Space Tolerant CNN FPGA Deployment
Part 3: DPU Control and Interfacing

A. C. McCormick

Space Tolerant CNN FPGA Deployment, Part 3

This paper is the third part of a four-part series of white papers providing an educational overview of the issues surrounding the deployment of Convolutional Neural Network solutions on FPGAs in a radiation susceptible environment. The first part documented a practical CNN processing core, which can be used to implement a wide range of CNN solutions. The second part, discussed the Space Hardening of that core, adding in Triple-Mode Redundancy for radiation effect tolerance to control path circuitry. This third part documents the higher level control structures necessary to move data to and from the CNN core and dynamically reconfigure its operation to match the functional requirements of a part of a network. The fourth part of this series will document the deployment of this FPGA solution on the Alpha Data ADA-SDEV-KIT3 Space Development kit for the Xilinx XQRKU060 FPGA device.

Data Movement From Memory to and from the DPU

With a flexible DPU-based solution for the CNN implementation, the low silicon footprint of the processor is balanced by the need to store data for the network off-chip. This can be essential for large networks, with memory footprints in excess of available on-chip memory resources. But even with smaller networks it can allow a very compact solution. However, as the data needs transferred, from memory, the application becomes memory bound. This applies to the weight configuration data of the network as well as the feature data for each layer where the input and output need to be read from and written back to memory for each layer as it it processed. With a single DPU implementing each layer in sequence, memory efficient techniques such as passing the output of one layer directly as inputs to the next layer (possible with multi-layer hardware) is not an option.

Moving data efficiently around is often one of the more tricky design tasks when using FPGAs. The reconfigurability of the hardware can however allow near optimal scheduling of the data transfers, achieving very high efficiency when correctly designed, providing one reason why FPGA designs can utilize a far higher percentage of their theoretical computational capability than other silicon architectures with fixed memory access infrastructure. The use of off-the-shelf IP cores can however simplify this design process. In this case the Xilinx DataMover core is ideal for handling the reading of a chunk of data from memory and converting it to the stream inputs required by the DPU for both weights and features. The DataMover IP core can also be configured to write the output stream of data to a different address in the memory block. For intermediate network layers, this will be used later as input features for the next layer. The DPU architecture selected, works most efficiently with larger batches as the weights are read in first, with the feature data pushed through after, and therefore batching allows many frames of features to be processed for each load from memory of the weight data.

DPU and Data Flow Control

To be really useful the DPU needs a higher level configuration and control structure. To handle the flexibility required, this needs to be programmable. Therefore, a simple state machine, or processor needs to be included in the design to perform the DPU configuration and schedule data transfers. To be programmable, each instruction word must be stored in memory, and readable by the processor. A block diagram indicating the connections between this control block, the DPU and the data mover cores is shown in figure 1.

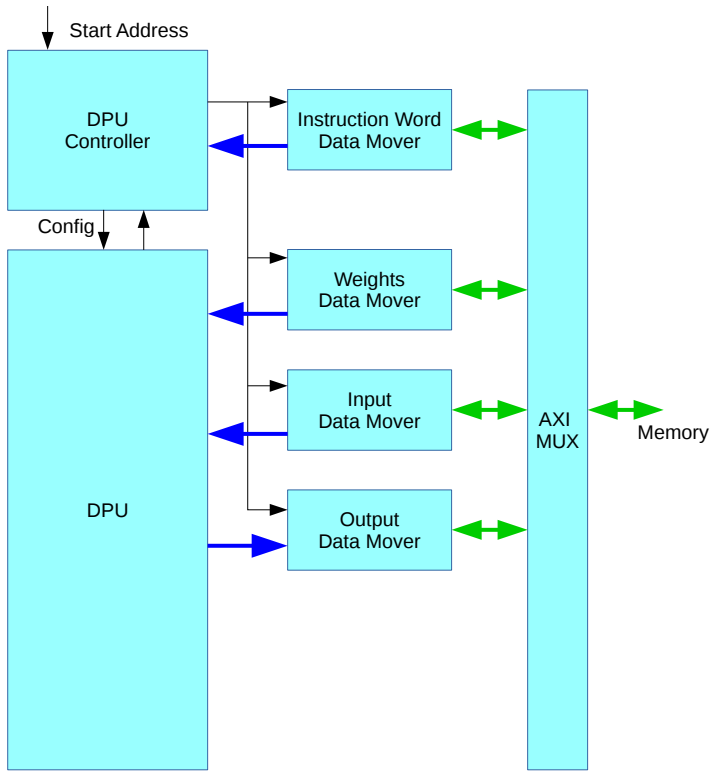

Figure 1 : DPU Control and Data Flow Block Diagram

The processing unit implemented is based around using a 1024-bit instruction word, which contains all the configuration data required to run many different layers. The following address map shows the format and fields of this instruction word, with some space reserved at this stage for future use and will be used later to implement some of the special features of the Yolo network. Fields provide the configuration of DPU layer features, selecting ReLU or Linear operation, 3x3 input convolution or 2x2 MaxPool or stride operations. Fields also fully specify the network layer the DPU is to implement in terms of numbers of features and neurons. Fields are also used to control the DataMover IP, specify the start address and size of the weights and feature data to be read in, and the size and location of the memory area for the results to be written to. To enable multi-layer and continuous operation, the programme word also contains a next word address, which is loaded and processed when the current layer operation is complete.

Instruction Word Format

Register	Address	Description
DPU CFG	0x0000	DPU Configuration
WDM Command	0x0010	Command to send to Weights Data Mover
IDM Command	0x0020	Command to send to Input Stream Data Mover
ODM Command	0x0030	Command to send to Output Data Mover
Next Instruction Word	0x0040	Address and Control of Next Instruction Word to allow automatic scheduling of next DPU layer operation.
Unused	0x0050	Reserved for Future Use
Unused	0x0060	Reserved for Future Use
Unused	0x0070	Reserved for Future Use

1024 bit Instruction Word format for DPU Controller, split into 8x128 bit sections.

DPU CFG (0x0000)

DPU Configuration

127:106	105:96	95:-90	89:80	79:-76	75:64	63:-62	61:48	47:-44	43:32	31:-30	29:16	15:4	3	2	1	0
R	TR	R	AN	R	MPF	R	MPW	R	NF	R	FIW	R	S2	MP	CO	RE

Field	Bit(s)	Mode	Description
ReLU (RE)	0	RW	Use ReLU non-linearity
Conv 3x3 (CO)	1	RW	Enable 3x3 convolution of input
Maxpool (MP)	2	RW	Enable MaxPool Layer port Neuron Output
Stride 2 (S2)	3	RW	Enable Stride of 2 across input data
Reserved (R)	15:4	RW	Reserved
Feature Image Width (FIW)	29:16	RW	Width of Image
Reserved (R)	31:30	RW	Reserved
Number of Features (NF)	43:32	RW	Number of Input Features
Reserved (R)	47:44	RW	Reserved
MP Width (MPW)	61:48	RW	Width of Max Pool Input
Reserved (R)	63:62	RW	Reserved
MP Features (MPF)	75:64	RW	Number of Max Pool Inputs

Field	Bit(s)	Mode	Description
Reserved (R)	79:76	RW	Reserved
Active Neurons (AN)	89:80	RW	Number of active neurons
Reserved (R)	95:90	RW	Reserved
Throttle Rate (TR)	105:96	RW	Input rate flow control
Reserved (R)	127:106	RW	Reserved

WDM Command (0x0010)

Command to send to Weights Data Mover

127:100	99:96	95:32	31	30	29:24	23	22:0
R	TAG	SA	DRR	EOF	DSA	T	BTT

Field	Bit(s)	Mode	Description
Bytes to Transfer (BTT)	22:0	RW	Size in bytes of the weights data
Type (T)	23	RW	Set to 1 for AXI INCR address Operation
DRE Stream Alignment (DSA)	29:24	RW	Not used: Set to Zero
End of Frame (EOF)	30	RW	End of Frame Command : Set to 1
DRE ReAlignment Request (DRR)	31	RW	Not used: Set to Zero
Start address (SA)	95:32	RW	Start address of the weights data
Command TAG (TAG)	99:96	RW	Command TAG for Data Mover
Reserved (R)	127:100	RW	Reserved

IDM Command (0x0020)

Command to send to Input Stream Data Mover

127:100	99:96	95:32	31	30	29:24	23	22:0
R	TAG	SA	DRR	EOF	DSA	T	BTT

Field	Bit(s)	Mode	Description
Bytes to Transfer (BTT)	22:0	RW	Size in bytes of the input feature data
Type (T)	23	RW	Set to 1 for AXI INCR address Operation
DRE Stream Alignment (DSA)	29:24	RW	Not used: Set to Zero

Field	Bit(s)	Mode	Description
End of Frame (EOF)	30	RW	End of Frame Command : Set to 1
DRE ReAlignment Request (DRR)	31	RW	Not used: Set to Zero
Start address (SA)	95:32	RW	Start address of the weights data
Command TAG (TAG)	99:96	RW	Command TAG for Data Mover
Reserved (R)	127:100	RW	Reserved

ODM Command (0x0030)

Command to send to Output Data Mover

127:100	99:96	95:32	31	30	29:24	23	22:0
R	TAG	SA	DRR	EOF	DSA	T	BTT

Field	Bit(s)	Mode	Description
Bytes to Transfer (BTT)	22:0	RW	Size in bytes of the output data
Type (T)	23	RW	Set to 1 for AXI INCR address Operation
DRE Stream Alignment (DSA)	29:24	RW	Not used: Set to Zero
End of Frame (EOF)	30	RW	End of Frame Command : Set to 0
DRE ReAlignment Request (DRR)	31	RW	Not used: Set to Zero
Start address (SA)	95:32	RW	Start address of the weights data
Command TAG (TAG)	99:96	RW	Command TAG for Data Mover
Reserved (R)	127:100	RW	Reserved

Next Instruction Word (0x0040)

Address and Control of Next Instruction Word to allow automatic scheduling of next DPU layer operation.

127:65	64	63:0
R	NIV	NIA

Field	Bit(s)	Mode	Description
Next Instruction Address (NIA)	63:0	RW	Size in bytes of the input feature data
Next Instruction Valid (NIV)	64	RW	Set to 1 for AXI INCR address Operation
Reserved (R)	127:65	RW	Reserved

Unused (0x0050)

Reserved for Future Use

127:0
R

Field	Bit(s)	Mode	Description
Reserved (R)	127:0	RW	Reserved

Unused (0x0060)

Reserved for Future Use

127:0
R

Field	Bit(s)	Mode	Description
Reserved (R)	127:0	RW	Reserved

Unused (0x0070)

Reserved for Future Use

127:0
R

Field	Bit(s)	Mode	Description
Reserved (R)	127:0	RW	Reserved

Figure 2 shows the flow chart of this DPU control module. When given a start address, it fetches the instruction word located there. This then configures the dynamic parameters of the network, before triggering a read of the weights from memory. Once these have been read into the DPU internal memory, the processor triggers both the result write back and feature reads. Once the last output word is written, the processor checks for a follow-on start address, and if one is present, it fetches and processes the word there.

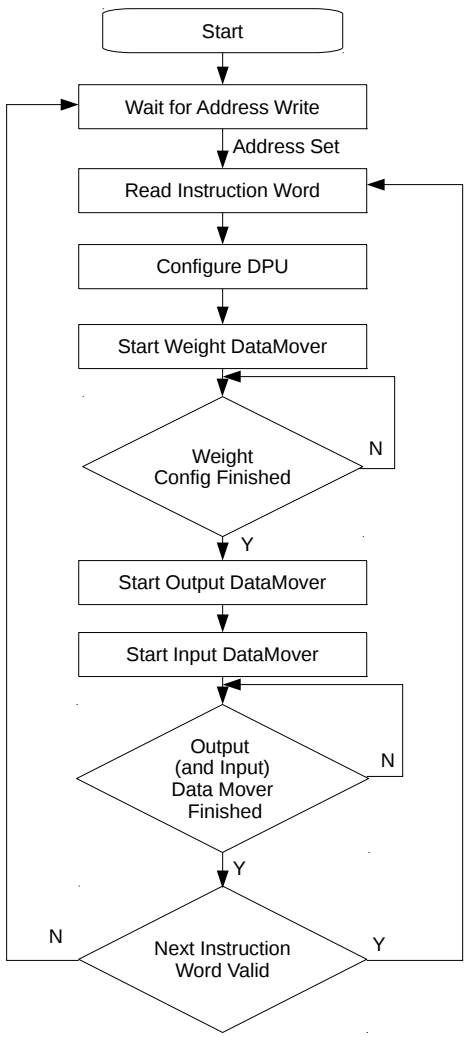

Figure 2 : Flowchart for DPU Control

The Vivado simulation of this DPU control and data movement is demonstrated by the project that can be generated by TCL script *dpuwrapper/prj/mkxpr-1ldpu-wrap.tcl*. This builds a project with a top level, the DPU, the dpu controller and all the data movement infrastructure and a simulation memory - pre-initialized with data to implement a few basic CNN layers.

Support for YoloV3 Layers

To implement the YoloV3 network requires some features beyond that supported in the basic DPU controller simulated in this first project. The first additional feature required to be supported is networks with more neurons than the DPU contains. This is implemented by running different groups of neurons effectively as independent layers sequentially in time. However, to output the data as a contiguous region suitable for use by the next layer, rather requiring a re-arrangement of data after the DPU runs for the different sections of the network have completed, a write striping modification to the output write DataMover, allows the data to be only written once into the correct format. This striping is easily implemented by splitting up the write commands to memory into contiguous stripes of data, for all neurons for each pixel, followed by an address jump to the start address for the next pixel, and thus the output from each stage gets interleaved together into the correct order.

The second YoloV3 feature required is the output of both layers 8 and 9, that is with the same convolution operation, but before and after the MaxPool operations. Since the DPU feeds the MaxPool operation directly from convolved neuron outputs, this data is not automatically written to memory. The basic DPU controller could support this by running the layer twice, once with MaxPool disabled and once with the MaxPool enabled. However, as the pre-maxpool data is available when the MaxPool is enabled, the alternative approach taken here is to slightly modify the DPU to allow this data to be output as a stream. The DPU wrapper then has a second write DataMover IP core implemented to simultaneously write this data to a different area of memory.

The third YoloV3 special feature comes in with Layer 20. This takes in the layer 8 output that the second write output can be used to capture and merges it with the output of layer 17. Layer 8 is 26x26 in size whereas layer 17 is 13x13 and therefore some rescaling of the data from layer 17 is also required for merging. To support all these requires a few extra logic blocks. An additional DataMover is required to read from a second memory area concurrently. The layer 17 data needs a re-scale expansion operation to replicate each pixel, up to 2x2 pixels. This involves a small memory buffer to output each pixel twice, and then repeat each line. Finally, these 2 streams need merged in sequence to form a single stream to feed into the DPU.

The YoloV3 network required a few small changes to the dataflow to implement special operations. Many other networks may require some special fixes to solve such issues, and this can make a fully general purpose DPU difficult to design. One solution, if an embedded CPU is available, is simply to apply the original software operation directly on the data in memory. As these operations are often of low computational complexity this may be the most flexible solution. But many of these operations can be solved with simple streaming operations on the existing data requiring only minor customizations of the DPU logic and its wrapper, with potentially a very low logic cost which may be preferrable to a CPU patch, which could slow throughput due to the required number of memory operations. A block diagram of the control and data flow for this enhanced DPU is shown in figure 3.

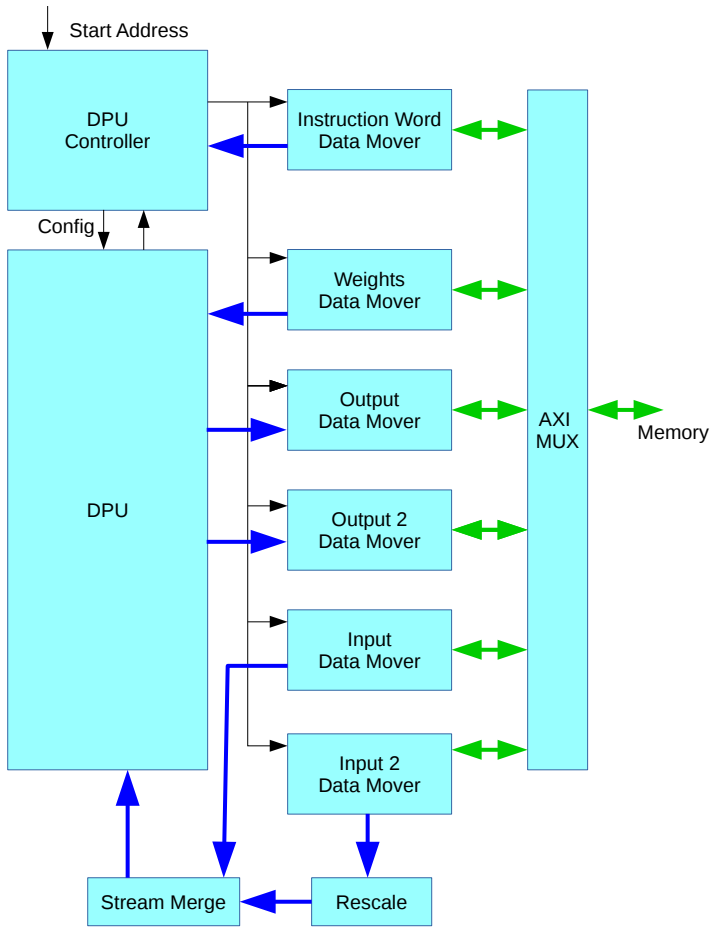

Figure 3 : YoloV3 DPU Control and Data Flow Block Diagram

The modified DPU wrapper project to support these extra features can be built with the scripts *dpuwrapper/prj/mkxpr-1ldpu-wrap2.tcl* and *dpuwrapper/prj/mkxpr-1ldpu-wrap2a.tcl*. The difference between these 2 projects is the data image in memory. In the first case, the simulated memory block is identical to the previous simulation and this simulation run only validates that these extra changes do not affect the previously demonstrated operation. In the second project a more complex memory image is built up in the simulation memory, including data to simulate the layers 8,9 and 20 that require these new features.

The field definitions for the enhanced instruction word format used for this more complex DPU controller are as follows:

Enhanced Instruction Word Format

Register	Address	Description
DPU CFG	0x0000	DPU Configuration
WDM Command	0x0010	Command to send to Weights Data Mover
IDM Command	0x0020	Command to send to Input Stream Data Mover
ODM Command	0x0030	Command to send to Output Data Mover
Next Instruction Word	0x0040	Address and Control of Next Instruction Word to allow automatic scheduling of next DPU layer operation.
IDM2 Command	0x0050	Command to send to Second Input Stream Data Mover
YoloV3 Misc	0x0060	Miscellaneous functions to support YoloV3 specific layers
ODM2 Command	0x0070	Command to send to Second Output Data Mover

1024 bit Instruction Word format for DPU Controller, split into 8x128 bit sections.

DPU CFG (0x0000)

DPU Configuration

127:106	105:96	95:-90	89:80	79:-76	75:64	63:-62	61:48	47:-44	43:32	31:-30	29:16	15:4	3	2	1	0
R	TR	R	AN	R	MPF	R	MPW	R	NF	R	FIW	R	S2	MP	CO	RE

Field	Bit(s)	Mode	Description
ReLU (RE)	0	RW	Use ReLU non-linearity
Conv 3x3 (CO)	1	RW	Enable 3x3 convolution of input
Maxpool (MP)	2	RW	Enable MaxPool Layer port Neuron Output
Stride 2 (S2)	3	RW	Enable Stride of 2 across input data
Reserved (R)	15:4	RW	Reserved
Feature Image Width (FIW)	29:16	RW	Width of Image
Reserved (R)	31:30	RW	Reserved
Number of Features (NF)	43:32	RW	Number of Input Features
Reserved (R)	47:44	RW	Reserved
MP Width (MPW)	61:48	RW	Width of Max Pool Input
Reserved (R)	63:62	RW	Reserved
MP Features (MPF)	75:64	RW	Number of Max Pool Inputs

Field	Bit(s)	Mode	Description
Reserved (R)	79:76	RW	Reserved
Active Neurons (AN)	89:80	RW	Number of active neurons
Reserved (R)	95:90	RW	Reserved
Throttle Rate (TR)	105:96	RW	Input rate flow control
Reserved (R)	127:106	RW	Reserved

WDM Command (0x0010)

Command to send to Weights Data Mover

127:100	99:96	95:32	31	30	29:24	23	22:0
R	TAG	SA	DRR	EOF	DSA	T	BTT

Field	Bit(s)	Mode	Description
Bytes to Transfer (BTT)	22:0	RW	Size in bytes of the weights data
Type (T)	23	RW	Set to 1 for AXI INCR address Operation
DRE Stream Alignment (DSA)	29:24	RW	Not used: Set to Zero
End of Frame (EOF)	30	RW	End of Frame Command : Set to 1
DRE ReAlignment Request (DRR)	31	RW	Not used: Set to Zero
Start address (SA)	95:32	RW	Start address of the weights data
Command TAG (TAG)	99:96	RW	Command TAG for Data Mover
Reserved (R)	127:100	RW	Reserved

IDM Command (0x0020)

Command to send to Input Stream Data Mover

127:100	99:96	95:32	31	30	29:24	15:12	22:0
R	TAG	SA	DRR	EOF	DSA	T	BTT

Field	Bit(s)	Mode	Description
Bytes to Transfer (BTT)	22:0	RW	Size in bytes of the input feature data
Type (T)	15:12	RW	Set to 1 for AXI INCR address Operation
DRE Stream Alignment (DSA)	29:24	RW	Not used: Set to Zero

Field	Bit(s)	Mode	Description
End of Frame (EOF)	30	RW	End of Frame Command : Set to 1
DRE ReAlignment Request (DRR)	31	RW	Not used: Set to Zero
Start address (SA)	95:32	RW	Start address of the weights data
Command TAG (TAG)	99:96	RW	Command TAG for Data Mover
Reserved (R)	127:100	RW	Reserved

ODM Command (0x0030)

Command to send to Output Data Mover

127:104	103-:100	99:96	95:32	31	30	29:24	23	22:0
CC	R	TAG	SA	DRR	EOF	DSA	T	BTT

Field	Bit(s)	Mode	Description
Bytes to Transfer (BTT)	22:0	RW	Size in bytes of the output data
Type (T)	23	RW	Set to 1 for AXI INCR address Operation
DRE Stream Alignment (DSA)	29:24	RW	Not used: Set to Zero
End of Frame (EOF)	30	RW	End of Frame Command : Set to 0
DRE ReAlignment Request (DRR)	31	RW	Not used: Set to Zero
Start address (SA)	95:32	RW	Start address of the weights data
Command TAG (TAG)	99:96	RW	Command TAG for Data Mover
Reserved (R)	103:100	RW	Reserved
Command Count (CC)	127:104	RW	Number of times to repeat command with address increment

Next Instruction Word (0x0040)

Address and Control of Next Instruction Word to allow automatic scheduling of next DPU layer operation.

127:65	64	63:0
R	NIV	NIA

Field	Bit(s)	Mode	Description
Next Instruction Address (NIA)	63:0	RW	Size in bytes of the input feature data
Next Instruction Valid (NIV)	64	RW	Set to 1 for AXI INCR address Operation
Reserved (R)	127:65	RW	Reserved

IDM2 Command (0x0050)

Command to send to Second Input Stream Data Mover

127:100	99:96	95:32	31	30	29:24	23	22:0
R	TAG	SA	DRR	EOF	DSA	T	BTT

Field	Bit(s)	Mode	Description
Bytes to Transfer (BTT)	22:0	RW	Size in bytes of the input feature data
Type (T)	23	RW	Set to 1 for AXI INCR address Operation
DRE Stream Alignment (DSA)	29:24	RW	Not used: Set to Zero
End of Frame (EOF)	30	RW	End of Frame Command : Set to 1
DRE ReAlignment Request (DRR)	31	RW	Not used: Set to Zero
Start address (SA)	95:32	RW	Start address of the weights data
Command TAG (TAG)	99:96	RW	Command TAG for Data Mover
Reserved (R)	127:100	RW	Reserved

YoloV3 Misc (0x0060)

Miscellaneous functions to support YoloV3 specific layers

127:112	111:96	95:64	63:48	47:32	31:1	0
O2I	O1I	R	RC2	RC1	R	RE

Field	Bit(s)	Mode	Description
Rescale Enable (RE)	0	RW	Reserved
Reserved (R)	31:1	RW	Reserved
Rescale Feature Count 1 (RC1)	47:32	RW	Number of features from IDM to use
Rescale Feature Count 2 (RC2)	63:48	RW	Number of features from IDM2 to use

Field	Bit(s)	Mode	Description
Reserved (R)	95:64	RW	Reserved
ODM Address Increment (O1I)	111:96	RW	Address increment for striping ODM writes
ODM2 Address Increment (O2I)	127:112	RW	Address increment for striping ODM2 writes

ODM2 Command (0x0070)

Command to send to Second Output Data Mover

127:104	103-:100	99:96	95:32	31	30	29:24	23	22:0
CC	R	TAG	SA	DRR	EOF	DSA	T	BTT

Field	Bit(s)	Mode	Description
Bytes to Transfer (BTT)	22:0	RW	Size in bytes of the output data
Type (T)	23	RW	Set to 1 for AXI INCR address Operation
DRE Stream Alignment (DSA)	29:24	RW	Not used: Set to Zero
End of Frame (EOF)	30	RW	End of Frame Command : Set to 0
DRE ReAlignment Request (DRR)	31	RW	Not used: Set to Zero
Start address (SA)	95:32	RW	Start address of the weights data
Command TAG (TAG)	99:96	RW	Command TAG for Data Mover
Reserved (R)	103:100	RW	Reserved
Command Count (CC)	127:104	RW	Number of times to repeat command with address increment

Triple-Mode Redundant Operation

As this wrapper is primarily control logic for the DPU, to work with the TMR enhanced version of the DPU, this part of the design should also run in a triple-mode redundant mode. This can be achieved in a relatively straight forward way by instantiating 3 copies of the logic. Where three copies of the control signals are required, these are directly fed through to the DPU. The triple - signals to and from the memory interface are resolved within the core to provide a standard AXI interface out to external logic.

A simulation project to test out these changes to the design can be built using the script *dpuwrapper/prj/mkxpr-1ldpu-wrap3.tcl*

Conclusions and Next Steps

This section of the paper has described the higher level configuration and control of the DPU core, which along with the DataMover IP allows the core to be configured, controlled and fed with data from an AXI-accessible memory block. Enhancements to the basic structure to efficiently support network-specific special features have been added, to allow the DPU to implement certain layers of the YoloV3 network. The implications of Triple-Mode Redundancy on this part of the design have also been briefly discussed.

The fourth and final section of this paper will take the DPU core, the wrapper control and data flow IP and provide a deployable implementation. This will primarily consist of providing AXI access to an external DDR3 memory bank, register access to allow the start address to be set, and external access by PCIe to allow the DDR3 memory to be loaded with the DPU instruction word programs and weight data for different layers. The DDR3 also needs to be accessible to load the images to be processed by the network, and to output the results.

Revision History

Date	Revision	Nature of Change
12/10/21	1.0	First draft

Address: Suite L4A, 160 Dundee Street, Edinburgh, EH11 1DQ, UK
Telephone: +44 131 558 2600
Fax: +44 131 558 2700
email: sales@alpha-data.com
website: http://www.alpha-data.com

Address: 10822 West Toller Drive, Suite 250 Littleton, CO 80127
Telephone: (303) 954 8768
Fax: (866) 820 9956 - toll free
email: sales@alpha-data.com
website: http://www.alpha-data.com