CSEE 4840 Embedded System Design Final Project Report

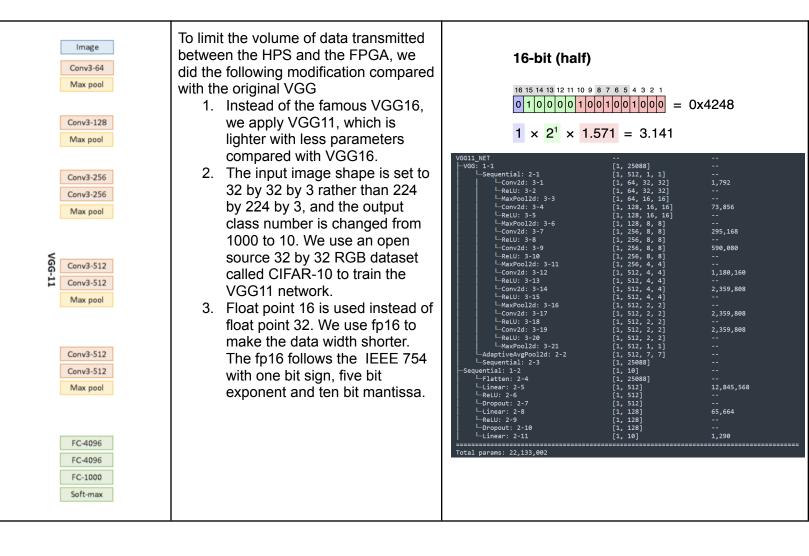
Convolutional Neural Network

1. Introduction

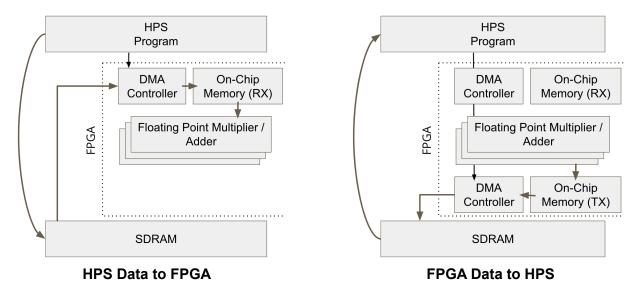
Convolutional Neural Network (CNN) is widely used in the machine learning task in the computer vision and neural language processing area. In this project, we implement the convolutional neural network algorithm on the DE-1 SOC FPGA + HPS to run a pre-trained CNN-based network: VGG-11.

2. Data Flow

The figure below shows the structure of the VGG11, which contains these types of operator: conv2d, Relu, max pooling 2d, adaptive average pooling, linear(fully connection).



3. Hardware System Architecture



The system is built with two main components, HPS and FPGA that work with each other. From high level, the HPS is in charge of interfacing with the user and obtains the initial data, whereas the FPGA is in charge of low level computations.

The system is first configured and generated using qsys (platform designer). The qsys configuration is as follows:

Columbia University Spring 2022

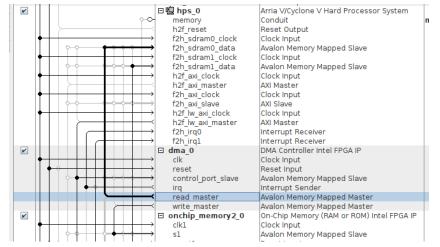
	Connections	Name	Description Clock Source	Export	Clock	Base	End	IRQ
	D-	clk_in	Clock Input	clk	exported			
0		clk_in_reset	Reset Input	reset	-			
		clk clk_reset	Clock Output Reset Output	Double-click to export Double-click to export	clk_0			
		⊟ III hps_0	Arria V/Cyclone V Hard Processor System					
	<u> </u>	memory h2f reset	Conduit Reset Output	memory Double-click to export				
++-	(f2h_sdram0_clock	Clock Input	Double-click to export	clk_0			
		f2h_sdram0_data	Avalon Memory Mapped Slave	Double-click to export	[f2h_sdram0_clock]	= 0x0000_0000	Oxffff_fff	
	· · · · · · · · · · · · · · · · · · ·	f2h_sdram1_clock f2h_sdram1_data	Clock Input Avalon Memory Mapped Slave	Double-click to export Double-click to export	clk_0 [f2h_sdram1_clock]	= 0x0000_0000	0xffff_ffff	
•	\rightarrow	h2f_axi_clock	Clock Input	Double-click to export	clk_0	- 0x0000_0000	0,1111_1111	
		h2f_axi_master	AXI Master	Double-click to export	[h2f_axi_clock]			
	· · · · · · · · · · · · · · · · · · ·	f2h_axi_clock f2h_axi_slave	Clock Input AXI Slave	Double-click to export Double-click to export	clk_0 [f2h_axi_clock]	-		
+		h2f_lw_axi_clock	Clock Input	Double-click to export	clk_0			
		h2f_lw_axi_master	AXI Master	Double-click to export	[h2f_lw_axi_clock]	700		
		f2h_irq0 f2h_irq1	Interrupt Receiver Interrupt Receiver	Double-click to export Double-click to export		IRQ IRQ		
		🗆 dma_0	DMA Controller Intel FPGA IP					
		clk	Clock Input	Double-click to export	clk_0			
	$\land \bullet \bullet$	reset control_port_slave	Reset Input Avalon Memory Mapped Slave	Double-click to export Double-click to export	[clk] [clk]	■ 0x0000_0000	0x0000_001f	
		irq	Interrupt Sender	Double-click to export	[clk]		0.0000_0011	<u>⊢</u> ₫-
		read_master	Avalon Memory Mapped Master	Double-click to export	[clk]			
		write_master onchip_memory2_0	Avalon Memory Mapped Master On-Chip Memory (RAM or ROM) Intel FPGA IP	Double-click to export	[clk]			
	\rightarrow	clk1	Clock Input	Double-click to export	clk_0			
		sl	Avalon Memory Mapped Slave	Double-click to export	[clk1]	= 0x0000	0x0fff	
	,	resetl s2	Reset Input Avalon Memory Mapped Slave	Double-click to export onchip_memory2_0_s2	[clk1] [clk2]	₽		
	\rightarrow	clk2	Clock Input	Double-click to export	clk_0			
	$\diamond + + + + + + \rightarrow$	reset2	Reset Input	Double-click to export	[clk2]			
		onchip_memory2_1	On-Chip Memory (RAM or ROM) Intel FPGA IP					
+	\rightarrow	clk1	Clock Input	Double-click to export	clk_0			
		sl resetl	Avalon Memory Mapped Slave Reset Input	onchip_memory2_1_s1 Double-click to export	[clk1] [clk1]			
	\diamond	s2	Avalon Memory Mapped Slave	Double-click to export	[clk2]	<i>≕</i> 0x0000	OxOfff	
+	\rightarrow	clk2	Clock Input	Double-click to export	clk_0			
1		reset2 dma_1	Reset Input DMA Controller Intel FPGA IP	Double-click to export	[clk2]			
♦ - -		clk	Clock Input	Double-click to export	clk_0			
♦→	\rightarrow	reset	Reset Input	Double-click to export	[clk]			
		control_port_slave irq	Avalon Memory Mapped Slave Interrupt Sender	Double-click to export Double-click to export	[clk] [clk]		0x0000_003f	, → lo
	\square	read_master	Avalon Memory Mapped Master	Double-click to export	[clk]			
		write_master	Avalon Memory Mapped Master	Double-click to export	[clk]			
		image_sent_ocm clk	PIO (Parallel I/O) Intel FPGA IP Clock Input	Double-click to export	clk_0			
II ₩		reset	Reset Input	Double-click to export Double-click to export	[clk]			
	$\diamond \bullet \bullet \bullet \bullet \bullet \rightarrow$	sl	Avalon Memory Mapped Slave	Double-click to export	[clk]	e 0x0000_0040	0x0000_004f	
	~~	external_connection	Conduit	image_sent_ocm				
		fpga_stat clk	PIO (Parallel I/O) Intel FPGA IP Clock Input	Double-click to export	clk_0			
🔶	+ + + + + + + + + + + + + + + + + + +	reset	Reset Input	Double-click to export	[clk]			
		sl external connection	Avalon Memory Mapped Slave	Double-click to export	[clk]		0x0000_005f	
		external_connection h2f_start	Conduit PIO (Parallel I/O) Intel FPGA IP	fpga_stat				
+		clk	Clock Input	Double-click to export	clk_0			
†		reset sl	Reset Input Avalan Memory Manped Slave	Double-click to export	[clk] [clk]		0x0000 006f	
		si external_connection	Avalon Memory Mapped Slave Conduit	Double-click to export h2f_start	[cik]	00000_0000	0x0000_0001	
		f2h_start	PIO (Parallel I/O) Intel FPGA IP	_				
		clk	Clock Input	Double-click to export	clk_0			
II TÎ		reset sl	Reset Input Avalon Memory Mapped Slave	Double-click to export Double-click to export	[clk] [clk]	= 0x0000_0070	0x0000_007f	
		external_connection	Conduit	f2h_start				
		D hof finish	BIO (Barallal I/O) Intel 50.04 ID	-				
++-	,	h2f_finish clk	PIO (Parallel I/O) Intel FPGA IP Clock Input	Double-click to export	clk_0			
+	• • • • • • • • • • • • • • • • • • •	reset	Reset Input	Double-click to export	[clk]			
		sl external connection	Avalon Memory Mapped Slave	Double-click to export	[clk]	= 0x0000_0080	0x0000_008f	
		external_connection f2h_finish	Conduit PIO (Parallel I/O) Intel FPGA IP	h2f_finish				
++-	· · · · · · · · · · · · · · · · · · ·	clk	Clock Input	Double-click to export	clk_0			
		reset sl	Reset Input Avalon Memory Mapped Slave	Double-click to export Double-click to export	[clk] [clk]		0x0000_009f	
	00	external_connection	Conduit	f2h_finish	[CIV]	0000_0090	0000_0091	
		h2f_read_length	PIO (Parallel I/O) Intel FPGA IP	-				
1	· · · · · · · · · · · · · · · · · · ·	clk reset	Clock Input Reset Input	Double-click to export Double-click to export	clk_0 [clk]			
		sl	Avalon Memory Mapped Slave	Double-click to export Double-click to export	[clk]	= 0x0000_0100	0x0000_010f	
	·•	external_connection	Conduit	h2f_read_length			_	
		f2h_write_length clk	PIO (Parallel I/O) Intel FPGA IP Clock Input	Double-click to export	clk_0			
↓	• 	reset	Reset Input	Double-click to export	[clk]			
		sl	Avalon Memory Mapped Slave	Double-click to export	[clk]	= 0x0000_0110	0x0000_011f	
	00	external_connection h2f_buf_offset	Conduit PIO (Parallel I/O) Intel FPGA IP	f2h_write_length				
+		clk	Clock Input	Double-click to export	clk_0			
•	• • • • • • • • • • • • • • • • • • •	reset	Reset Input	Double-click to export	[clk]			
		sl ovternal connection	Avalon Memory Mapped Slave	Double-click to export	[clk]	= 0x0000_0120	0x0000_012f	
	\land	external_connection f2h buf offset	Conduit PIO (Parallel I/O) Intel FPGA IP	h2f_buf_offset				
	$ + + + + \rightarrow$	clk	Clock Input	Double-click to export	clk_0			
	+ 1 + + + + + + + + + + + + + + + + +	reset	Reset Input	Double-click to export	[clk]		0.0000 0000	
		s1 external_connection	Avalon Memory Mapped Slave Conduit	Double-click to export f2h_buf_offset	[clk]	= 0x0000_0130	0x0000_013f	
		■ feat_map_dim	PIO (Parallel I/O) Intel FPGA IP	izi_bui_onset				
an 1 - 1 - 1		clk	Clock Input	Double-click to export	clk_0			
•			Reset Input	Double-click to export	[clk]			
•		reset sl	Avalon Memory Mapped Slave	Double-click to export	[clk]		0x0000_014f	

The detailed configuration of each block is as follows:

 The HPS system is configured with two DDR3 SDRAM interfaces, connected through the Avalon Memory Mapped interface as shown below. One is configured to read-only and the other as write-only.

* AXI Bridges				
FPGA-to-HPS interface v	64-bit	•		
HPS-to-FPGA interface v	64-bit	•		
Lightweight HPS-to-FPG	A interface width:	32-bit	•	
* FPGA-to-HPS SDRAM	Interface			
Click the '+' and '-' butt	ons to add and re	move FPG	A-to-HPS	SDRAM ports.
Name	Туре		Width	
f2h sdram0	Avalon-MM Rea	d-Only	32	
f2h_sdram1	Avalon-MM Writ	te-Only	32	

 Two DMA-onchip_memory pairs are created for exchanging data between HPS and FPGA. As seen below, the first dma0 has its read_master connected to HPS's sdram0 (read-only), and write_master to onchip_memory0. This dma copies data from HPS to the onchip_memory. Vise versa, dma1 has its read_master connected to on chip memory1 and write master connected to HPS's sdram1 (write-only).



- A list of PIO IP cores are added for HPS to send or receive control signals from / to FPGA. As seen below, the h2f_finish signal has one bit and is configured as an input, with conduit "h2f_finish" and base address 0x0000080.

+ Use Connections	Name	Description	Export	Clock	Base	End
	> sl	Avalon Memory Mapped Slave	Double-click to export	[clk]		0x0000_007f
	external_connection	Conduit	f2h_start			
	h2f_finish	PIO (Parallel I/O) Intel FPGA IP				
	→ clk	Clock Input	Double-click to export	clk_0		
	reset	Reset Input	Double-click to export	[clk]		
	sl	Avalon Memory Mapped Slave	Double-click to export	[clk]	© 0x0000_0080	0x0000_008f
	 external_connection f2h finish 	Conduit PIO (Parallel I/O) Intel FPGA IP	h2f_finish			
	→ clk	Clock Input	Double-click to export	clk_0		
		clock input	Bouble click to export			
		PIO (Parallel I/O) Intel FPGA IP - I	of finish			🚫 009f
PIO (Parallel I/O) Intel FPG altera_avalon_pio	A IP		21_110150		Document	
Core altera_avalori_pio			2f_mnsn		Document	
llock Diagram	Basic Se	ttings	zr_misn		Document	ation
lock Diagram	Basic Se Width (1-3	ttings 12 bits): 1	21_IMISN		Document	ation
lock Diagram Show signals	Basic Se	ttings	21_IMISN		Document	ation
lock Diagram	Basic Se Width (1-3	ttings 2 bits): <u>1</u> O Bidir	21_IMISN		Document	ation
Inck Diagram	Basic Se Width (1-3	ttings 2 bits): <u>1</u> O Bidir © Input	21_IMISH		Document	ation 010f
lock Diagram Show signals h2f_finish etc.	Basic Se Width (1-3	ttings 2 bits): <u>1</u> O Bidir	21_IMISN		Document	ation 010f
Iock Diagram Show signals	Basic Se Width (1-3	ttings 2 bits): <u>1</u> O Bidir © Input	21_IMISN		Document	ation 010f
lock Diagram Show signals h2f_finish etkclack	Vidth (1-3 Direction:	ttings 22 bits): <u>1</u> Bidir @ Input Input InOut Output	21_IMISN	_	Document	ation 010f
are a construction of the second of the seco	Vidth (1-3 Direction:	ttings 12 bits): 1 O Bidir O Input O InOut	21_IMISN		Document	ation 010f

This signal can be used in the top level module as follows, as part of the generated qsys module.

502	// PIO to start the image reading	from OCM
503	<pre>.image_sent_ocm_export</pre>	(start_ocm_wire),
504		
505	.fpga_stat_export	(fpga_stat_wire),
506 507	.h2f_start export	(h2f_start),
508	.h2f_read_length_export	<pre>(h2f_read_length),</pre>
509	.feat map dim export	(feat_map_dim),
510	<pre>.h2f_buf_offset_export</pre>	(h2f_buf_offset),
511	.f2h_start_export	(f2h_start),
512	<pre>.f2h_write_length_export</pre>	(f2h_write_length),
513	.f2h_buf_offset_export	(f2h_buf_offset),
514	haf finish surrent	(b)f finish)
515	$\rightarrow .h2f_finish_export \rightarrow \rightarrow \rightarrow$	(
516 517	.f2h_finish_export	(f2h_finish),
517		

In the HPS code, we first take advantage of the "sopc-create-header-files" tool from quartus to generate a header file that has the configured base addresses defined using the following command:

```
sopc-create-header-files "/path/to/qsys.sopcinfo" --single hps_0.h
--module hps 0
```

250 * Macros for device 'h2f_finish', class 'altera_avalon_pio'	
251 * The macros are prefixed with 'H2F_FINISH_'.	
252 * The prefix is the slave descriptor.	
253 */	
<pre>254 #define H2F_FINISH_COMPONENT_TYPE altera_avalon_pio</pre>	
<pre>255 #define H2F_FINISH_COMPONENT_NAME h2f_finish</pre>	
256 #define H2F_FINISH_BASE 0x80 You, seconds ago • Uncommitted of	
257 #define H2F_FINISH_SPAN 16	
258 #define H2F_FINISH_END 0x8f	
<pre>259 #define H2F_FINISH_BIT_CLEARING_EDGE_REGISTER 0</pre>	
<pre>260 #define H2F_FINISH_BIT_MODIFYING_OUTPUT_REGISTER 0</pre>	

The generated header file includes the defines above, and we can use the PIO in the HPS C code as follows:

```
//lightweight HPS-to-FPGA bridge
void *virtual_base;
virtual_base = mmap( NULL, HW_REGS_SPAN, ( PROT_READ | PROT_WRITE ),
MAP_SHARED, fd, HW_REGS_BASE );
uint32_t * h2f_finish = virtual_base + ( ( unsigned long )(
ALT LWFPGASLVS OFST + H2F FINISH BASE) & ( unsigned long)(
```

HW REGS MASK));

*h2f_finish = 0;

Similarly, as the DMA controller is also on the lightweight HPS-to-FPGA bridge, we are able to command the DMA controller similarly:

```
DMA REG STATUS (h2p_lw_dma_addr0) = 0;
```

Yanchen Liu (yl4189) Columbia University Minghui Zhao (mz2866) Spring 2022 DMA REG READ ADDR(h2p lw dma addr0) = physical addr1; // read from F2SDRAM 0 DMA REG WRITE ADDR(h2p lw dma addr0) = 0; // write to F2SDRAM 1 DMA REG LENGTH(h2p lw dma addr0) = <size>; //write <size> //start the transfer DMA REG CONTROL(h2p lw dma addr0) = DMA CTR BYTE | DMA CTR GO | DMA CTR LEEN; // ======== FPGA ----DMA----> HPS ======== void *h2p lw dma addr1 = NULL; h2p lw dma addr1 = virtual base + ((unsigned long)(ALT LWFPGASLVS OFST + DMA 1 BASE) & (unsigned long) (HW REGS MASK)); DMA REG STATUS(h2p lw dma addr1) = 0; DMA REG READ ADDR(h2p lw dma addr1) = 0; // read from OCM DMA REG WRITE ADDR(h2p lw dma addr1) = physical addr2; // write to SDRAM (DDR3) DMA REG LENGTH(h2p lw dma addr1) = <size>; // read <size> bytes //start the transfer DMA REG CONTROL(h2p lw dma addr1) = DMA CTR BYTE | DMA CTR GO | DMA CTR LEEN; // wait for DMA to be finished waitDMAFinish(h2p lw dma addr1); void waitDMAFinish(void *BASE ADDR) { while(!(DMA REG STATUS(BASE ADDR) & DMA STAT DONE) && (DMA REG STATUS(BASE ADDR) & DMA STAT BUSY)); }

4. Main Modules

4.1 readOCM.sv

This module handles reading from the OCM, including both reading weight and bias, and loading the feature map.

After HPS finishes asking the DMA controller to copy data into OCM, it sends a signal to the "start" wire below, after which the readOCM module will load the weight and bias from OCM into "weight_bias" and raise "in_data_ready" when finished.

The module also reads feature maps from OCM at index "conv_idx", after receiving a rising edge on "start_fm". It will read the feature map into "feat_map_in" and raise "finish_fm" when done.

The detailed interface is as follows.

module	e readO input	•		clk, reset,
		•		
		R€ t from ⊦	ead weight and bias IPS	
	input			start,
	•	•	[15: 0]	read_length,
	input	logic	[5: 0]	read_data_dim,
	-	ut to pi	peline	
	output	0	[(3*3+1)*16-1: 0]	in_data_ready, weight_bias,
	//	Re	ead feature map 3x3	
		t from p	•	
	input	-		start_fm,
	input	logic	[15: 0]	conv_idx,
	-	ut to pi	peline	
	output	•	[3*3*16-1: 0]	finish_fm, feat_map_in,
				iout_map_m,
	// On-C	Chip RA	M 0 s2 (read)	
	•	logic		ocm0_readdata,
	output	•	[16: 0]	ocm0_addr, ocm0_chip,
	output	-		ocm0_clk_enab,
	// Debu	10		
		logic	[2: 0]	debug_state
<u>١</u> .				

);

The module is implemented using a finite state machine structure with the following states. Please refer to the code for detailed implementation. S0: Reset

Columbia University Spring 2022

S1: Prepare to read weight bias

- S2: Read weight bias
- S3: Finished reading weight bias. Wait for feature map request
- S4: Received feature map request, prepare to read
- S5: Read feature map 8 LSB
- S6: Read feature map 8 MSB
- S7: Finished feature map reading

4.2 convOpt.sv

This module implements convolution. Input "in_data_ready" signals that weight and bias is ready, and "in_data_dim" is the dimension of the data. "Weight_bias" is the weight and bias read by readOCM module. The module uses a finite state machine to manage states. Please see the source code for detailed implementation.

module convo input	Opt(logic		clk, reset,
input input	It from p logic logic logic logic		in_data_ready, in_data_dim, weight_bias,
// Req	uest FM	1 from pipeline	
-	t logic		start_fm,
output	logic	[15: 0]	conv_idx,
input input	•	[3*3*16-1: 0]	finish_fm, feat_map_in,
nipor	logio		
input	logic		finish_out,
// Outj	out to pi	peline	
-	out to pi t logic	peline	start_out,
output	t logic t logic	[16: 0]	out_idx,
output	t logic t logic		
outpu outpu outpu	t logic t logic	[16: 0] [3*3*16-1: 0]	out_idx,
outpur outpur outpur	t logic t logic t logic	[16: 0] [3*3*16-1: 0]	out_idx,
outpur outpur outpur	t logic t logic t logic t logic out to H t logic	[16: 0] [3*3*16-1: 0]	out_idx, feat_map_out,
outpur outpur outpur // Outp outpur	t logic t logic t logic t logic out to H t logic	[16: 0] [3*3*16-1: 0] PS	out_idx, feat_map_out,

Columbia University Spring 2022

4.3 writeOCM.sv

This module handles writing data to OCM that will be DMA'd back to HPS. The module uses a finite state machine to manage states. Please see the source code for detailed implementation.

module writeOCM(input logic input logic input logic input logic	[16: 0] [3*3*16-1: 0]	clk, reset, start_out, out_idx, feat_map_out,
output logic		finish_out,
// On-Chip RA output logic output logic output logic output logic output logic output logic		ocm1_writedata, ocm1_addr, ocm1_chip, ocm1_clk_enab, ocm1_write,
output logic	[15: 0]	count,
// Debug output logic);	[2: 0]	debug_state

);

5. Testbench

Version 1: For one 32x32 feature map, total time is 472us Loading 32*32: 40us Computation: 390us Sending 32*32: 40us



Version 2: For one 32x32 feature map, total time is 840us

gnals	Waves	
Time		1
clk=1		
start =1		
Read		
state[2:0] =7	7	
weight_bias[159:0] =3BA3BAEE3A75BF79BE073E813BB	3BA3BAEE3A75BF79BE073E813BB5BA8C3E223E19	
in_data_ready=1		
ocm0_addr[16:0] =00000	00000	
ocm0_readdata[7:0]=19	19	
fm_save_idx[3:0] =A	Α	
feat_map_in[143:0] =000000000000000002D052CC4000	00000000000000000002D052CC400002EC62EC6	
Conv		
state[2:0] =7		7
finish out=0		
start_fm=1		
finish_fm=1		
feat_map_in[143:0] =00000000000000000000000000000000000	0000000000000000000002D052CC400002EC62EC6	
mult_idx[15:0] =1024		1024
add_idx[3:0] =9		9
finish=1		
Write		
state[2:0] =1		1
start_out=0		
out_idx[16:0] =047EE		047EE
finish_out=0		
n_written[4:0]=0		θ
ocml_write=0		
ocm1_addr[16:0] =18431		18431
ocm1_writedata[7:0]=00		00
out_idx[16:0] =047EE		047EE
finish=1		

6. Synthesis result

Flow Status	Successful - Thu May 12 16:22:09 2022
Quartus Prime Version	21.1.0 Build 842 10/21/2021 SJ Lite Edition
Revision Name	cnn_process
Top-level Entity Name	main
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	5,361 / 32,070 (17 %)
Total registers	3925
Total pins	97 / 457 (21 %)
Total virtual pins	0
Total block memory bits	73,728 / 4,065,280 (2 %)
Total DSP Blocks	0/87(0%)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0/6(0%)
Total DLLs	1/4(25%)

	Resource	Usage
1	Estimate of Logic utilization (ALMs needed)	5455
2		
3	 Combinational ALUT usage for logic 	8938
1	7 input functions	156
2	6 input functions	1540
3	5 input functions	1262
4	4 input functions	2119
5	<=3 input functions	3861
4		
5	Dedicated logic registers	3688
6		
7	I/O pins	97
8	I/O registers	80
9	Total MLAB memory bits	0
10	Total block memory bits	73728
11		
12	Total DSP Blocks	0
13		
14	Total DLLs	1
15	Maximum fan-out node	CLOCK_50~input
16	Maximum fan-out	3763
17	Total fan-out	52077
18	Average fan-out	3.95

Columbia University Spring 2022

	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Block Memory Bits	DSP Blocks	Pins	Virtual Pin
1	▼ main	8938 (1)	3688 (0)	73728	0	97	0
1	cnn_hps_system:The_System	3852 (0)	2896 (0)	73728	0	0	0
2	 tbOpt:opt 	5085 (0)	792 (0)	0	0	0	0
1	 [convOpt:cv] 	4385 (162)	250 (250)	0	0	0	0
1	float_adder:fp_add	265 (265)	0 (0)	0	0	0	0
2	float_multi:gen_fkernel[0].fp_mult	437 (437)	0 (0)	0	0	0	0
3	float_multi:gen_fkernel[1].fp_mult	438 (438)	0 (0)	0	0	0	0
4	float_multi:gen_fkernel[2].fp_mult	438 (438)	0 (0)	0	0	0	0
5	float_multi:gen_fkernel[3].fp_mult	438 (438)	0 (0)	0	0	0	0
6	float_multi:gen_fkernel[4].fp_mult	437 (437)	0 (0)	0	0	0	0
7	float_multi:gen_fkernel[5].fp_mult	438 (438)	0 (0)	0	0	0	0
8	float_multi:gen_fkernel[6].fp_mult	438 (438)	0 (0)	0	0	0	0
9	float_multi:gen_fkernel[7].fp_mult	442 (442)	0 (0)	0	0	0	0
10	float_multi:gen_fkernel[8].fp_mult	419 (419)	0 (0)	0	0	0	0
11	 lpm_mult:Mult0 	33 (0)	0 (0)	0	0	0	0
1	mult_ug11:auto_generated	33 (33)	0 (0)	0	0	0	0
2	 readOCM:rd 	653 (503)	499 (499)	0	0	0	0
1	 lpm_mult:Mult1 	31 (0)	0 (0)	0	0	0	0
1	mult_ug11:auto_generated	31 (31)	0 (0)	0	0	0	0
2	 lpm_mult:Mult2 	119 (0)	0 (0)	0	0	0	0
1	mult_li11:auto_generated	119 (119)	0 (0)	0	0	0	0
3	writeOCM:wr	47 (47)	43 (43)	0	0	0	0

6. Other Modules & Test Files Developed

Main.sv: Top module for synthesis

readFMPipeline.sv: Version 1's read from OCM

readImg.sv: Test module

tbEchoWrite.sv: Test bench top module for writing to OCM and echo back

tbFPU.sv: Test bench top module for the float16 module used

tbOpt.sv: Test bench top module for version 2

tbPipeline.sv: Test bench top module for version 1

tbRWFeatureMap.sv: Test bench top module for reading and writing feature map

testEcho.sv: Test bench top module for echoing data from input OCM to output OCM

writeFeatMap.sv: Write feature map to output OCM initial test

writeFMPipeline.sv: Write feature map to output OCM version 1

writeOCM.sv: Write feature map to output OCM version 2

writeOCM8.sv: Initial test to write to OCM 8-bit data