Modular Scatter-Gather DMA Intel FPGA IP
☑		⊞ MSGDMA3	Modular Scatter-Gather DMA Intel FPGA IP
☑		⊞ B1_BAR_to_S	Avalon-MM Clock Crossing Bridge
☑		⊞ B2_Mdata_to_Mem	Avalon-MM Clock Crossing Bridge
☑		⊞ B3_WR_to_Mem	Avalon-MM Clock Crossing Bridge
☑		⊞ B4_RD_to_Mem	Avalon-MM Clock Crossing Bridge
☑		⊞ pcie_reconfig_driver_0	PCIe Reconfig Driver Intel FPGA IP
☑		⊞ alt_xcvr_reconfig_0	Transceiver Reconfiguration Controller I...
☑		⊞ ADCSim_pll	PLL Intel FPGA IP
☑		⊞ onchip_memory2_0	On-Chip Memory (RAM or ROM) Intel FP...
☑		⊞ pio_led	PIO (Parallel I/O) Intel FPGA IP
☑		⊞ pio_button	PIO (Parallel I/O) Intel FPGA IP

Fig. 5. Interface with PCIE_AVMM controller

Continuous transmission to DDR3 with data copying to PCIe showed a speed above 800 MB/s. The performance of scatter-gather DMA with buffers allocated by C *malloc()* was similar to the configuration with a pre-allocated continuous memory. An example of waveforms captured by SignalTap logic analyzer is shown in Fig. 6. It shows a change of state from the pre- to post-trigger state for a data rate of 1087 MB/s.

TABLE I
REPORTED THROUGHPUT

	Test conditions DAQ_C FIFO size: 65536B; burst: 256B; TLP: 128B	Throughput [MB/s]
1	DMA write to PCIe	1000 – 1200
2	DMA write to DDR3	1593
3	DMA write to PCIe + DMA3 PCIe - DDR3	610
4	DMA write to DDR3 + DMA3 read DDR3 - PCIe	> 800
5	DMA3 PCIe read – DDR3 write DMA3 DDR3 read – PCIe write	1100 – 1250 1100 – 1270


log: Trig @ 2021/07/07 19:19:16 (0:0:11.4 elapsed)

Type	Alias	Name	Value
		top:u0|altpcie_cv_hip_avmm_hwtcl:pcie_avmm|TxsRead_i	0
		top:u0|altpcie_cv_hip_avmm_hwtcl:pcie_avmm|TxsReadData_o[127..0]	
		top:u0|altpcie_cv_hip_avmm_hwtcl:pcie_avmm|TxsReadDataValid_o	0
		top:u0|altpcie_cv_hip_avmm_hwtcl:pcie_avmm|TxsWrite_i	1
		top:u0|altpcie_cv_hip_avmm_hwtcl:pcie_avmm|TxsWriteData_i[127..0]	
		top:u0|altpcie_cv_hip_avmm_hwtcl:pcie_avmm|TxsWaitRequest_o	1
		top:u0|altpcie_cv_hip_avmm_hwtcl:pcie_avmm|RxmIrq_i[15..0]	0002h
		top:u0|top_MSGDMA1:msgdma1|descriptor_write_master_write	0
		top:u0|top_MSGDMA1:msgdma1|descriptor_write_master_writeresponsevalid	0
		top:u0|top_MSGDMA1:msgdma1|descriptor_read_master_read	0
		top:u0|top_MSGDMA1:msgdma1|descriptor_read_master_readdatavalid	0
		top:u0|q_sys_DAQ_cmp_0:daq_c|FIFOIn_ST_source_valid1	0
		top:u0|q_sys_DAQ_cmp_0:daq_c|FIFOIN_ST_source_EOP1	0
		top:u0|top_MSGDMA1:msgdma1|mm_write_write	0
		top:u0|top_MSGDMA1:msgdma1|mm_write_waitrequest	0
		top:u0|top_MSGDMA1:msgdma1|mm_write_address[36..0]	011C11FD00h
		top:u0|top_MSGDMA1:msgdma1|mm_write_burstcount[4..0]	0Ah
		top:u0|top_MSGDMA1:msgdma1|mm_write_writedata[127..0]	
		top:u0|q_sys_DAQ_cmp_0:daq_c|DMA_DAQ_App:DMA_ADC_i|DMA_r.SD2	1
		top:u0|q_sys_DAQ_cmp_0:daq_c|FIFOIN_ST_source_SOP2	0
		top:u0|q_sys_DAQ_cmp_0:daq_c|FIFOIn_ST_source_valid2	1
		top:u0|top_MSGDMA1:msgdma2|mm_write_address[36..0]	014F194040h
		top:u0|top_MSGDMA1:msgdma2|mm_write_write	1
		top:u0|top_MSGDMA1:msgdma2|mm_write_burstcount[4..0]	10h
		top:u0|top_MSGDMA1:msgdma2|mm_write_writedata[127..0]	
		top:u0|top_MSGDMA1:msgdma2|mm_write_waitrequest	1
		top:u0|q_sys_DAQ_cmp_0:daq_c|ADC_simulator:simb.ADC_sim_i|ADC_DataIn[127..0]	
		top:u0|q_sys_DAQ_cmp_0:daq_c|ADC_simulator:simb.ADC_sim_i|ADC_datavalid	1

Fig. 6. S-G DMA to PCIe memory with data rate of 1087 MB/s. Waveforms captured by SignalTap.

IV. THE PERFORMANCE

The measured speed for PCIe transfers is smaller than 1600 MB/s. It is a boundary value for this DMA engine, decreased due to application of the clock-crossing bridges. Theoretical maximum data throughput for PCIe Gen2 x4, TLP = 128 – 256B, is 1684 – 1828 MB/s [11]. Factors decreasing the performance are related to both the host system and the DMA engine. It was noticeable, that increasing the number of processes running in OS may introduce overruns.

Additional constrains should be assumed for a continuous transmission. The obligatory condition is that processing time in software must be lower than the time of filing the DMA buffers. From software perspective, queuing/linking of descriptors in advance can be programmed. For the pre-trigger mode, the DMA1 channel is reprogrammed to next buffer as a part of IRQ processing. Programming of the DMA2 channel is adding new descriptors on-line. Additionally, each of the channels must be unlocked for the next trigger sequence. To avoid overruns, the sizes of DMA buffers should be adjusted according to sample rate. The S-G buffers with a large number of physical pages are the next issue for analysis.

The full design with the PCI_AVMM and DMA1-3 configured in the DMA_EXTPREF utilizes 18% of ALMs and 17% of memory on 5CGTFD9D5F27C7 FPGA. The DMA components (mSGDMAs, DAQ_C and bridges) use only 3% of ALMs and 8% of memory. More than a half of used ALMs are generated by the PD interconnections. Design containing only the stream DMA in the DMA_EXT, without the mSGDMA3, decreases ALM demand by 7%. Decreasing of resource usage can be one of further optimizations.

V. SUMMARY AND CONCLUSION

In conclusion, using the SoPC resources to design a functional PCI Express DMA has yielded assumed results. This paper presents a low-cost PCIe DMA interface for FPGA-based DAQ cards. This interface supports pre-trigger, continuous, and multichannel data acquisition with sample rate above 1000 MB/s. The described DMA engine was successfully prototyped on Terasic OVSK board in two projects with the PCIE_AVMMDMA and PCIE_AVMM controllers. Further enhancements in hardware are planned to improve this DMA engine. WinDriver toolkit enables writing API library with a reduced effort for the kernel-mode programming. Due to high data rate, development of API library will require more analysis of software environment to obtain intendent features.

REFERENCES

[1] PCI Express Base Specification, rev. 3.0, PCI-SIG, Nov. 2010

[2] A. Wójcik, R. Łukaszewski, R. Kowalik, W. Winiecki, “Nonintrusive Appliance Load Monitoring: An Overview, Laboratory Test Results and Research Directions”, *Sensors*, 2019, 19, 3621

[3] A. Wójcik, P. Bilski, R. Łukaszewski, K. Dowalla, R. Kowalik, “Identification of the State of Electrical Appliances with the Use of a Pulse Signal Generator”, *Energies*, 2021, 14, 673.

[4] K. N. Trung, E. Dekneuvel, B. Nicolle, O. Zammit, C. N. Van, G. Jacquemod, “Using FPGA for Real Time Power Monitoring in a NIALM System”, *In Proc. 2013 IEEE International Symposium on Industrial Electronics (ISIE)*, 2013, pp. 1-6

[5] Intel Corporation, Modular Scatter-Gather DMA Core, In Embedded Peripherals IP User Guide v. 18.1

[6] Intel Corporation, Intel® Quartus® Prime Standard Edition User Guide v. 18.1, Platform Designer

[7] Intel Corporation, Cyclone® V Avalon® Memory Mapped (Avalon-MM) Interface for PCIe Solutions User Guide, UG-01110, 2020

[8] Intel Corporation,V-Series Avalon-MM DMA Interface for PCIe Solutions User Guide, UG-01154, 2016

[9] WinDriver, *https://www.jungo.com/st/products/windriver/wd_windows/*

[10] OpenVINO Stater Kit GT Edition User Manual, available on *https://www.terasic.com.tw/*

[11] L. Rota, M. Caselle, S. Chilingaryan, A. Kopmann, M. Weber, “A PCIe DMA Architecture for Multi-Gigabyte Per Second Data Transmission”, *IEEE Transactions on Nuclear Science*, vol. 62, no. 3, 2015, pp. 972 - 976

[12] A. Byszuk, J. Kołodziejski, G. Kasprowicz, K. Poźniak, W. M. Zabołotny “Implementation of PCI Express bus communication for FPGA-based data acquisition systems”, *In Proceedings of SPIE Vol. 8454*, 2015

[13] L. Boyang, “Research and Implementation of XDMA High Speed Data Transmission IP Core Based on PCI Express and FPGA”, in *2019 IEEE 1st International Conference on Civil Aviation Safety and Information Technology (ICCASIT)*, Oct. 2019, pp. 408–411