International Journal of Engineering Sciences & Research Technology

(A Peer Reviewed Online Journal)
Impact Factor: 5.164

IJESRT

RESEARCHERID
THOMSON REUTERS

[Narayanam* *et al.*, 8(2): February, 2019]
IC™ Value: 3.00

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

IJESRT

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

FPGA IMPLEMENTATION OF NOVEL ARCHITECTURES FOR DIGITAL IMAGE PROCESSING CONVOLUTION FILTER: DEVELOPMENT OF NOVEL QUARTILE DIVISION ARCHITECTURES

Ranganadh Narayanam
Assistant Professor, ECE department, Faculty of Science and Technology, The ICFAI Foundation for Higher Education, Hyderabad, (Deemed to be University under section 3 of UGC Act, 1956), India

DOI: 10.5281/zenodo.2578164

ABSTRACT

Image convolution is a very important operation in signal and image processing. It is general purpose filter effect and an algorithm used to filter images. In this the mathematical concepts of convolution and the kernel matrix are used to apply filters to images, to perform functions such as blurring, sharpening, embossing, extracting edges and reducing unwanted noise. Convolution could operate in 1D (e.g. speech processing), 2D (e.g. image processing) or 3D (video processing). Image Convolution finds applications in the specialized areas of Digital Image Processing such as Computed tomography, Image recognition, Imaging Spectroscopy, Artificial Intelligence, feature extraction etc. In this research I have developed 8 different Novel hardware architectures of my own for Convolution Process, which are nowhere in literature before, and first of their kind. All these architectures are designed and implemented first as a prototypic 16 by 16 image and 3 by 3 integer kernels; 32 by 32 image and compared. The programming is done using Verilog HDL and implementation is done on Xilinx Artix-7 FPGAs using Vivado 2015.2 Tool.

1. INTRODUCTION

In Digital Image Processing, the spatial domain processing is a visually rich area of study deals with pixel manipulation techniques. Different operations can be performed over images in this spatial domain. All these matrix-based operations are performed between a larger matrix, which represents an image, and a smaller matrix, known as 2D kernel. The size and associated values in the kernel determine the impact exerted by it on the image considered [2]. The convolution process is one of the spatial domain pixel manipulation technique which is done between an image and a 2D kernel of different sizes such as 3 by 3 or 5 by 5, and the result can be edge detected image, enhanced image, de-noised image, blurred image etc basing on the values in the kernel [2]. There are certain standard kernels are there for each different effect on the image. Basing on the level of enhancement, edge detection, blurring and basing on the application requirement the values in the kernel can be any values and the 2D kernel can be customized, and sometimes it can be integer kernel and sometimes it can be fractional numbers.

Some very important applications of Convolution in Digital Image Processing

Convolution in 2D spatial which is mostly used in image processing for feature extraction and is also the core block of Convolution Neural Networks (CNNs). Convolution is the first step to understand Convolution Neural Networks.

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

Convolution process

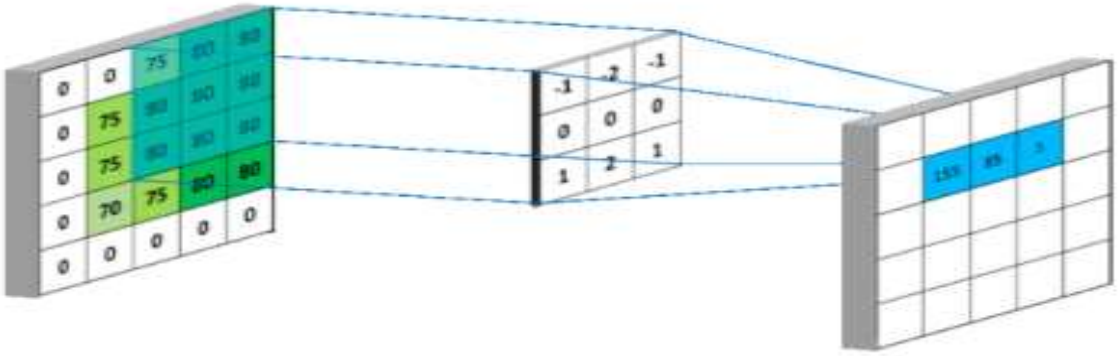


Figure 1: The first big matrix is the original image matrix, middle matrix is an example 2D kernel, and the output is the resulting image after convolution. In this process the image shrinks as the border vertical columns and border horizontal rows of the image are not involved in convolution process.

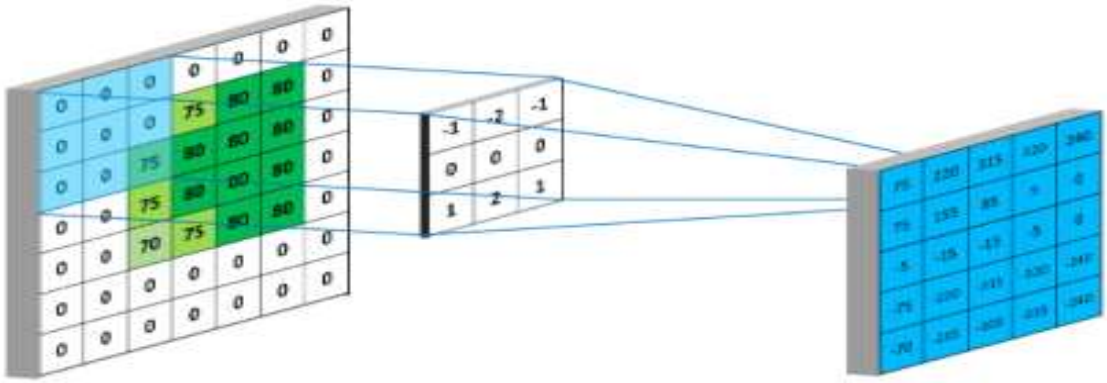


Figure 2: By padding "zeros" at the borders as in this figure, the image does not shrink.

In the above figures kernel is the centre small matrix, image is the first image and the output convolved image is the output image. At the top left corner of the image kernel is placed, and the corresponding pixels of the two are multiplied and all products are summated, the result is placed in the new image at the point corresponding to the centre of the kernel. An example for this first step is shown in the Figure 1. The kernel is moved over by one pixel and this process is repeated until all of the possible locations in the image are filtered. Notice that there is a border of empty values around the convolved image – image shrinks. This is because the result of convolution is placed at the centre of the kernel. To deal with this, a process called 'padding' or more commonly 'zero-padding' is used – Figure 2. This simply means that a border of zeros is placed around the original image to make it a pixel wider all around. The convolution is then done as normal, but the convolution result will now produce an image that is of equal size to the original.
As in the Figure 1, the convolution output 85 is obtained as follows
(0*-1+75*-2+80*-1) + (75*0+80*0+80*0) + (75*1+80*2+80*1) = 85 (1)

Motivation behind this design

a) In my novel work on various Digital Image Processing filters, along with Translation Invariance (TI) algorithm for filtering process [1], I found Convolution filter is found to be efficiently working in high noise conditions, than remaining filters [1]. The results are tabulated in [1].
b) Convolution networks are finding plenty of applications in Artificial Intelligence such as Neuromorphic computing and Deep Learning.
c) In most recent times, Convolution is having advanced applications in vision, image classification, retinal implants including artificial retinal implants etc.

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

Keeping in mind of this advanced role of Convolution, I am highly motivated towards doing research in hardware design of Convolution Filter. I developed 8 different Novel architectures for this convolution filtering, which are nowhere before in the literature. In all these architectures I have taken integer kernel. This design is useful directly in any integer kernel based convolution and also as core part in the fraction kernel convolution also (if the common fraction denominator part is taken out of the kernel then everything is integer part of the kernel). In this research I developed and implemented these designs on to Xilinx Artix-7 FPGAs and compared their performance in terms of speed, power, path delay, Hardware utilization.

The organization of the paper is this way: Section 2 deals with design methodology of the novel architectures, section 3 deals with Quartile Division Architectures, section 4 implementation results, section 5 deals with conclusions and future research.

2. NOVEL ARCHITECTURES

2.1. Convolution Result Generator (CRG)

In this research I have developed 8 different novel hardware architectures for implementing the Convolution filter. The very important arithmetic block in all these architectures is Convolution Result Generator (CRG). The critical units in this CRG are

a) Multiplication unit
b) Addition unit

The multiplication and addition are basic building blocks, which are the important units in the Convolution Result Generator. This CRG is the critical arithmetic building block in all these developed architectures. The Convolution Result is generated as in the given equation (i) in the section 1.

When it comes to hardware:

The image pixels are loaded in a contiguous memory in the order of first row, and then second row, third row and so on of the entire image. I have taken 3 by 3 kernel based convolution. So, the kernel is loaded in one more contiguous memory of 9 locations. For each convolution result the corresponding 9 image data points are accessed simultaneously from the corresponding memory locations, and the 9 kernel points are accessed simultaneously and the data and corresponding kernel points are multiplied. All the 9 multiplication operations are done concurrently and then added to get the convolution result. The hardware architecture is designed as a tree structure as given in the following hardware diagram.

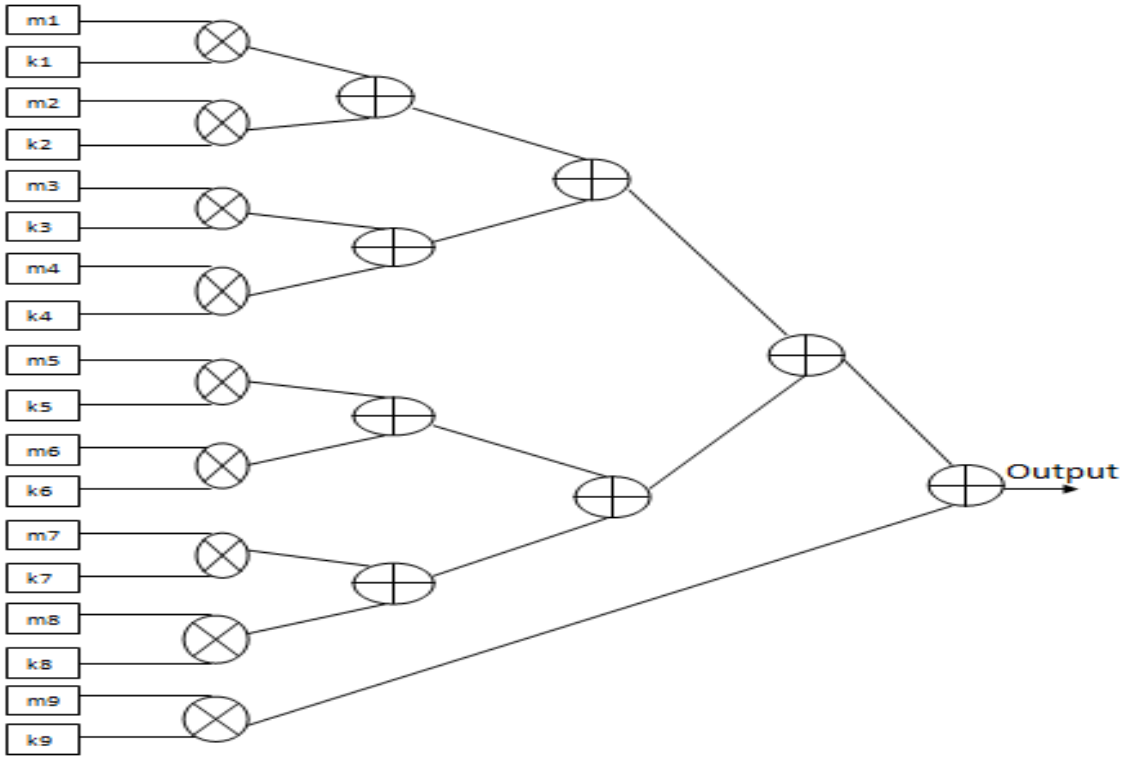
m1	m2	m3
m4	m5	m6
m7	m8	m9

(a) Original Image part

k1	k2	k3
k4	k5	k6
k7	k8	k9

(b) 2D Kernel

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7



(c) Tree Structure for Convolution Result Generator (CRG)

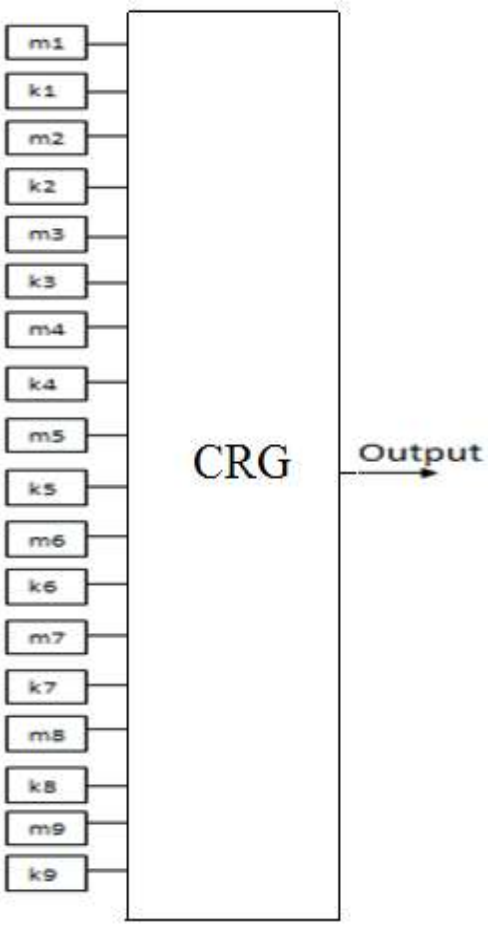
Figure 3: Convolution Result Generator Tree structured architecture

The convolution output

Output = m1*k1 + m2*k2 + m3*k3 + m4*k4 + m5*k5 + m6*k6 + m7*k7 + m8 *k8 + m9*k9 (2)

The entire image (pixels) coming from external world is loaded in a contiguous memory row by row. *Figure 3 (a)*, The 9 image pixels of the corresponding image matrix part which is under the view of convolution are accessed from the corresponding locations simultaneously from the contiguous memory, and similarly the convolution kernel points from its memory *Figure 3 (b)*. The values are specified in the *Figure 3 (c)* at the input of the Convolution Result Generator (CRG). Then the first stage of the CRG contains 9 multiplication units to finish the multiplication process in parallel. Then the outputs are to be added, for which it needs 9 numbers to be added, and 8 addition units required. The second stage of the CRG has 4 adder units to finish up the 4 additions in parallel, then third stage has 2 adders and then 4th stage and final stage have one adder each to finish up the all additions and the convolution result is available at the output of the CRG. This CRG is the most critical part of all of the 8 architectures.

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7



Convolution Result Generator (CRG) block diagram – 18 inputs by 1 output

Figure 4: CRG block diagram

The following are my 8 Novel architectures. I have implemented all these for 32 by 32 and 16 by 16 size images. Here I am presenting all these architectures based on 16 × 16 size image, and 3 × 3 2D integer kernel. This implementation I have done as a prototypic design to evaluate the performances of these architectures using 16 and 32 size images. It can be extended these architectures to any size such as 256 × 256 or 1024 × 1024 size images. In these architectures all the image pixels from external world are being received serially one after the other by our convolution system from a transmitter.

2.2 Architecture 1: All concurrent CRGs

The block diagram is given in the Figure 5.

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

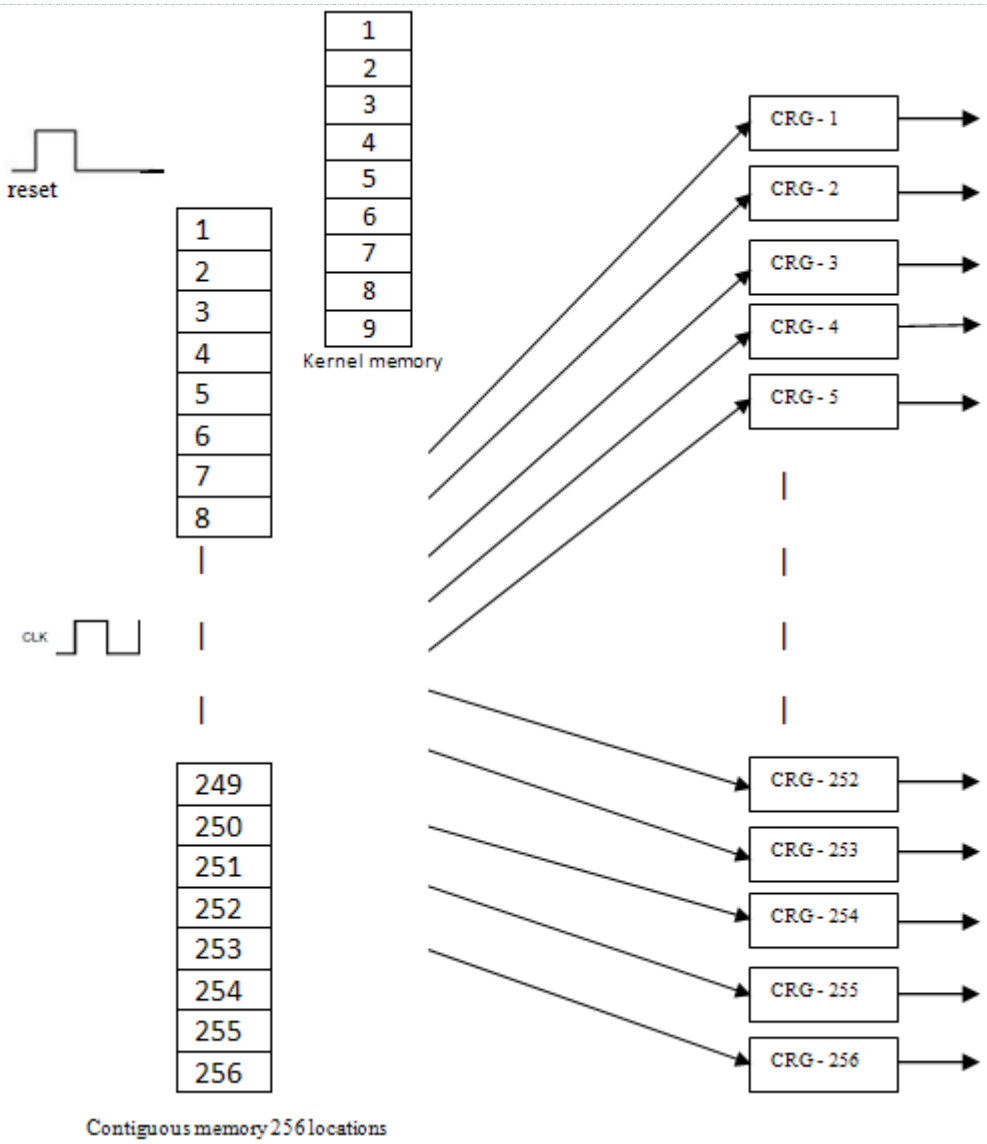


Image pixels memory	data from corresponding pixel locations from memory	256 concurrent CRG units	256 concurrent convolution results

Figure 5: Architecture – 1.

In this architecture, there is a contiguous memory of 256 locations, and one more memory of 9 locations for kernel matrix. At reset all the 256 size memory is cleared to zero; and the kernel memory is loaded with kernel values row by row. Consideration is that all the pixels of the 16 × 16 image are received to the system serially from external world one pixel after the other and in the order of row after row. There is an enable signal generated by an external controller basing on which a counter inside this system starts counting and loads all the 256 pixels into the contiguous memory. Once all the pixels are loaded, all the corresponding pixels inputs from the image pixels memory, and the kernel values from the kernel memory are simultaneously available at the inputs of all the CRGs, then an enable signal is generated which starts the executing the multiplication stage, stage -1, of all the CRGs concurrently and then all the addition stages also finished concurrently and all the convolution results are available simultaneously at a time. There is contiguous array memory available for storing all the convolution output results.

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

2.3 Architecture 2: State machine based row by row architecture

Here I am using 16 × 16 size image. So, I am using the concurrent architecture approach to each row separately. So, for the entire image a single set of 16 concurrent CRGs used, one row at a time, row by row, instead of entire concurrency as in Architecture – 1, so that can save hardware resources at the expense of more delay for all the final results are available when compared to Architecture - 1. For this I designed a state machine, which has 16 states.

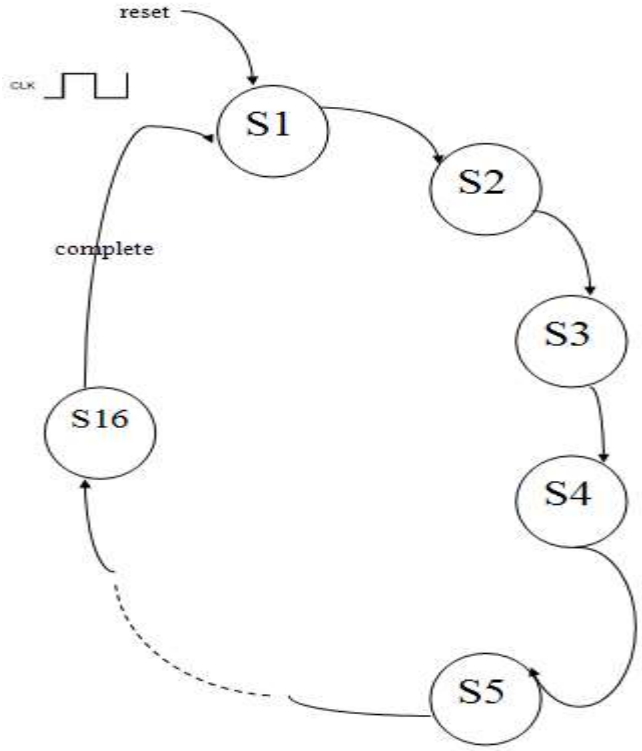


Figure 6: 16 states State Machine; each state generates 16 concurrent convolution results using 16 CRGs

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

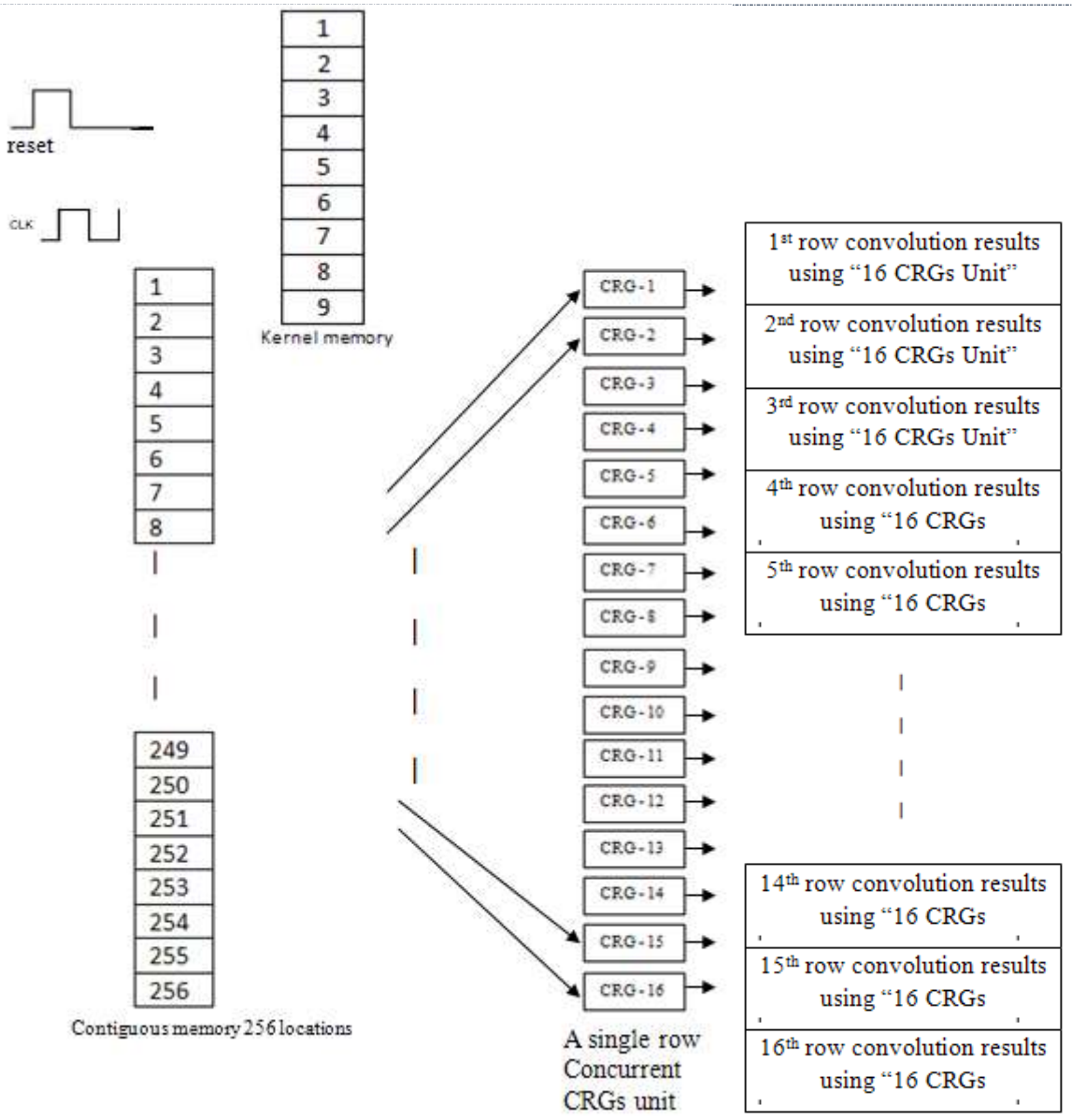


Image pixels memory uses	data from	a single unit of	each row of image matrix
generate	corresponding	16 concurrent	this single unit to
convolution results	pixel locations	CRGs	that row's
state machine	from memory		using the 16 states

Figure 7: Single unit which contains the 16 concurrently executing CRGs works for each row of the 16 by 16 image using 16 states state machine, one state per each row

As per the figure "Figure: 7", this architecture has a single unit of 16 CRGs which takes care of concurrent generation of convolution coefficients for a single row of the image matrix at a time. There is a contiguous memory of 256 locations loaded with zeros at reset, and a single contiguous memory of 9 memory locations

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

loaded with Kernel values on reset. When there is an external controller sends the signal for loading the image pixels one by one into the memory locations, the externally coming image pixel data is loaded at each positive edge of the internal system clock, into the memory pixel by pixel, row after row. Each CRG takes the corresponding pixels as inputs from the contiguous memory and kernel memory and when the enable signal is generated all CRGs start executing, and generate the convolution results concurrently. The same 16 CRGs unit is used to generate all convolution results of the whole image, a single row at a time. This process is controlled by using a State machine approach. Here there is a state machine which stays in the state S1 on reset. When there is a start signal is generated, then the CRG unit starts executing and consequently generates the convolution results of the first row concurrently. Then when a complete signal is generated then the state machine goes into state S2, then there the same 16 CRGs unit works for generating second row convolution results. Then this process goes on from state S1 through state S16 and finishes all 16 rows convolution results. Then the state machine goes back to the state S1. There is contiguous array memory available for storing all the convolution output results in order.

2.4 Architecture – 3: State machine based row by row architecture, with CLA adder instead of RCA for addition unit in each CRG.
This architecture is same as Architecture – 2, but only the difference is adder units in CRGs. The adder units in all the architectures are ripple carry adders (RCAs). But in the case of this Architecture-3, Carry Look-ahead Adders (CLAs) are used instead of RCAs, which speeds up the process of generating convolution results, and all the 256 convolution results are generated much earlier than in the case of Architecture-2. There is contiguous array memory available for storing all the convolution output results in order.

2.5 Architecture – 4: All sequential convolution results architecture
In this architecture single CRG is used to generate all 256 convolution results. Initially at reset all the 256 locations of the contiguous memory is loaded with zeros, and the kernel memory is loaded with kernel values. The data of the digital image is being sent from external world serially one pixel after the other, row after row in that order, to this convolution system. When an external controller generates a signal to take the data of the image to this convolution system by generating a start control signal, then the image is loaded into the contiguous memory and when all 256 pixels are loaded, then the internal controller gives a start signal to the CRG unit and a state machine which consists of 256 different states. At reset the state machine will be in first state S1 and then the CRG starts executing to generate first row first convolution result, by taking the corresponding input image data from the contiguous memory and kernel memory and at the end of the process the CRG generates complete signal indicating the completion of the Convolution result generation process. Then the state machine goes into second state S2 and the same single CRG starts executing to generate first row second convolution result, by taking the corresponding input image data from the contiguous memory and kernel memory and at the end of the process the CRG generates complete signal indicating the completion of the Convolution result generation process. Then the state machine goes into second state S3 and this process continues until all the 256 convolution results are generated and at the end the state machine goes into state S1. This way all the convolution results are generated. Figure 8 & 9 describe all this.

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

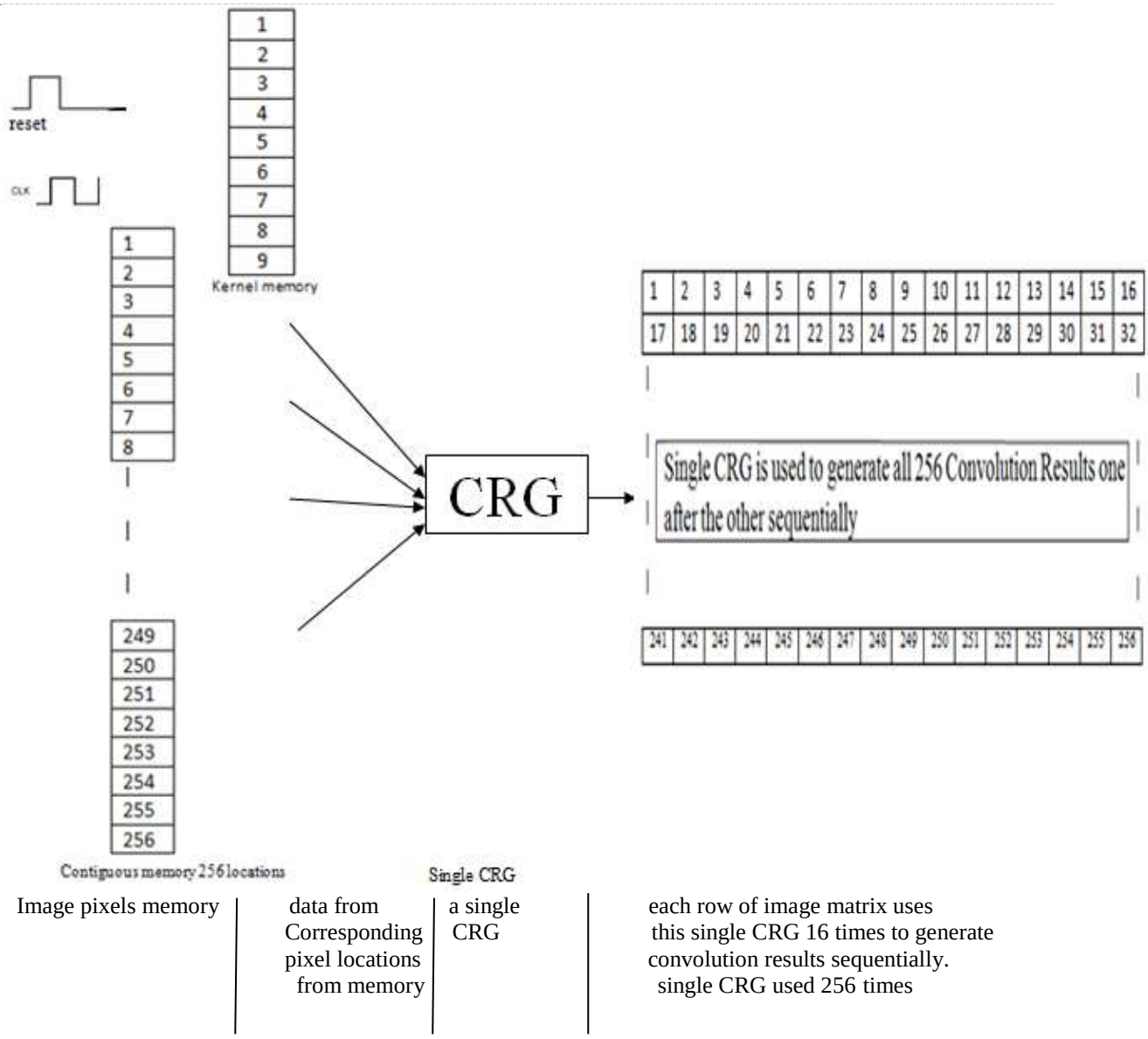


Figure 8: In this a single CRG is used 256 times to generate all the convolution results

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

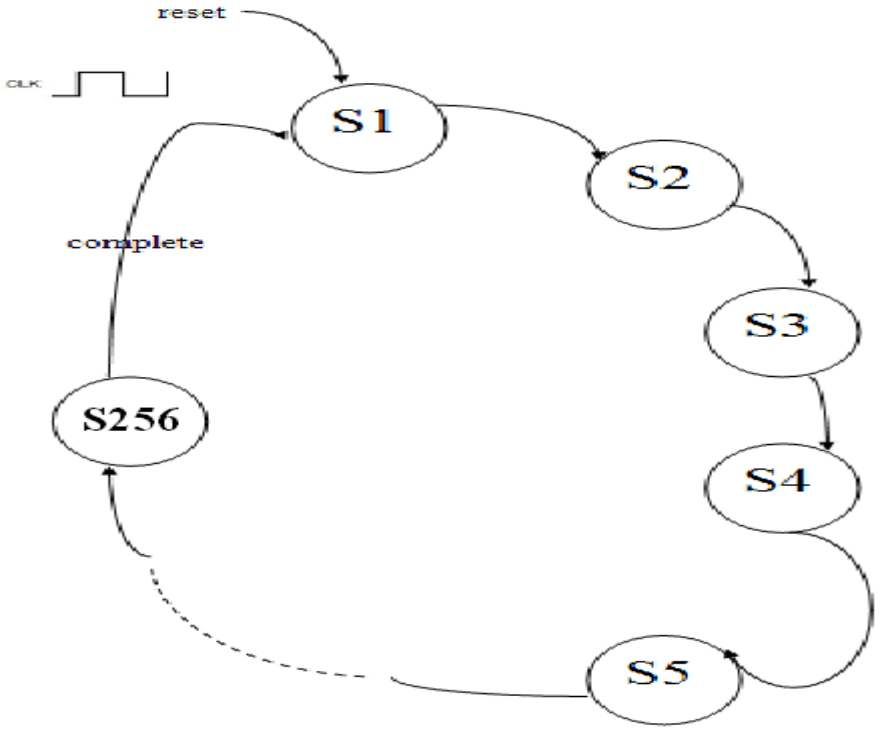


Figure 9: 256 states State Machine, each state generates one convolution results using single CRG

There is contiguous array memory available for saving all the convolution output results in order.

3. QUARTILE DIVISION ARCHITECTURES

To speed up the convolution process of the Architectures – 2, 3, 4 I have developed a process called quartile division. Using this process I have divided the whole image matrix into 4 equal parts, each one a quadrant. In this architecture I have taken the image of 16 × 16 size, where 256 pixels are there. So, if I divide the image into 4 quarters, each quadrant contains 8 × 8 matrix. A single contiguous 256 locations memory is there for loading entire image pixels, one more 9 locations contiguous memory is there for loading kernel values, and within each quadrant an individual array memory will be there to save the convolution result values in order. Here I developed 4 different Quartile Division Architectures.

3.1 Architecture – 5: Quartile Division Architecture – 1

In this quartile division architecture, for each quadrant ie., for each 8 × 8 matrix image pixel data, I applied the FSM based architectural approach as in Architecture – 2. On reset the 256 locations contiguous memory is loaded with zeros, and kernel memory is loaded with kernel values, and the state machine will be in state S1. Figure 10 & 11 represents this architecture.

Here in this architecture each quadrant contains the following components

a) 8 states state machine
b) A single unit containing 8 different CRGs, to generate 8 convolution coefficients concurrently for a single row. This unit called 8 times, once for each row convolution results.
c) An array memory to save all the convolution results in order (for the imaginary quadrant of 8 rows and 8 columns of 8 × 8 image pixels of the original image matrix)

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

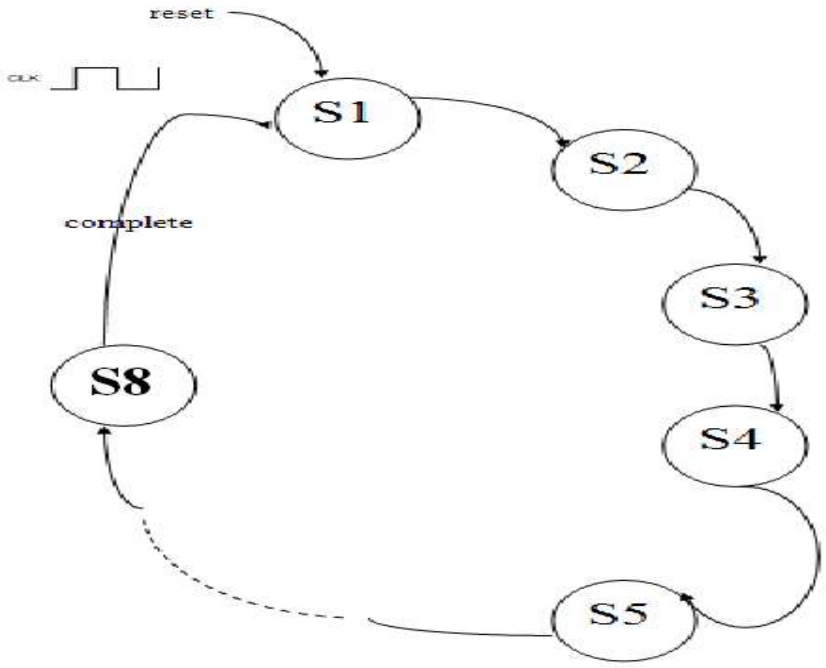


Figure 10: States State Machine, each state generates 8 convolution results using a single unit of 8 CRGs

When an external controller indicates the convolution system to start, internal controller generates a start signal, then the memory gets start loading the 256 memory locations with externally coming serial data of image pixels one after the other. This 256 locations memory and kernel memory is common for all 4 quadrants. When a start convolution signal is generated by the internal controller, the unit of 8 CRGs starts executing to generate the first row 8 convolution results concurrently by taking the corresponding image pixels and kernel values as inputs, like this all the 4 units of 8 CRGs in all the 4 quadrants start executing in parallel to generate the first row 8 convolution results. When a complete signal is generated by the internal controller indicating the completion of generation of convolution results for the first row in each quadrant, then all the 4 state machines go into second state 2 – S2 in parallel. Then all the 4 units of 8 CRGs each, in each quadrant start executing by taking corresponding image pixels and kernel values as inputs and to generate next row convolution results in each quadrant in parallel. When it finishes the state machines go into next state S3 and so on and this process continues until all 8 rows of convolution results of each quadrant are generated, and finally all the state machines go back into the state S1. In this way it can be achieved a speed of 2 times when compared to

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

Architecture – 2

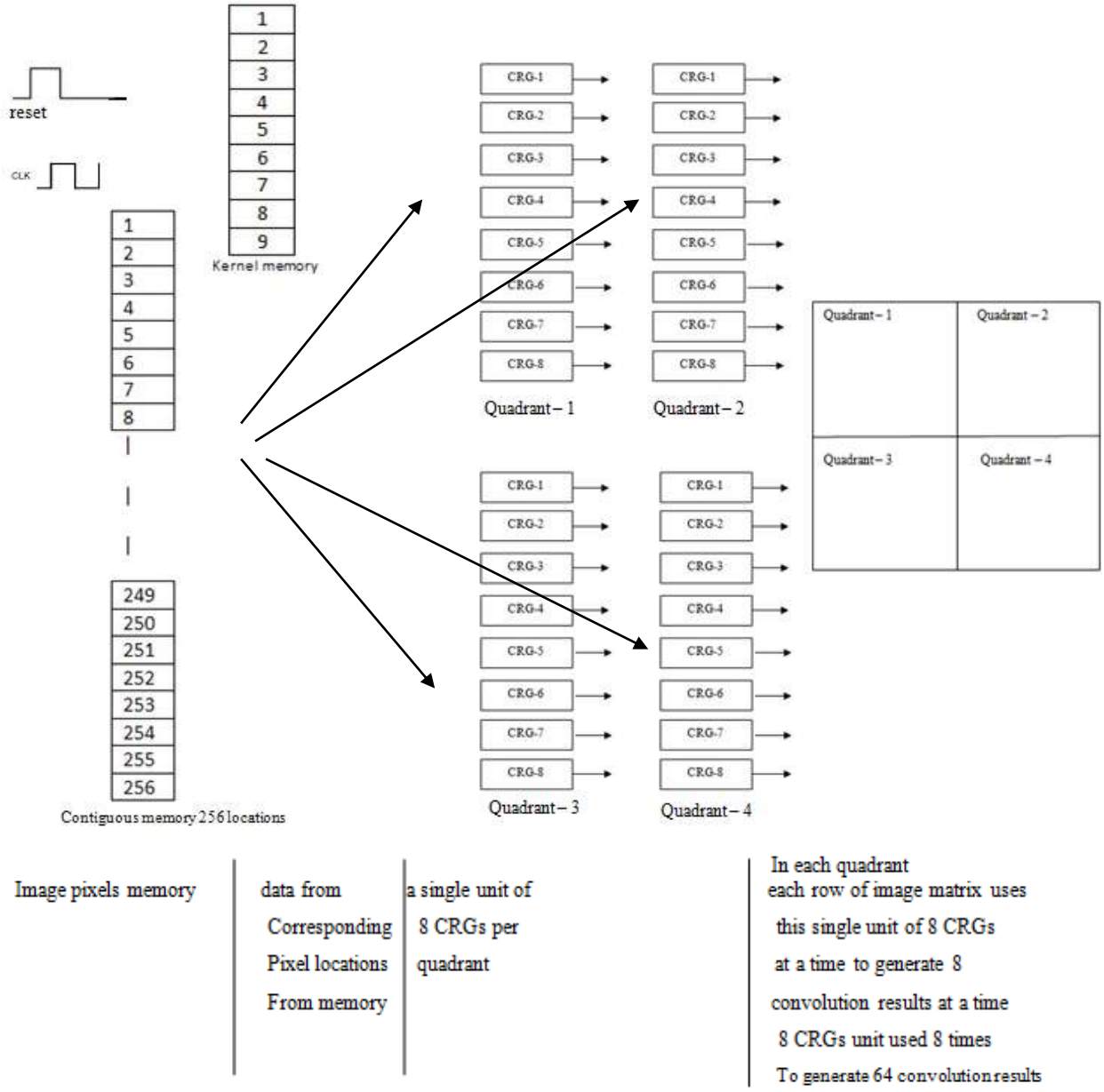


Figure 11: In this a single unit of 8 CRGs is used for a single at a time, and used 8 times for each row to generate 64 convolution results in a quadrant. Like that there are 4 different units of 8 CRGs working in parallel one for each quadrant.

3.2 Architecture – 6: Quartile Division Architecture – 2

In the architecture 5 all the CRGs are having **ripple carry adders (RCAs)** as the addition units. But in this architecture 6, **Carry Look-ahead adders (CLAs)** are used instead of RCAs in Architecture-5 in all CRGs similar to the Architecture - 3. As CLA is much faster than RCA, consequently the convolution results of the CRGs are generated much faster than Architecture – 5.

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

3.3 Architecture – 7: Quartile Division Architecture – 3

In this architecture similar to Architecture – 4, I have used single CRG unit per each quadrant means 4 CRGs are working in parallel one per each quadrant.

Here in this architecture each quadrant contains the following components

a) 64 states state machine
b) A single CRG called 64 times to generate 64 convolution results of the 8 × 8 image pixels.
c) An array memory to save all the convolution results in order (for the imaginary quadrant of 8 rows and 8 columns of 8 × 8 image pixels of the original image matrix)

Like this in all 4 quadrants, all the 4 CRGs work in parallel dedicated to that particular quadrant 8 × 8 image, and work sequentially one after the other, row after row and generate all 64 convolution results and finally all 256 convolution results are generated much faster than Architecture-4. Which are all represented in Figure 12 and 13.

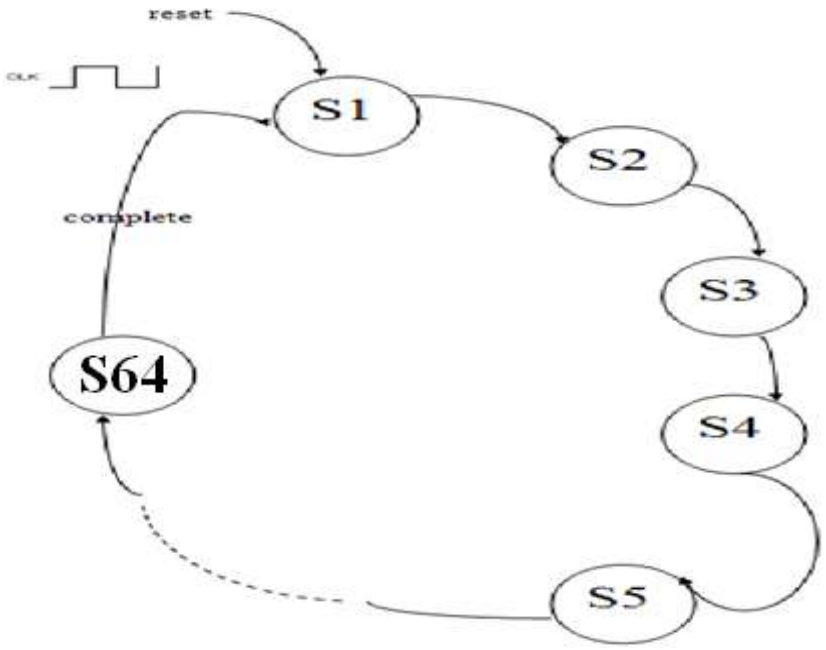


Figure 12: 64 states State Machine, each state generates 1 convolution result using a single CRG, sequentially one after the other.

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

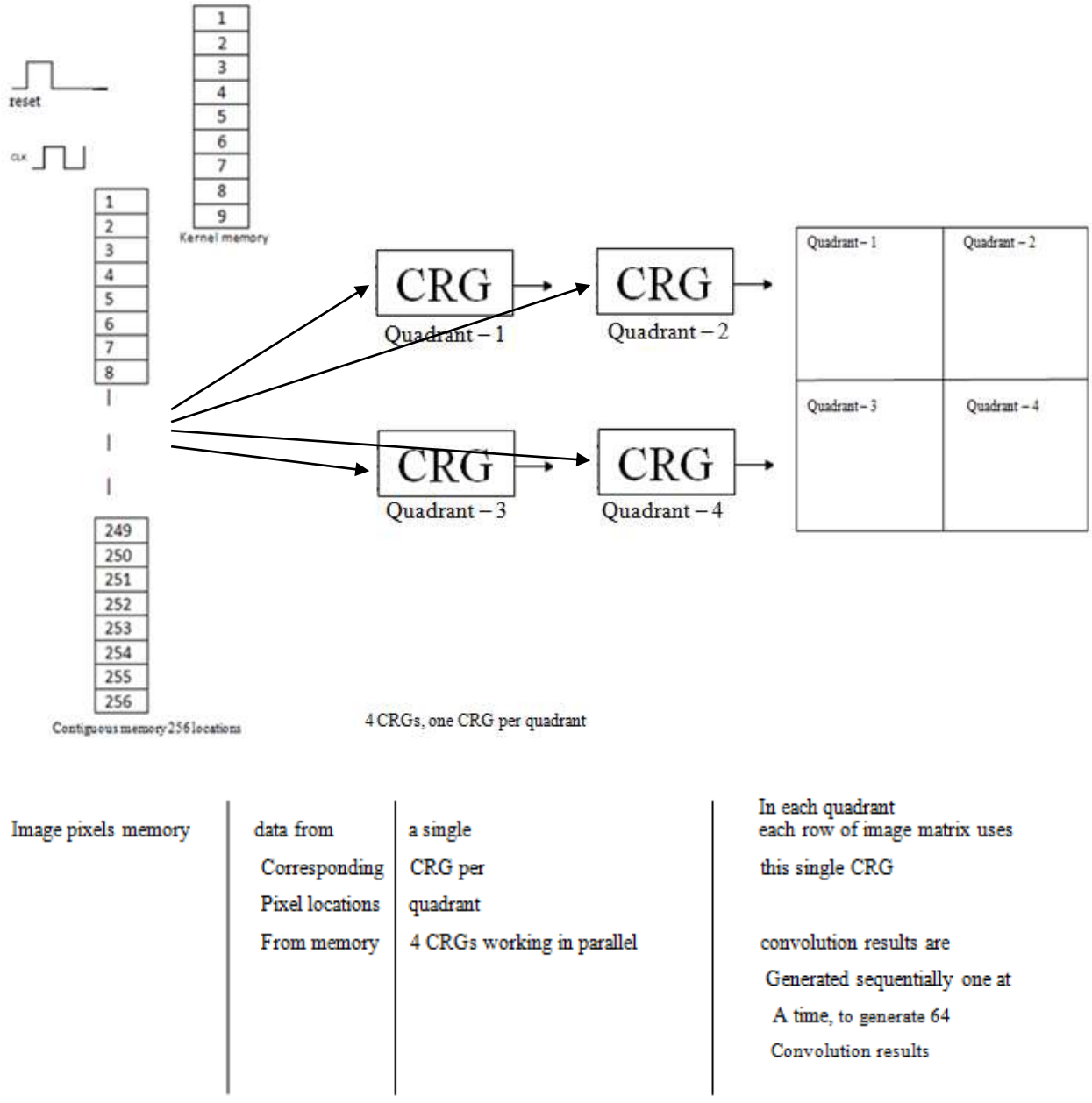


Figure 13: In this a single CRG is used and called 8 times for each row and called 64 times in that quadrant to generate 64 convolution results one after the other sequentially. Like that there are 4 different CRGs working in parallel one for each quadrant.

In this Architecture, total 16 × 16 image matrix is divided into 4 quadrants each of 8 × 8 image matrix of 64 pixels per quadrants. In each quadrant a single CRG which takes corresponding inputs required for the corresponding convolution results from the contiguous memory, is used to generate single convolution result at a time and called for 64 times to generate all the 64 convolution results. This way all the 4 CRGs work in parallel to generate all the 256 convolution results. This process makes the generation of convolution results much faster than architecture – 4.

3.4 Architecture – 8: Quartile Division Architecture – 4: Hybrid architecture

This architecture is hybrid architecture of all the four different architectures specified previously. It is a highly efficient multi-core architecture (generalized term "core"). In this architecture first the main 16 × 16 original

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

image matrix is divided into 4 quadrants. In my design I have used "All concurrent CRGs: Architecture -1" in the Quadrant – 1, "row by row state machine based Architecture: Architecture – 2" in the Quadrant – 2, "row by row state machine based Architecture with CLA: Architecture – 3" in the Quadrant – 3, and "All sequential CRGs: Architecture – 4" in the Quadrant – 4. This is fourth one in Quartile Division Architectures.

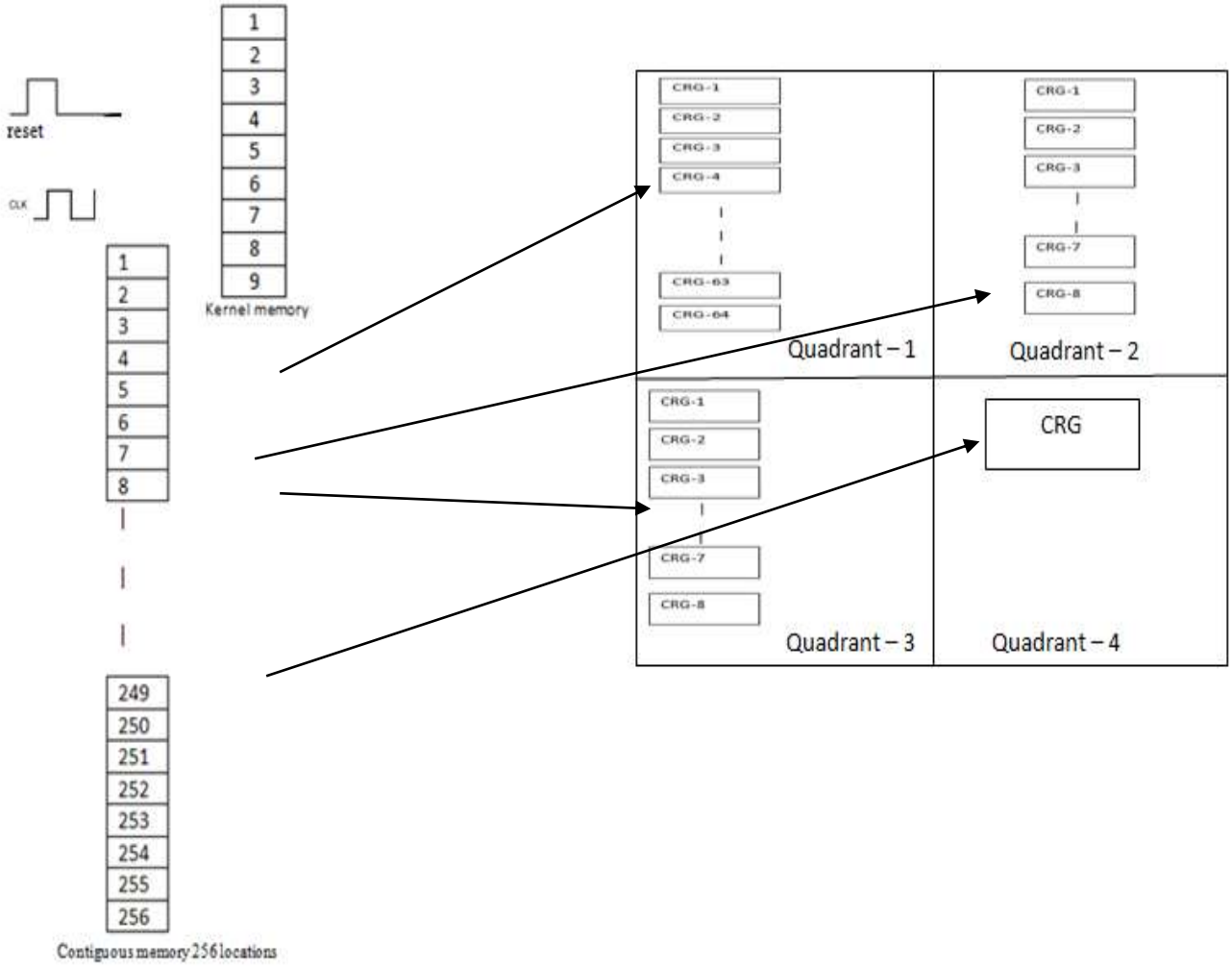


Figure 14: Hybrid quartile division architecture

In this Hybrid Quartile Division architecture, in Quadrant-1 Convolution results are generated by using the Architecture – 1. 64 different CRGs are used, and they will take corresponding image pixels from 256 location contiguous memory as inputs and concurrently generate all 64 convolution results and saved in order into array memory corresponding to the quadrant. In Quadrant – 2, I used Architecture – 2 for generating all 64 convolution results. A unit of 8 CRGs, which takes corresponding image pixels form the input memory, is used to generate convolution results of a row at a time, and using a state machine all the 64 convolution results are generated and saved in order into array memory corresponding to the quadrant. In Quadrant – 3, Architecture – 3 is used for generating 64 convolution results where CLA is used instead of RCA as in Architecture – 2, and saved in order into an array memory. Then in the Quadrant – 4 a single CRG is used sequentially 64 times to generate all 64 convolutions results and saved into an array memory of the Quadrant.

Advantages of the Quartile Division Architectures

In Quartile Division Architectures – 1, 2, 3 all the corresponding 4 CRG based units are working in parallel to generate 256 convolution results instead of a single unit in the respective mother architectures of Architecture –

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

2,3,4. So these Quartile architectures work as speeding up architectures of their mother architectures and generate all the Convolution results with high speed.

Advantages of the Hybrid Quartile Division Architecture:
In this Quadrant – 1 results are generated first, Quadrant – 3 results second, Quadrant – 4 results third, and then finally Quadrant 4 results. Basing on the organization various different quartile hybrid architectures can be developed means which architecture in which quadrant and a different organization. These types of Hybrid architectures have the following advantages.
Applications of the Novel Quartile Division Hybrid Architecture are:

1) In the image display after convolution process, first quadrant data can start displaying first instead of waiting for the all convolution finished, while displaying the 1st, during the mean time 2nd quadrant data is ready to be displayed, so then 3rd quadrant and 4th quadrant data, which speeds up the display process and saves a lot of time
2) This architecture is highly useful in finding regional (each quadrant) SNR with a single small size SNR hardware and saves a lot of time. First Quadrant 1 SNR, then Second, then third and finally 4th quadrant.
3) If to transmit multi byte serial data transmission in a network it can be started sending Quadrant 1 data first, by that time Quadrant 2 data will be ready and so on all quadrants data can be transmitted with high speed and less delay.
4) In telemedicine data transmission, on the other side of the network if there are 4 different expert doctors, each one has to look at different parts of biomedical image. Then it can be transmitted one quadrant at a time to each different expert for biomedical image analysis. This saves a lot of time instead of waiting for whole convolution process is finished.
5) This architecture is very useful and efficient in sequential memory storing. First available data will start loading and by that time next data ready to be loaded and so on. This saves a lot of time until the whole convolution process is finished.
6) If you want to do convolution of a small part of the whole big image such as 16 by 16 part; and transmit it through a network this architecture is highly useful.

And many more advantages can be found basing on the application.

4. IMPLEMENTATION RESULTS

In this research all the 8 Novel architectures are programmed using Verilog HDL and implementation is done on Xilinx Artix-7 FPGAs using Vivado 2015.2 Tool. All the architectures resource utilization, clock speed, total delay and on chip power utilization are observed. The speed improvement of quartile division architectures over first four architectures in terms of total delay is compared. The Quartile architectures and first four architectures are compared in terms of total resource utilization, on chip power utilization, total delay. The results are tabulated for all the 8 architectures for "16 by 16 size image"; "32 by 32 size image" in section 4.1 and 4.2.

4.1 16 by 16 size image results:
Architecture – 1

Table 1: Results for Architecture – 1

FFs (267600)	LUT(133800)	BUFG(32)	On chip power (W)	TOTAL DELAY(ps)	CLOCK SPEED (GHz)	TIME PERIOD(ps)
1072	442	1	38.901	768	383	2.61

Architecture – 2

Table 2: Results for Architecture – 2

FFs (267600)	LUT(133800)	BUFG(32)	On chip power (w)	TOTAL DELAY(ps)	CLOCK SPEED (GHz)	TIME PERIOD(ps)
638	222	1	24.442	2126	395	2.53

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

Architecture – 3

Table 3: Results for Architecture – 3

FFs (267600)	LUT(133800)	BUFG(32)	On chip power (w)	TOTAL DELAY(ps)	CLOCK SPEED(GHz)	TIME PERIOD(ps)
638	268	1	26.324	1741	390	2.56

Architecture – 4

Table 4: Results for Architecture – 4

FFs (267600)	LUT(133800)	BUFG(32)	On chip power (w)	TOTAL DELAY (ps)	CLOCK SPEED (GHz)	TIME PERIOD(ps)
431	142	1	9.801	26072	450	2.22

Architecture – 5: Quartile Division Architecture – 1

Table 5: Results for Architecture – 5

FFs (267600)	LUT(133800)	BUFG(32)	On chip power (w)	TOTAL DELAY(ps)	CLOCK SPEED(GHz)	TIME PERIOD(ps)
682	462	1	28.295	1354	392	2.55

Architecture – 6: Quartile Division Architecture – 2

Table 6: Results for Architecture – 6

FFs (267600)	LUT(133800)	BUFG(32)	On chip power (w)	TOTAL DELAY(ps)	CLOCK SPEED(GHz)	TIME PERIOD(ps)
690	484	1	29.092	1084	388	2.58

Architecture – 7: Quartile Division Architecture – 3

Table 7: Results for Architecture – 7

FFs (267600)	LUT(133800)	BUFG(32)	On chip power (w)	TOTAL DELAY(ps)	CLOCK SPEED(GHz)	TIME PERIOD(ps)
510	169	1	12.036	5481	445	2.25

Architecture – 8: Quartile Division Architecture – 4: Hybrid Architecture

Table 8: Results for Architecture – 8

FFs (267600)	LUT(133800)	BUFG(32)	On chip power (w)	TOTAL DELAY(ps)	CLOCK SPEED(GHz)	TIME PERIOD(ps)
920	539	1	32.978	5708 (max of)	386	2.59

4.2 32 by 32 size image results:

Architecture – 1

Table 9: Results for Architecture – 1

FFs (267600)	LUT(133800)	BUFG(32)	On chip power (w)	TOTAL DELAY(ps)	CLOCK SPEED(GHz)	TIME PERIOD(ps)
2573	1061	1	60.301	2074	370	2.70

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

Architecture – 2

Table 10: Results for Architecture – 2

FFs (267600)	LUT(133800)	BUFG(32)	On chip power (w)	TOTAL DELAY(ps)	CLOCK SPEED(GHz)	TIME PERIOD(ps)
1468	578	1	35.440	5013	384	2.60

Architecture – 3

Table 11: Results for Architecture – 3

FFs (267600)	LUT(133800)	BUFG(32)	On chip power (w)	TOTAL DELAY(ps)	CLOCK SPEED(GHz)	TIME PERIOD(ps)
1596	644	1	43.435	4353	377	2.65

Architecture – 4

Table 12: Results for Architecture – 4

FFs (267600)	LUT(133800)	BUFG(32)	On chip power (w)	TOTAL DELAY(ps)	CLOCK SPEED(GHz)	TIME PERIOD(ps)
906	341	1	13.231	67788	434	2.30

Architecture – 5: Quartile Division Architecture – 1

Table 13: Results for Architecture – 5

FFs (267600)	LUT(133800)	BUFG(32)	On chip power (w)	TOTAL DELAY (ps)	CLOCK SPEED (GHz)	TIME PERIOD (ps)
1726	1088	1	42.445	2980	379	2.64

Architecture – 6: Quartile Division Architecture – 2

Table 14: Results for Architecture – 6

FFs (267600)	LUT(133800)	BUFG(32)	On chip power (w)	TOTAL DELAY (ps)	CLOCK SPEED (GHz)	TIME PERIOD (ps)
1812	1123	1	44.729	2710	374	2.67

Architecture – 7: Quartile Division Architecture – 3

Table 15: Results for Architecture – 7

FFs (267600)	LUT(133800)	BUFG(32)	On chip power (w)	TOTAL DELAY (ps)	CLOCK SPEED (GHz)	TIME PERIOD (ps)
1204	404	1	28.766	13155	427	2.34

Architecture – 8: Quartile Division Architecture – 4: Hybrid Architecture

Table 16: Results for Architecture – 8

FFs (267600)	LUT(133800)	BUFG(32)	On chip power (w)	TOTAL DELAY (ps)	CLOCK SPEED (GHz)	TIME PERIOD (ps)
2199	1375	1	45.542	14429(max of)	371	2.70

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

4.3 Result Analysis of 16 by 16 size image

Table 17: % Comparison of Architecture – 5 over Architecture – 2

	Architecture - 2	Architecture – 5	Percentage (%)
Total on chip resources	**861**	**1145**	**32.98 % more**
Total on chip power(W)	**24.442**	**28.295**	**15.76 % more**
Total Delay (ps)	**2126**	**1354**	**57% less (1.57 times faster)**

Table 18: % Comparison of Architecture – 6 over Architecture – 3

	Architecture - 3	Architecture – 6	Percentage (%)
Total on chip resources	**907**	**1175**	**29.55 % more**
Total on chip power(W)	**26.324**	**29.092**	**10.52 % more**
Total Delay (ps)	**1741**	**1084**	**60.61% less (1.61 times faster)**

Table 19: % Comparison of Architecture – 7 over Architecture – 4

	Architecture - 4	Architecture – 7	Percentage (%)
Total on chip resources	**574**	**680**	**18.47 % more**
Total on chip power(W)	**9.801**	**12.036**	**22.80 % more**
Total Delay (ps)	**26072**	**5481**	**375.68 % less (4.76 times faster)**

The observed implementation results of these designs on Xilinx Artix-7 FPGAs for 16 by 16 size image are tabulated for comparison purposes in Table 17, Table 18 and Table 19. From the **Table 17** it is observed that the Architecture – 5 over Architecture – 2 is utilizing 32.98%, 15.76% more resources and power on the chip, but taking far less time means 57% less time, to get the final result and going 1.57 times faster than the Architecture – 2. This is highly useful speed improvement.

From the **Table 18** it is observed that Architecture – 6 over Architecture – 3 is taking far less time 60.61% less time and running 1.61 times faster, but at the expense of 29.55%,10.52% more resources and power on the chip, Very useful speeding up process.

It is observed largely the improvement in speed when it comes to Architecture – 7 over Architecture – 4 in the **Table 19**. In this case Architecture – 7 taking 18.47% more resources, 22.80% more power but taking 375.68% far less time and running at 4.76 times faster.

4.4 Result Analysis of 32 by 32 size image

Table 20: % Comparison of Architecture – 5 over Architecture – 2

	Architecture - 2	Architecture – 5	Percentage (%)
Total on chip resources	**2047**	**2815**	**37.52 % more**
Total on chip power(W)	**35.44**	**42.445**	**19.77 % more**
Total Delay (ps)	**5013**	**2980**	**68.22% less (1.68 times faster)**

Table 21: % Comparison of Architecture – 6 over Architecture – 3

	Architecture - 3	Architecture – 6	Percentage (%)
Total on chip resources	**2241**	**2936**	**31 % more**
Total on chip power(W)	**43.435**	**44.729**	**2.98 % more**
Total Delay (ps)	**4353**	**2710**	**60.63 % less (1.61 times faster)**

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

Table 22: % Comparison of Architecture – 7 over Architecture – 4

	Architecture - 4	Architecture – 7	Percentage (%)
Total on chip resources	**1248**	**1609**	**28.93 % more**
Total on chip power(W)	**13.231**	**28.766**	**1.17 % more**
Total Delay (ps)	**67788**	**13155**	**415.3 % less (5.15 times faster)**

The implementation results tabulated in Table 20, Table 21, and Table 22 gives the performances of Quartile Architectures (Architecture – 5, Architecture – 6, Architecture – 7). Architecture – 5 is taking 37.52%, 19.77% more resources and power but running at 1.68 times faster by taking 68.22% less time over Architecture – 2 (**Table 20**). Architecture – 6 is working with 1.61 times faster by taking 60.63% less time by utilizing 31%, 2.98% more resources and more power than Architecture – 3 (**Table 21**). A very specific observation can be made similar to the case of 16 by 16 size image, that Architecture – 7 is running at 5.15 times faster than Architecture – 4 by taking 415.3% less time, by utilizing 28.93%, 1.17% more time on chip resources and power (**Table 22**).

4.5 Result analysis of Architecture – 1 and Architecture – 8

The ultimate goal of all these architectures is to achieve high speeds, at the same time keeping in mind of optimized considerations on power, device utilization.

4.5.1 All Concurrent Architecture: Architecture – 1

The Maximum speed can be obtained from the Architecture – 1. This architecture generates all convolution results concurrently at a time and this architecture generates all the results in shortest time than all the architectures including Quartile Division Architectures. For 16 by 16 size image all results are available in 768 Pico seconds. When it comes to 32 by 32 size image it generates all the results in 2074 Pico seconds. But this architecture takes highest resources than any architecture including quartile division architectures in both size images.

Architecture – 1

Table 23: Results for Architecture – 1 (16 size)

FFs (267600)	LUT(133800)	BUFG(32)	On chip power (W)	TOTAL DELAY(ps)	CLOCK SPEED (GHz)	TIME PERIOD(ps)
1072	**442**	**1**	**38.901**	**768**	**383**	**2.61**

Architecture – 1

Table 24: Results for Architecture – 1 (32 size)

FFs (267600)	LUT(133800)	BUFG(32)	On chip power (w)	TOTAL DELAY(ps)	CLOCK SPEED(GHz)	TIME PERIOD(ps)
2573	**1061**	**1**	**60.301**	**2074**	**370**	**2.70**

4.5.2 Hybrid Architecture: Architecture – 8

This architecture consists of 4 different architectures in its all 4 quadrants. So it is hybrid architecture and a sort of Multi-core architecture because each core consists of its own computational units and memory. This architecture is highly efficient than all architectures. Here in all the 8 architectures the array memory for saving the convolution results is having the property that all memory locations are accessible concurrently while writing the result, but while reading the result the data has to be accessed sequentially. Due to this property to access the results to read total convolution process has to be finished. This constraint is there in all 8 architectures including the quartile architectures. Until all the process is finished in each quartile the data cannot be accessed from each independent quarter memory. Each Quarter contains its own memory block of this property. Such a memory blocks are commonly used in computer systems. Keeping in mind of such property of memory I have designed all these 8 architectures.

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

In section 3.4 the described hybrid architecture is very useful than the Architecture – 4 – All sequential architecture in terms of Hybridization when compared to Architecture - 4. This architecture is also efficient in when compared to Architecture – 7: Quartile Division All sequential architecture. This architecture efficiency over Architectures – 4 & 7 is considered for the example application of byte by byte serial data transmission in a network.

For 16 by 16 size image:
All the details of this Hybrid Architecture are given as follows. Here given for **16 by 16 size image.**
Clock Speed: 386 GHz
Clock period: 2.59 ps

Total Delay:
Concurrent: Quarter - 1: Total Delay = 850 ps
Row by Row - RCA: Quarter – 2: Total delay = 1466 ps
Row by Row - CLA: Quarter – 3: Total delay = 1184 ps
Sequential: Quarter – 4: Total delay = 5708 ps (Maximum delay of the final result)

If to transfer all the 256 convolution coefficient results through a network byte by byte serial data transmission, in this regard it can be started transmitting Quarter – 1, 64 results in advance instead of waiting for all the result is ready, and then some wait time for Quarter – 3 results and transmit its 64 results and then some wait time for the result in Quarter – 2 and then transmit its 64 results and then wait time for Quarter – 4 results and then transmit its 64 results.

If each one of the results to be sent in one clock cycle, then there are 256 results to be sent. The maximum time it takes is 5708 ps + 64 * (clock period = 2.59 ps) = 5708 + 165.76 = 5873.76 ps.
If it is Architecture – 4, it takes total time of 26072 ps + 256 * (clock period = 2.22 ps) = 26640.32 ps. If it Architecture – 7, it takes a total time of 5481 ps + 256 *(clock period = 2.25 ps) = 6057 ps.

So, this Hybrid architecture does 183.24 ps earliest transmission over Architecture – 7; and 20763.56 ps earliest transmission over Architecture – 4.
This similar way this Hybrid architecture is very useful in the case of regional SNR calculation also.

For 32 by 32 size image:
The details of its Hybrid Architecture are
Clock Speed: 371 GHz
Clock period: 2.70 ps

Total Delay:
Concurrent: Quarter - 1: Total Delay = 1785 ps
Row by Row - RCA: Quarter – 2: Total delay = 3372 ps
Row by Row - CLA: Quarter – 3: Total delay = 2842 ps
Sequential: Quarter – 4: Total delay = 14429 ps (Maximum delay of the final result)

If it is considered for the serial data transmission application this Hybrid architecture takes 14429 ps + 256 * (clock period = 2.70 ps) = 15120.2 ps. If it is Architecture – 4, it takes a total time of 67788 ps + 1024 * (clock period = 2.30 ps) = 70143.2 ps. If it is Architecture – 7, then it takes 13155 ps + 1024 * (clock period 2.34 ps) = 15551.16 ps. So, Hybrid architecture does the serial data transmission by 430.96 ps earliest over Architecture - 7, 55023 ps earliest over Architecture – 4.

This similar way this Hybrid architecture is very useful in the case of regional SNR calculation also.
Basing on the requirement the Hybrid architecture can be designed in such a way that Quarter – 1 and Quarter – 4 with Architecture – 1, and Quarter – 2 and Quarter – 3 with CLA and RCA based FSM architectures so that such Hybrid Architecture can be much faster than Architectures - 2, 3, 5, 6. This way various architectural organization of Hybrid Architecture can be very useful in increasing the throughput basing on the application.

ISSN: 2277-9655
Impact Factor: 5.164
CODEN: IJESS7

5. CONCLUSION AND FUTURE RESEARCH

In this research I have developed 8 different Novel architectures of my own for Digital Image Processing Convolution Filter Hardware Design which never were there before in literature. I have implemented all these architectures on Xilinx Artix-7 FPGAs using VIVADO with Verilog HDL programming language. Quartile Division architectures are found to be highly useful in speeding up the convolution process. Hybrid architectures are found to be specifically very useful Quartile Division Architectures with some important way of efficiency. All these architectures are useful in all applications wherever convolution is required. These Small size convolution hardware I have implemented is useful wherever a small part of a whole large image to be found its convolution.

Future research:

First I started designing these architectures with half-half split multiplication method. This is giving results for positive numbers correctly, for negative numbers some times. So, I have used CLA based design. As future research I would like to thoroughly test this split multiplication computer arithmetic for hardware for negative numbers also. This way it can speed up the process of multiplication unit and adder unit together in a single architecture and one more much faster architecture can be developed.

REFERENCES

[1] Ranganadh Narayanam, "Translation Invariance (TI) based Novel Approach for better De-noising of Digital Images", IRJET, vol 4, Issue 3, March 2017.
[2] Rafael C Gonzalez and Richard E Woods, "Digital Image Processing", third edition, Pearson Education, 2007

CITE AN ARTICLE

Narayanam, R. (2019). FPGA IMPLEMENTATION OF NOVEL ARCHITECTURES FOR DIGITAL IMAGE PROCESSING CONVOLUTION FILTER: DEVELOPMENT OF NOVEL QUARTILE DIVISION ARCHITECTURES. *INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY*, *8*(2), 205-227.