APIR-DSP: An Approximate PIR-DSP Architecture for Error-Tolerant Applications

Yuan Dai[1†], Simin Liu[1†], Yao Lu[1†], Hao Zhou[1], SeyedRamin Rasoulinezhad[2], Philip H.W. Leong[2], Lingli Wang[1*]
[1]*State Key Lab of ASIC and System, Fudan University, Shanghai 201203, China*
[2]*School of Electrical and Information Engineering, The University of Sydney, Australia 2006*
[*]Email: llwang@fudan.edu.cn

2021 International Conference on Field-Programmable Technology (ICFPT) | 978-1-6654-2010-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICFPT52863.2021.9609927

***Abstract*—In error-tolerant applications such as low-precision DNNs and digital filters, approximate arithmetic circuits can significantly reduce hardware resource utilization. In this work we propose an embedded block for field-programmable gate arrays, called APIR-DSP, which incorporates an approximate 9×9 hard multiplier based on the PIR-DSP architecture to improve speed and reduce area. In addition, a DSP unit evaluation platform based on Yosys and VPR which packs multiply accumulate operations into DSP blocks is developed. Using this tool we synthesis designs from Verilog implementations of matrix multiplication in DeepBench and the DoReFaNet low-precision neural network and show that APIR-DSP significantly reduces DSP resources and improves hardware utilization and performance compared with the Xilinx DSP48E2 embedded block. Compared with exact multiplication, it is shown that accuracy loss is optimized with the SNR of an FIR filter being reduced by 1.03 dB. For DNNs, accuracy loss for AlexNet is 0.31% on CIFAR10 dataset and no accuracy loss for LeNet on MNIST dataset is observed. Synthesis results show that the APIR-DSP enjoys an area reduction of 21.60%, critical path reduction of 4.85% and power consumption is reduced by 2.80%, compared with PIR-DSP.**

I. Introduction

Deep learning technology based on the convolutional neural networks has had tremendous impact in a number of diverse applications. However, with the evolution of models trending towards those with greatly increased storage and computational requirements, techniques for their efficient implementation become increasingly important. For example, AlexNet proposed in 2012 can achieve a top-1 error rate of 37.5% on ImageNet by performing 727M floating-point operations with 60.9M parameters in five convolutional layers and three fully-connected layers [1]. In contrast, the more advanced convolutional neural network ResNet152 requires 11.3B floating-point operations, utilizes 152-layers, and achieves a single-model top-1 validation error rate of 19.38% [2]. Large neural network models cannot be deployed in application scenarios where computing resources and memory resources are limited, particularly in embedded devices where delay and power are strictly constrained. To overcome these limitations, a low-precision inference neural network with acceptable accuracy loss is proposed, which reduces storage requirement and computational complexity through approximate arithmetic.

[†]These three authors contributed equally to this work.

For high performance inference, application specific integrated circuit (ASIC) or field-programmable gate array (FPGA) hardware acceleration platforms are preferred as they offer the highest degree of customization. FPGA is programmable and arbitrary data paths can be developed using fine-grained logic resources [3], [4]. Being commercial off-the-shelf products where design costs are amortised over all users, compared with ASICs, FPGAs enjoy lower design cost, faster time to market, and lower production cost for all but the highest volume applications. On FPGAs, efficiency is achieved via embedded digital signal processing (DSP) blocks which can effectively implement multiply-accumulate (MAC) operations prevalent in low-precision neural networks.

FPGA vendors have evolved their architectures to enable low-precision computing. The width supported by typical FPGA DSPs is 8-18 bits. For example, Xilinx provides a scheme of using their DSP48E2 to realize MAC operations [5]. Each DSP block can implement two 8-bit MAC operations with a shared multiplicand and no more than 7 multiplications accumulated. Recently, FPGA vendors have evolved their architectures to efficiently support lower (< 8 bits) precision [6]. For example, AI tensor blocks in Intel Stratix 10 NX FPGA can support up to 30 *int8* MAC operations and 60 *int4* MAC operations [7].

Approximate arithmetic circuits have become a viable energy-saving solution for digital systems [8]. In some error tolerant applications, such as deep neural networks (DNNs) and digital filters, approximate arithmetic circuits can provide many benefits including smaller area, lower critical path delay and lower power consumption. By carefully designing the approximate multiplier, the accuracy loss can be controlled. In addition, the distribution of DNN and filter parameters mainly lies in the range of multiplier with no or little error.

To realize low precision operations and take advantage of approximate arithmetic circuits, in this paper, we propose an architecture supporting approximate computation, based on the PIR-DSP architecture [9]. The new architecture is called APIR-DSP (approximate PIR-DSP), and this work makes the following contributions:

- A novel DSP architecture, based on an approximate 9×9 multiplier which achieves significantly better area, critical path delay and power consumption than a full multiplier.

To the best of our knowledge this is the first approximate DSP embedded block.

- We show a number of applications that APIR-DSP can be used with minimal reduction in computational accuracy. APIR-DSP can improve utilization by supporting a large number of low-precision multiply accumulate operations, while at the same time increasing speed and reducing area.
- An open source programmable DSP block evaluation platform[1], based on the Yosys and VPR open source electronic design automation (EDA) tools, which supports both DSP48E2 and APIR-DSP blocks. This is the first academic platform which can pack multipliers described in Verilog into DSP blocks having large numbers of low-precision multiply accumulate units.

II. BACKGROUND AND RELATED WORK

A. DSP Blocks on Current FPGAs

Xilinx proposed the DSP48E1 architecture for Virtex 7 series FPGA, which contains a 25-bit pre-adder, a 25×18 multiplier and a 48-bit arithmetic logic unit (ALU), and can realize subtraction, addition, accumulation or logical operations.

For the Ultrascale series FPGA, Xilinx proposed the DSP48E2 architecture [10], which is the improved architecture of DSP48E1. In addition, DSP48E2 is the basis architecture of PIR-DSP, and its simplified schematic is shown in Fig. 1. Compared with DSP48E1, the DSP48E2 mainly has the following improvements:

a) Wider multiplier: the multiplier width is improved from 25×18 to 27×18.

b) More flexible pre-adder: the pre-adder width is improved from 25-bit to 27-bit, and A or B can be selected as input to the pre-adder. In addition, the output of pre-adder can be squared.

c) Wide XOR: DSP48E2 can support a 96-bit wide XOR function, which is not supported in DSP48E1.

However, as previously mentioned, there are limitations when DSP48E2 performing low precision computation.

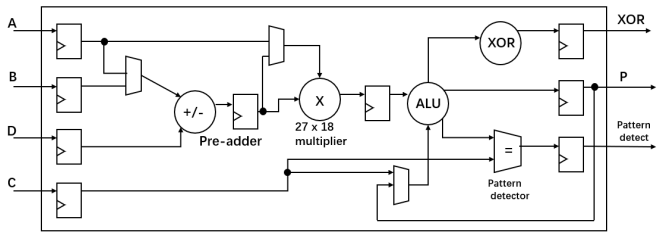

Fig. 1. The simplified schematic of DSP48E2.

The Intel Agilex FPGA is manufactured using a 10nm process, which can be flexibly adapted to AI application scenarios with other functions. The DSP block can support multiple precisions of FP32, BFLOAT16, FP16 and INT8. Floating-point performance of up to 40 TFLOPS can be achieved [11].

In addition, there are many research papers describing the architecture design of multi-precision DSP blocks. A DSP block with a width of 16 that supports run time precision configuration is proposed in [12], where it supports two 8-bit multiplications but with 13% area overhead. In [13], one 18×18 multiplier can be used as eight 2×2 multipliers. A multi-precision DSP block based on radix-4 Booth multiplier is proposed in [14], which supports 9/12/18/24/36 bit multiplications and multi-input additions. In [15], the authors presented a modification of the Arria-10 DSP block, which can pack twice as many 9-bit multipliers and four times as many 4-bit multipliers to support 4×9 or 8×4 MAC operations. The DSP block area is increased by 12%, but the total FPGA area is increased by 0.6% only. The enhanced DSP block is used in two state-of-the-art accelerators and evaluated over different DNN models (ALEXNET, VGG-16, and RESNET-50) with speed acceleration by 1.32×/1.6× and chip area decreased by 15%/30% for 8/4-bits respectively.

A multi-precision, semi-2D interconnection and multiplexing optimized DSP block (PIR-DSP) is proposed in [9], which is the basis for APIR-DSP. In addition to multi-precision multiply accumulate support, PIR-DSP adds a register file (RF) at the input of a DSP48E2 block to support data multiplexing and reduce energy loss in data movement, and changes the interconnection between DSPs to support low-precision cascaded data streams. Compared with the baseline DSP48E2, PIR-DSP reduced runtime power consumption by 31%/19%/13% when implementing a 9/4/2-bit MobileNet-V2 DNN.

B. Open Source Design Flow for FPGA

Yosys is an open source synthesis tool originally developed by Claire Wolf for the Icestorm project [16]. It takes Verilog files as inputs, and can generate netlists in various formats for use with simulators, formal verification, and place and route (PnR) tools. Yosys supports the Xilinx DSP48E1 DSP block and in this paper we have added support for the DSP48E2, PIR-DSP and APIR-DSP blocks.

For placement and routing, a number of open source tools are available. VTR [17] is a research tool to design and test simulated FPGA architectures. It uses an XML based format to describe an FPGA. Another option is Nextpnr [18], which uses programmatic descriptions of FPGA architectures, and is written from the ground up with commercial FPGAs in mind.

Our evaluation platform is based on Yosys and VPR. Together these two tools can map a benchmark circuit in Verilog to the target FPGA architecture, applying synthesis and placement and routing to allow performance to be evaluated.

III. APIR-DSP BLOCK ARCHITECTURE

PIR-DSP uses a divide-and-conquer technique [19] to partition the 27×18 multiplier into six smaller multipliers, each being a signed/unsigned 9-bit multiplication. As shown in Fig. 2(a), a 27×18 multiplication is done by evaluating six

[1]GitHub.com/Dai-dirk/APIR-DSP

partial results with appropriate shifts. Fig. 2(b) shows that by controlling the shift steps for the first, fourth and fifth partial results, the summation can be arranged into two separate columns, therefore, PIR-DSP can compute 2 parallel signed/unsigned (9×9+9×9+9×9) MAC operations and the PIR-DSP multiplier is composed from six 9×9 multipliers. In this paper, we replace these six exact 9×9 multipliers of the PIR-DSP with approximate ones to form a new DSP architecture, called APIR-DSP.

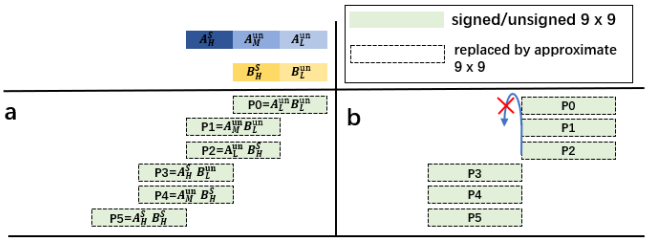

Fig. 2. PIR-DSP multiplication modes, a 27 × 18 bit multiplication is implemented as a number of 9 × 9 operations to support different precisions.

As shown in Fig. 3, an unsigned 9×9 multiplier can be partitioned into nine 3×3 multipliers, where "Pi" ($0 \leq i \leq 8$) is a 6-bit partial result. With appropriate shifts, the final result can be obtained by summing these partial results. Therefore, we use the unsigned approximate 3×3 multipliers to construct an unsigned approximate 9×9 multiplier.

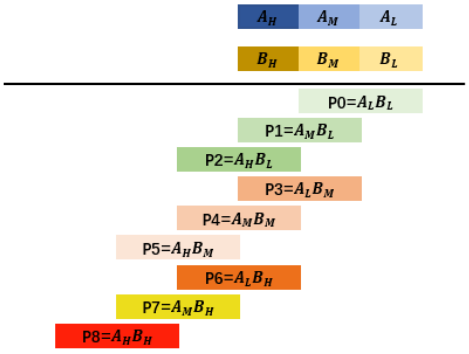

Fig. 3. nine 3×3 multipliers compose a 9×9 multiplier

A. Approximate 3×3 multiplier

A 3×3 multiplier has 6 inputs α_{2-0}, β_{2-0} and 6 outputs O_{5-0}. Modifying the output values of the exact 3×3 multiplier can simplify the logic and decrease resource cost. For 6 inputs, there are 64 (2^6) output values. We modify six cases, as shown in TABLE I, the ***Value*** and ***Value′*** represent exact values and approximate values, respectively. From TABLE I, the output logic function expressions (1)-(4) can be obtained. Compared with the exact multiplier, the logic complexity of our approximate multiplier is lower for synthesis. In reference [20], the authors suggested a number of error metrics including: error distance (ED), error rate (ER), mean error distance (MED), normalized mean error distance (NMED) and mean relative error distance (MRED). In this work we use ED, ER and MED, and their expressions are shown in equations (5)(6)(7), where *m* is the number of values approximated and *n* is the number of input variables. Our approximate multiplier has EDs less than 8, ER is 9.375% and MED is 0.5.

TABLE I
APPROXIMATE TRUTH TABLE FOR 3×3 MULTIPLIER

α_{2-0}	β_{2-0}	O_{5-0}	Value	O'_{5-0}	Value′	ED
101	111	100011	35	**011011**	27	8
110	110	100100	36	10**1000**	40	4
110	111	101010	42	101**110**	46	4
111	101	100011	35	**011011**	27	8
111	110	101010	42	101**110**	38	4
111	111	110001	49	**10110**1	45	4

TABLE II
COMPARISON OF 3×3 APPROXIMATE MULTIPLIER IN TERMS OF AREA, POWER AND DELAY

Name	***Area*(μm^2)**	***Power*(mW)**	***Delay*(nS)**
Exact_mul	67.68	3.73	0.45
Appro_mul	46.44	2.36	0.26
Improvement	31.38%	36.73%	42.22%

$$O'_2 = \overline{\alpha_2}\alpha_1\overline{\alpha_0}\beta_1 + \alpha_1\overline{\beta_2}\beta_1\overline{\beta_0} + \overline{\alpha_2\alpha_1}\alpha_0\beta_2 + \overline{\alpha_2}\alpha_0\beta_2\overline{\beta_1} + \alpha_1\beta_2\beta_1\beta_0 + \alpha_2\overline{\alpha_1\alpha_0}\beta_0 + \alpha_2\overline{\alpha_0}\overline{\beta_1}\beta_0 + \alpha_2\alpha_0\overline{\beta_2}\beta_0 + \alpha_2\alpha_0\beta_2\overline{\beta_0} \quad (1)$$

$$O'_3 = \alpha_1\beta_2\overline{\alpha_0} + \overline{\alpha_2}\alpha_1\alpha_0\overline{\beta_2}\beta_1\beta_0 + \alpha_1\overline{\beta_1}\beta_2 + \alpha_2\overline{\alpha_1}\beta_1 + \alpha_2\alpha_0\beta_0\beta_2 + \alpha_2\beta_1\overline{\beta_0} \quad (2)$$

$$O'_4 = \overline{\alpha_1}\alpha_1\alpha_0\beta_2\beta_1 + \alpha_2\overline{\alpha_1}\beta_2 + \alpha_2\alpha_1\overline{\beta_2}\beta_1\beta_0 + \alpha_2\beta_2\overline{\beta_1} \quad (3)$$

$$O'_5 = \alpha_2\alpha_1\beta_2\beta_1 \quad (4)$$

$$ED = |Value - Value'| \quad (5)$$

$$ER = \frac{m}{2^n} \quad (6)$$

$$MED = \frac{\sum_{i=1}^{2^n} ED}{2^n} \quad (7)$$

Both our approximate multiplier and a baseline exact multiplier are described using Verilog, and synthesized with Synopsys Design Complier (DC) using the ASAP-7nm process library [21] to compare their performances. As shown in TABLE II, our approximate multiplier achieves a 31.38% decrease in area and 36.73% lower power consumption. Delay is decreased by 42.22% compared with the baseline.

B. Approximate PIR-DSP achitecture

The APIR-DSP supports both unsigned and signed inputs, which is based on the 9×9 approximate multipliers, as shown in Fig. 4. If operand A or B is signed, we extract its sign bit and convert the operand into 2's complement form, after which unsigned multiplication is performed. The extracted sign bits of the two inputs are XORed to obtain the sign bit of the result. Finally, we control whether the unsigned multiplication result is complemented according to the signs of the operands, the final result is generated appropriately.

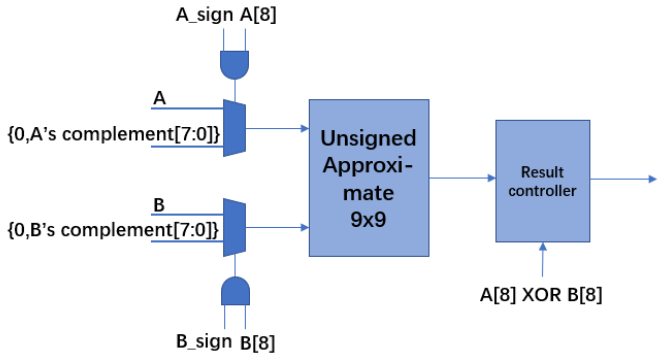

Fig. 4. The proposed 9×9 approximate multiplier

The proposed APIR-DSP is formed using the proposed approximate 9×9 multipliers to replace the exact 9×9 multipliers in the PIR-DSP, which is shown in Fig. 5.

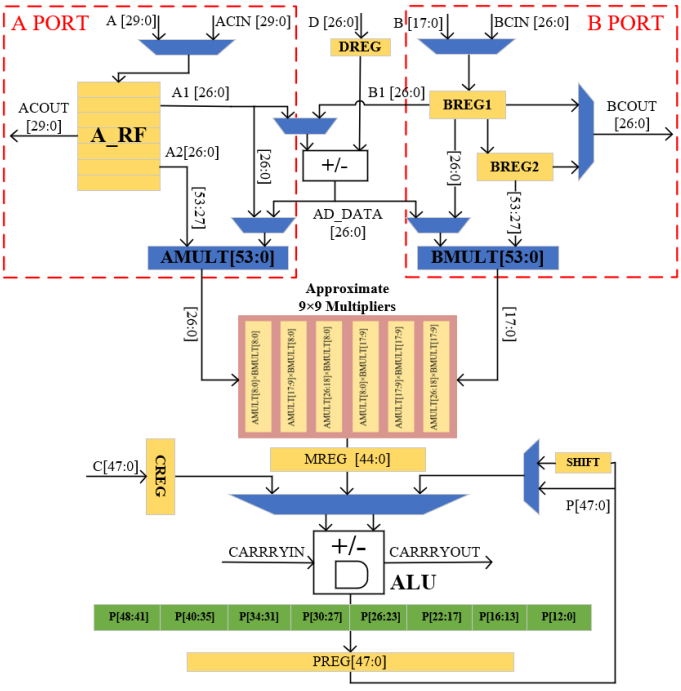

Fig. 5. The APIR-DSP architecture

IV. PROGRAMMABLE DSP BLOCK EVALUATION PLATFORM

In the section, we introduce our programmable DSP block evaluation platform by expanding the front-end synthesis tool Yosys and the architecture description file for back-end PnR tool VPR.

A. Synthesis and Technology Mapping

Our programmable DSP block evaluation platform uses Yosys for synthesis and the embedded ABC [22] for technology mapping. As previously mentioned, Yosys supports some features of the Xilinx Virtex 7 series, such as BRAM, Carry-chain, etc., but lacks support for Ultrascale DSP48E2. Therefore, we first modify the Yosys library files based on the Xilinx 7 series, and then add support for the DSP48E2, PIR-DSP and APIR-DSP blocks.

Yosys's *techmap* command converts the components into the basic logical units in the target architecture. In FPGA CAD flow, the basic logical unit can be a LUT or a special functional unit, such as a multiplier. In order to be compatible with the VPR input format, we have modified the LUT, carry-chain and other units in Yosys FPGA synthesis library.

For the LUT unit, Yosys usually calls ABC to generate LUT2, LUT3 and other fixed subunits. This is not suitable for VPR packing, place and route. Therefore, we add the "*synth_pirdsp*" command in Yosys to support both PIR-DSP and APIR-DSP architectures based on the original "*synth_xilinx*" to generate the basic $lut units and allow VPR to pack them into clusters.

The carry-chain module in Yosys synthesis library lacks the carry output port CO_CHAIN, which does not match the CLB architecture in the VPR description file. To address this problem, we modify the architecture description of carry-chain module, where the most significant bit (MSB) of CO is assigned to output CO_CHAIN, as shown in Fig. 6(b).

Yosys natively supports DSP48E1 block which can convert a $mul into a DSP48E1 block based on *techmap* command. Then it can pack the corresponding $add and other units into the DSP48E1 block. Therefore, it is relatively straightforward to add support for the DSP48E2, PIR-DSP and APIR-DSP blocks to the Yosys synthesis library.

For the DSP48E2, a single $mul unit can be converted into one DSP48E2 block, this being one-to-one mapping flow. However, for PIR-DSP or APIR-DSP, it contains six 9×9 low-precision multiplications, so multiple $mul units must be packed into a single DSP block. In order to perform synthesis for the PIR-DSP/APIR-DSP blocks, we add a pre-pack step based on Yosys's *extract* command which packs the six $mul units into one DSP block. Using the same method, we pack the input registers that meet the condition for being used as port A of the DSP block into a RF. Hence, the synthesis result of Yosys contains DSP and RF blocks, which can be packed by VPR into a PIR-DSP/APIR-DSP block.

As previously mentioned, Yosys can pack the $add unit into a DSP block. The different connections of $add and other units in a DSP block constitute a variety of working modes.

```
module CARRY4(
  output [3:0] CO,
  output [3:0] O,
  input        CI,
  input        CYINIT,
  input  [3:0] DI, S
);
  assign O = S ^ {CO[2:0], CI | CYINIT};
  assign CO[0] = S[0] ? CI | CYINIT : DI[0];
  assign CO[1] = S[1] ? CO[0] : DI[1];
  assign CO[2] = S[2] ? CO[1] : DI[2];
  assign CO[3] = S[3] ? CO[2] : DI[3];
endmodule
```

(a)

```
module CARRY4(
  output CO_CHAIN,
  output [3:0] CO,
  output [3:0] O,
  input        CI,
  input        CYINIT,
  input  [3:0] DI, S
   );

  parameter CYINIT_AX = 1'b0;
  parameter CYINIT_C0 = 1'b0;
  parameter CYINIT_C1 = 1'b0;
  wire CI0;
  assign CI0 = CYINIT_AX ? CYINIT :
               CYINIT_C1 ? 1'b1 :
               CYINIT_C0 ? 1'b0 :
               CI;
  assign CO[0] = S[0] ? CI0 : DI[0];
  assign CO[1] = S[1] ? CO[0] : DI[1];
  assign CO[2] = S[2] ? CO[1] : DI[2];
  assign CO[3] = S[3] ? CO[2] : DI[3];
  assign O = S ^ CO;
  assign CO_CHAIN = CO[3];
endmodule
```

(b)

Fig. 6. Carry-chain description in Yosys synthesis library: (a) before modification,(b) after modification

We write packing architecture Pattern Matcher Generator (PMG) files according to the format specified by Yosys [16], for DSP48E2, PIR-DSP and APIR-DSP respectively. These files describe the connections in the DSP blocks under each working mode. A Python script reads the one PMG file to generate a corresponding header file, which is included by a C++ file to realize the mode matching.

After the mapping for $lut, carry-chain, $mul, $add and RF, the final step is for Yosys to generate a blif file for VPR.

B. Placement and Routing

Our programmable DSP unit evaluation platform uses VPR for placement and routing, and the architecture description file for VPR is based on the Xilinx 7 series. The APIR-DSP block can replace the DSP48E1 in Xilinx 7 series, and analysis performed following placement and routing. The FPGA architecture description files in VPR can be implemented at different levels [23]. If the device model of FPGA is well matched with the data model, a reasonable and accurate FPGA architecture model can be obtained. For example, Murray et al. use VPR to model commercial FPGA Intel Stratix IV [24]. We use the same method and model our target FPGA at the device model level.

In our target FPGA architecture, the APIR-DSP block's interconnection and placement are shown in Fig. 7, where DSP units are arranged in a columnar fashion and are the same height as BRAMs. Each APIR-DSP unit cotains two DSP blocks. In the architecture description file, the height parameter is used to define the height relationship of different units. In this paper, the DSP and BRAM have a height of 5. While BRAM is not the focus of this paper, we note that Yosys treats it as a blackbox, and VPR only considers the communication and interconnection of external ports without any information regarding the specific internal connections.

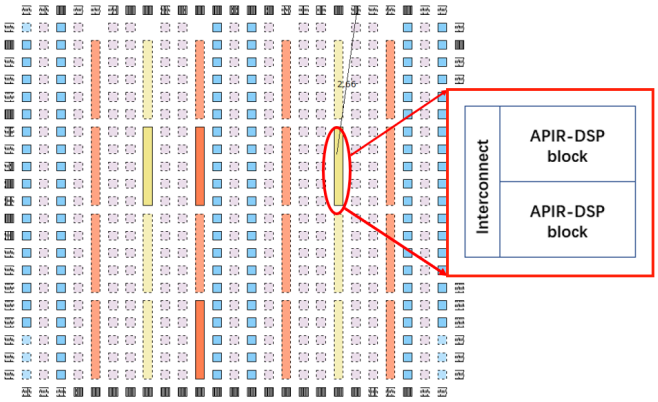

Fig. 7. APIR-DSP interconnection and placement

Compared with DSP48E2, PIR-DSP or APIR-DSP block's interconnection is a semi-2D systolic structure. In this structure, each DSP block connects to non-adjacent DSP blocks in the same column [9]. As shown in Fig. 8, we implement this structure by modifying the "direct" and the parameters "y_offset" in the architecture description file. As previously mentioned, each DSP unit contains two DSP blocks, and the height of DSP unit is 5. Therefore, we set the "y_offset" equals to -5, which indicates that the DSP block in current DSP unit connects to the DSP block in previous DSP unit, instead of the DSP block in the same DSP unit.

As previously mentioned, we use VPR to pack the RF of port A into APIR-DSP. We use "pack_pattern" in the architecture description file to guide the VPR.

In addition to DSP and BRAM units, the synthesis results of Yosys also contain LUTs and flip-flops and other units, these resources being provided by the configurable logic block (CLB). We describe the detailed architecture and internal connection of CLB unit in the architecture description file. For the LUT unit, the basic LUT unit generated by Yosys is represented by an instance marked with ".names".

For flip-flops, there are different units in the Yosys standard library. Examples include $dffe with enable signal and $adff with synchronous reset signal. Yosys converts these units into the register units of the target FPGA. In our flow the register units are FDRE, FDSE, etc. Therefore, we provide blif modules for each register unit (i.e. FDRE, FDSE, etc.) in the architecture description file to indicate that it is the basic unit, and VPR will pack it into a CLB block.

Finally, in the layout section of the architecture description file in VPR, we define the same cell arrangement structure and

```
<direct from_pin="DSP.ACOUT0" name="DSP_ACAS0" switch_name="routing_4" to_pin="DSP.ACIN0" x_offset="0" y_offset="-5" z_offset="0"/>
<direct from_pin="DSP.BCOUT0" name="DSP_BCAS0" switch_name="routing_4" to_pin="DSP.BCIN0" x_offset="0" y_offset="-5" z_offset="0"/>
<direct from_pin="DSP.PCOUT0" name="DSP_PCAS0" switch_name="routing_4" to_pin="DSP.PCIN0" x_offset="0" y_offset="-5" z_offset="0"/>

<direct from_pin="DSP.ACOUT1" name="DSP_ACAS1" switch_name="routing_4" to_pin="DSP.ACIN1" x_offset="0" y_offset="-5" z_offset="0"/>
<direct from_pin="DSP.BCOUT1" name="DSP_BCAS1" switch_name="routing_4" to_pin="DSP.BCIN1" x_offset="0" y_offset="-5" z_offset="0"/>
<direct from_pin="DSP.PCOUT1" name="DSP_PCAS1" switch_name="routing_4" to_pin="DSP.PCIN1" x_offset="0" y_offset="-5" z_offset="0"/>
```

Fig. 8. Description of the direct connections for APIR-DSP architecture

TABLE III
COMPARISON OF THE SNR OF FIR FILTER

Muliplier Type	SNR (dB)
Exact(baseline)	33.80
Approximate	32.77

that the FPGA automatically expands the grid size (i.e. setting "auto_layout") with an aspect ratio of 0.6 during placement and routing.

V. EXPERIMENTAL RESULTS

In this section, we evaluate the proposed approximate 9×9 multiplier in an FIR filter and DNN applications. We also synthesise hard blocks for PIR-DSP and APIR-DSP and compare their area, power consumption and critical path delay. Finally, using two low-precision test cases on the programmable DSP unit evaluation platform, we compare the performance of DSP48E2 and APIR-DSP.

A. Result of approximate 9×9 multiplier

We apply the proposed approximate 9×9 multiplier and the exact 9×9 multiplier to a 20 order adaptive low-pass FIR filter and compare their maximum output signal to noise ratio (SNRs) to evaluate the accuracy loss introduced by the approximate multiplier. As shown in TABLE III, compared with exact FIR filter, the approximate FIR filter's SNR is decreased 1.03 dB.

We also use a TensorFlow DNN platform to study DNN top-1 accuracy loss (DAL) caused by our approximation. During the training stage, the fake-quantization method [25] is used to simulate the quantization effect with floating-point operations to reduce the accuracy loss caused by the quantization error. Fig. 9 shows the structure of the convolutional layer, where weights are in the form of floating-point numbers. Hence, in order to use the proposed approximate multiplier without increasing the quantization complexity, we quantify the weights to 8-bit, corresponding to the unsigned 8×8 multiplication, which is the subset of the approximate multiplier operations. Finally, the fake-quantization method is applied to the output of the ReLU6 function to obtain the output.

We compare an unsigned 8×8 exact multiplier to our proposed approximate 9×9 multiplier by introducing the multiplier approximation in look up table (LUT) form to the platform which was subsequently used to train the DNN on the MNIST and CIFAR10 datasets.

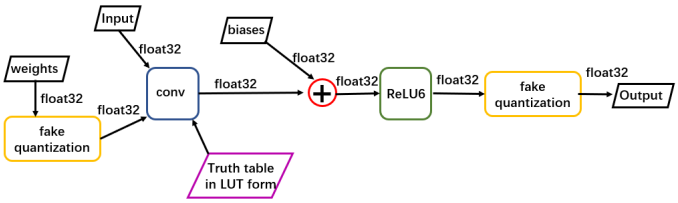

Fig. 9. A convolutional layer in training with fake-quantization method

TABLE IV
COMPARISON OF DAL WITH THE PROPOSED MULTIPLIER

Multiplier Type	MNIST	CIFAR10
	LeNet	AlexNet
Exact(baseline)	99.32%	88.48%
Approximate	99.32%	88.17%
DAL	0	0.31%

As shown in TABLE IV, when computing unsigned 8×8 multiplications, compared with the baseline exact multiplier, the proposed approximate multiplier achieves a very small loss of accuracy. In addition, the DNN is able to compensate for the approximation of the multipliers by adjusting its weights.

In addition, we use the MED and NMED expressions of (7) and (8), respectively, to further illustrate the accuracy of the proposed approximate multiplier. Since the error distribution of our proposed approximate 9×9 multiplier is not uniform, small operand has small error. In these specific applications where parameters do not tend to be very large, hence our approximate multiplier has a low accuracy loss.

TABLE V shows these two error metric values of the proposed approximate multiplier in specific applications, where the MEDs and NMEDs are very small.

$$NMED = \frac{MED}{(2^n - 1)^2} \tag{8}$$

B. DSP Performance

PIR-DSP and APIR-DSP were implemented in Verilog and synthesized using Synopsys Design Compiler (DC) with

TABLE V
THE MED AND NMED RESULTS

Applications	MED	NMED($\times 10^{-6}$)
FIR filter	0.07	1.12
LeNet	5.54	85.24
AlexNet	0.36	5.59

TABLE VI
COMPARISON OF APIR-DSP AND PIR-DSP IN TERMS OF AREA, POWER AND DELAY

DSP Type	*Area(μm^2)*	*Power(mW)*	*Delay(ps)*
PIR-DSP	1280.09	181.14	4250
APIR-DSP	1003.68	176.06	4044
Improvement	21.60%	2.80%	4.85%

TABLE VII
COMPARISON EVALUATION RESULTS OF DSP BY GEMMS

DSP Type	Size	DSP Usage	BRAM Usage	CLB Usage	Delay(ns)
DSP48E2	70×117	65	269	986	9.22
APIR-DSP	70×117	33	270	1089	8.49
Improvement	0	49.2%	-0.3%	-9.4%	7.9%

ASAP-7nm process library [21]. From the Synopsys DC tool reports, the area, power consumption and critical path delay information can be extracted, and this is summarized in TABLE VI. Compared with PIR-DSP, APIR-DSP's area is decreased by 21.60%, critical path delay decreased 2.80% and power consumption decreased 4.85%. Thus the APIR-DSP simultaneously improves performance for these three metrics.

C. Evaluation Platform Test

We use general matrix multiplication (GEMM) in DeepBench [26] and low-precision neural networks DoReFaNet [27] as Verilog test cases to evaluate the performance of DSP48E2 and APIR-DSP. The results after VPR placement and routing form the basis for our evaluation. Timing information of CLB, BRAM and other units in the architecture file are provided by Symibiflow/prjxray-db [28], and the timing information of DSP blocks is obtained by DC synthesis.

In the GEMM case, as shown in TABLE VII, using APIR-DSP, the size of FPGA (i.e. the array size to implement the given application) does not change, and the usage of DSP blocks is decreased from 65 to 33, this being because APIR-DSP supports more multiply accumulate operations within a single DSP. However, in order to route the APIR-DSP input data, CLB usage is increased. The critical path delay is also decreased from 9.22 ns to 8.49 ns.

In the DoReFaNet case, as shown in TABLE VIII, using APIR-DSP, the size of the FPGA array is significantly decreased from 250×417 to 207×345. The usage of DSP blocks is decreased from 216 to 72 and CLBs are decreased from 74482 to 50931. The critical path is in the address generation (when DNN weights are read from BRAM), and both have the same value of 9.84 ns.

These two examples show that the APIR-DSP can significantly reduce the resource usage and increase the performance compared with DSP48E2.

VI. CONCLUSION AND FUTURE WORK

In this paper, we propose APIR-DSP, an approximate PIR-DSP architecture with low accuracy loss. Compared with PIR-DSP, by using approximate 9×9 multiplier, APIR-DSP's area, power consumption and critical path delay are decreased by 21.60%, 2.80% and 4.85%, respectively. Based on Yosys and VPR, we have developed a programmable DSP unit evaluation platform, and used DeepBench and the DoReFaNet low-precision neural network to compare the Xilinx DSP48E2 with APIR-DSP. We conclude that APIR-DSP can significantly reduce number of DSP blocks and improve hardware utilization.

TABLE VIII
COMPARISON EVALUATION RESULTS OF DSP BY DOREFANET

DSP Type	Size	DSP Usage	BRAM Usage	CLB Usage	Delay(ns)
DSP48E2	250×417	216	12	74482	9.84
APIR-DSP	207×345	72	12	50931	9.84
Improvement	31.5%	66.7%	0	31.6%	0%

In future work, we can further partition the approximate 9×9 multiplier into 4×4 or 2×2 approximate multipliers so that APIR-DSP can support a wider range of low precision MAC operations, improving the performance of APIR-DSP. In addition, we intend to refine compatibility between Yosys and VPR, and include support for other DSP blocks.

ACKNOWLEDGEMENTS

This work is supported by the National Natural Science Foundation of China under grant 61971143.

REFERENCES

[1] Krizhevsky A, Sutskever I, Hinton G. *ImageNet Classification with Deep Convolutional Neural Networks*. NIPS. Curran Associates Inc. 2012.
[2] He K, Zhang X, Ren S, et al., "Deep Residual Learning for Image Recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770-778.
[3] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, et al., "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference," in *Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2017, pp. 65-74.
[4] A. Prost-Boucle, A. Bourge, F. Pétrot, et al.,"Scalable high-performance architecture for Convolutional ternary Neural Networks on FPGA," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1-7.
[5] Xilinx Inc. "WP486-Deep Learning with INT8 Optimization on Xilinx Devices," 2017.
[6] A. Boutros and V. Betz, "FPGA Architecture: Principles and Progression," in *IEEE Circuits and Systems Magazine*, 2021, vol. 21, no. 2, pp. 4-29.
[7] Intel Corp. "Intel Stratix 10 NX FPGA: AI-optimized FPGA for high-bandwidth, low-latency AI acceleration," 2020.
[8] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *18th IEEE European Test Symposium*, Avignon, France, 2013, pp. 1-6.
[9] S. Rasoulinezhad, H. Zhou, L. Wang and P. H. W. Leong, "PIR-DSP: An FPGA DSP Block Architecture for Multi-precision Deep Neural Networks," in *IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019, pp. 35-44.
[10] Xilinx Inc. "UG579: UltraScale Architecture DSP Slice," 2018.
[11] Intel Corp. "UG-S10-DSP Intel Stratix 10 Variable Precision DSP Blocks User Guide," 2018.
[12] Warrier R, Shreejith S, Zhang W, et al., "Fracturable DSP Block for Multi-context Reconfigurable Architectures," in *Circuits Systems and Signal Processing*, 2016, 36(7).
[13] P. Colangelo, N. Nasiri, E. Nurvitadhi, et al., "Exploration of Low Numeric Precision Deep Learning Inference Using Intel® FPGAs," in *IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2018, pp. 73-80.

[14] H. Parandeh-Afshar and P. Ienne, "Highly Versatile DSP Blocks for Improved FPGA Arithmetic Performance," in *18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2010, pp. 229-236.
[15] A. Boutros, S. Yazdanshenas and V. Betz, "Embracing Diversity: Enhanced DSP Blocks for Low-Precision Deep Learning on FPGAs," in *28th International Conference on Field Programmable Logic and Applications (FPL)* , 2018, pp. 35-42.
[16] C. Wolf, Yosys Open Synthesis Suite. *http://www.clifford.at/yosys/*.
[17] Murray K E, Petelin O, Zhong S, et al., "VTR 8: High-performance CAD and Customizable FPGA Architecture Modelling," in *ACM Transactions on Reconfigurable Technology and Systems*, 2020, vol. 13, no. 2, pp. 1-55.
[18] Nextpnr. *https://github.com/YosysHQ/nextpnr.*
[19] M. Sjalander and P. Larsson-Edefors, "Multiplication Acceleration Through Twin Precision," in *IEEE Transactions on Very Large Scale Integration Systems*, 2009, vol. 17, no. 9, pp. 1233-1246.
[20] I. Scarabottolo, G. Ansaloni, G. A. Constantinides, et al.,"Approximate Logic Synthesis: A Survey," in *Proceedings of the IEEE*, 2020, vol. 108, no. 12, pp. 2195-2213.
[21] Clark, L. T., Vashishtha, V., Shifren, L., et al., "ASAP7: A 7-nm finFET predictive process design kit," in *Microelectronics Journal*, 2016, vol. 53, pp. 105-115.
[22] Berkeley logic synthesis and verification group, "Abc: a system for sequential synthesis and verification." *https://people.eecs.berkeley.edu/alanmi/abc/.*
[23] K. E. Murray, M. A. Elgammal, V. Betz, T. Ansell, K. Rothman and A. Comodi, "SymbiFlow and VPR: An Open-Source Design Flow for Commercial and Novel FPGAs," in *IEEE Micro*, 2020, vol. 40, no. 4, pp. 49-57.
[24] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz, "Timing-driven Titan: Enabling large benchmarks and exploring the gap between academic and commercial CAD," in *ACM Transactions on Reconfigurable Technology and Systems*, 2015, vol. 8, no. 2 p. 1-18.
[25] B. Jacob et al., "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition)*, 2018, pp. 2704-2713.
[26] HLS4ML. *https://github.com/raminrasoulinezhad/DeepBenchVerilog.*
[27] L. Jiao, C. Luo, W. Cao, X. Zhou and L. Wang, "Accelerating low bit-width Convolutional Neural Networks with embedded FPGA," in *27th International Conference on Field Programmable Logic and Applications (FPL)* , 2017, pp. 1-4.
[28] Project X-Ray. *https://github.com/SymbiFlow/prjxray−db/blob/artix7/timings/.*