CHAPTER 3

DESIGN OF RECONFIGURABLE ARCHITECTURE FOR WCDMA AND OFDM RECEIVER

3.1 INTRODUCTION

This chapter comprises two sections. The first section briefs about the motivation to design the target architecture used for WCDMA and OFDM receiver systems. The second section presents various blocks that are used to describe the target architecture. It also discusses the arithmetic elements used in the reconfigurable cells by examining the commonly used adders and multipliers.

3.2 ARCHITECTURE MOTIVATION

In this section motivation of the proposed reconfigurable architecture is presented by analyzing the limitations of existing reconfigurable architectures.

The primary design objective is to develop application domain-specific CGRAs, which achieve high performance and energy efficiency approaching those of ASICs, while retain adequate flexibility, as they can be reconfigured to implement different applications. When a certain application domain is targeted, there are special features that must be considered and exploited. Specifically, the number of computationally intensive kernels must be small enough to perform repetitive execution of word level processing.

Various popular CGRAs discussed in Section 3.5 have been emerging as a potential architecture for various kinds of applications. However, there are still many outstanding issues that prevent existing CGRAs from using the application specific reconfigurable receiver architectures. They vary a lot in architectural aspects and computational models. The existing reconfigurable architectures comprise arrays of computational cells enabling the mapping of all the desired behaviours. However this is achieved at the expense of large area overheads and low utilization of the computational resources since only half of the computational components are active per control step in every flexible cell. So the area efficiency and the high utilization of the underlying hardware remove the inherent limitations of the existing reconfigurable architectures, where large silicon area portions remains unutilized during execution.

3.3 PROPOSED RECONFIGURABLE ARCHITECTURE

The proposed Reconfigurable Architecture is shown in Figure 3.1.

This architecture consists of

1. An Array of Processing Element Slice (PES) and their interconnects tiled in a two dimensional array.
2. A controlling block to control the functions of all the blocks
3. Input/Output (I/O) block configured to receive pre-processed data, deliver processed data out and determine the required functionality. It is a configurable I/O block that connects the processing unit with the outside world.
4. A memory block that is configured to cache the buffer data.

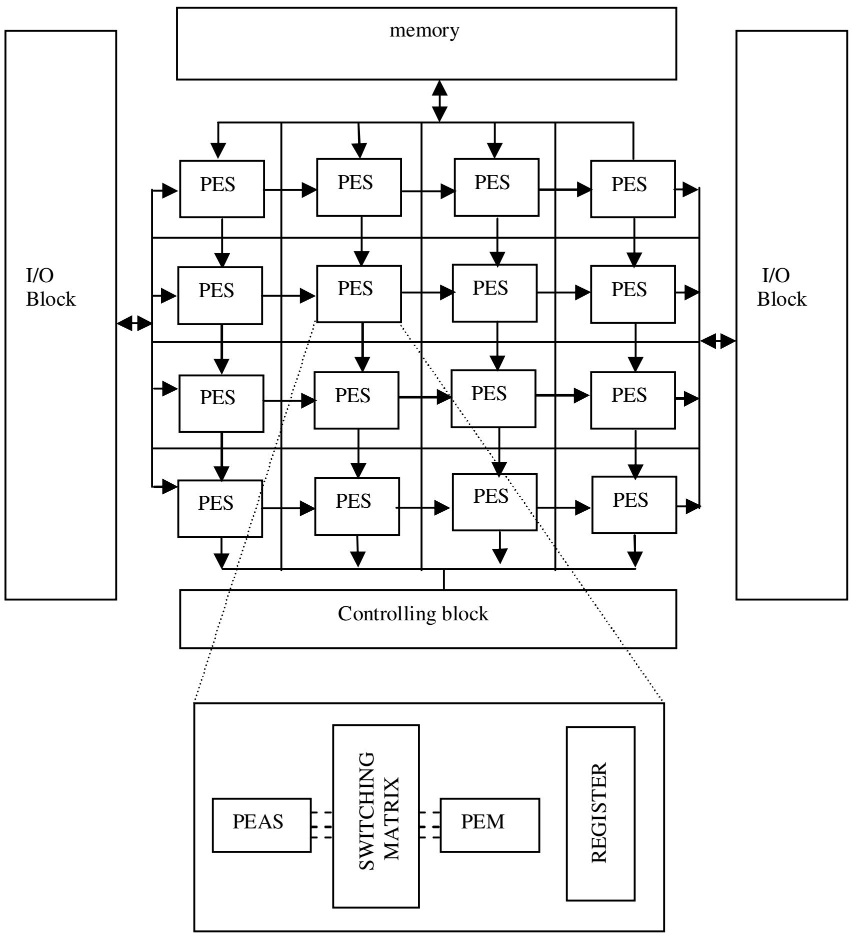


Figure 3.1 Proposed reconfigurable architecture

3.3.1 RC Array

Reconfigurable component of the proposed architecture is an array of RCs or PESs. Considering the target applications (WCDMA or OFDM based WiFi), it tends to be processed in clusters of 4×4 PESs, the RC array has 16 cells in a two-dimensional matrix, as illustrated in Figure 3.1. This architecture is chosen to maximally utilize the parallelism inherent in an application, which in turn enhances throughput. All RCs in the same row/column share the same configuration data. However, each RC operates on different data. Sharing the context across a row/column is useful for data-parallel applications. The RC array has an extensive three layer interconnection network, designed to enable fast data exchange between the RCs. This results in enhanced performance for application kernels that involve high data movement, for example, FFT computation used in OFDM based WiFi standard. Each RC incorporates adder, subtractor, multiplier, multiplexers and a register file. In addition, there is a context register that is used to store the current context and provide control/configuration signals to the RC components.

3.3.2 Processing Element Slices

The proposed architecture has a 2-D array of PESs. Each PES can be configured to perform a specific word-level (16 or 32 bit) operation in every cycle. Each PES consists of two types of PE named PEAS (Processing Element Add/Sub) and PEM (Processing Element Multiplier) as shown in Figures 3.2 and 3.3. The computations supported by these PEs are addition, subtraction and multiplication and these operations are configured depending on the need of a specific application. PEAS contains 4 add/sub blocks. Each add/sub block is configured to perform either addition or subtraction using the control bit. PEM contains 3 adders, 2 subtractors and 3 multipliers. Data flow between these PEs are configured through a switching matrix, and which also depends on the requirements of different applications. PES is the basic computational block used to perform FFT operation for OFDM based WLAN standard and RAKE finger operation for WCDMA cellular standard. A local register file inside each PES acts as a temporary storage for intermediate values between computations and values fetched from memory.

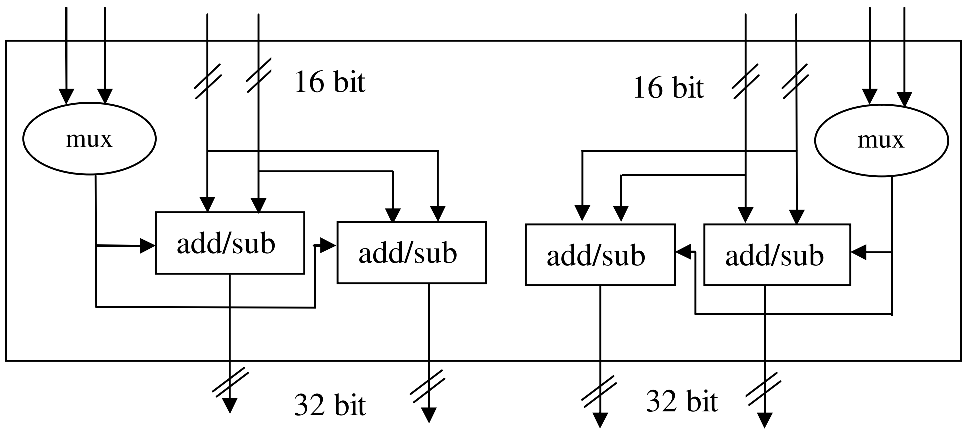


Figure 3.2 Computational elements and its structure of PEAS

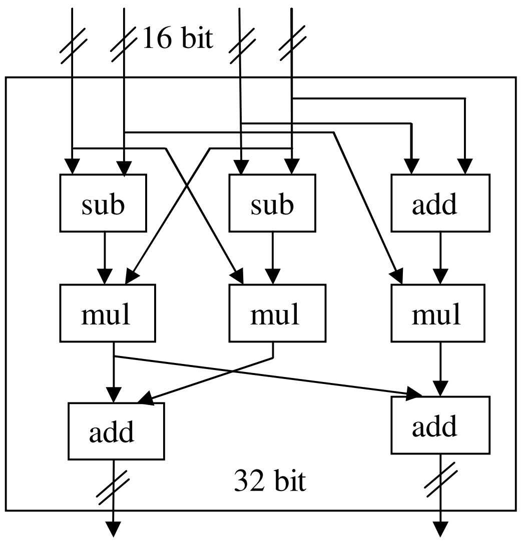


Figure 3.3 Computational elements and its structure of PEM

3.3.3 Adder Block

Addition is a very crucial operation because it usually involves a carry ripple step which must propagate a carry signal from each bit to its higher bit position. This results in a substantial circuit delay. The adder, therefore which lies in the critical delay path, effectively determines the system's overall speed. On the other hand, the option of reducing area and power consumption of the designed adder, which for many years has been a narrow specialty, has recently been gaining prominence. This can be attributed to the emergence and increasing popularity of smaller and more durable mobile computing and communication systems.

3.3.3.1 Ripple Carry Adder (RCA)

The RCA is the simplest adder circuit. An N-bit RCA structure is shown in Figure 3.4. It consists of N full adders with the carry signal propagating from one full adder stage to the next from LSB to MSB. It has many advantages which include low power, low area and a simple layout. The drawback of the ripple carry adder, though, is its slow speed. The delay of the adder is linearly dependent on the bit-width of the adder. The critical path of the ripple carry adder consists of the carry chain from the first full adder to the last.

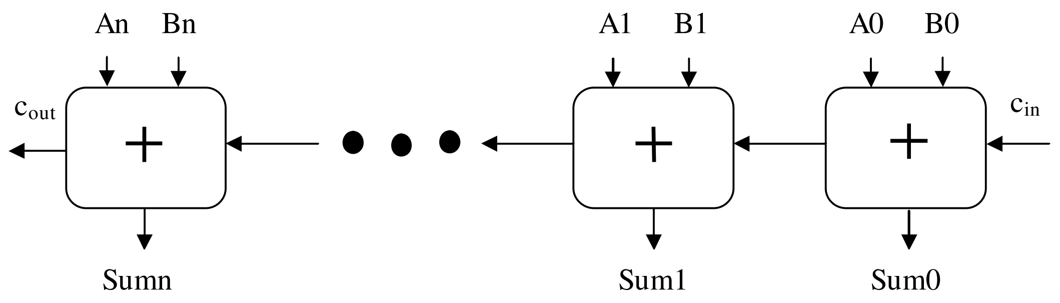


Figure 3.4 Structure of N-bit RCA

Area requirement of 8 bit RCA is calculated as given in Table 3.1 and Table 3.2. From the Table 3.1 it is observed that the area occupied by full adder is 62.18 square µm. So the total area occupied by 8-bit RCA is 497.44 square µm since 8 full adders are required to design 8-bit RCA. The total delay is 0.96 ns which is due to 8 full adders since the circuit delay of each gate is 0.017 ns.

Table 3.1 Area requirements of full adder

Gate Requirements for full adder	Area (sq. µm)	Quantity	Total Area (sq.µm)
XOR	9.02	2	18.04
AND	8.82	3	26.46
OR	8.84	2	17.68
Net Area (sq.µm)			62.18

Table 3.2 Area requirements of RCA

Name of the adder	No. of full adder	Total Area (sq.µm)
RCA-8 bit	8	497.44

3.3.3.2 Carry Select Adder (CSA)

The CSA employs an intelligent technique to reduce the carry propagation delay. As the carry signal takes on a value of either a 1 or a 0, if the sum is calculated for both the cases in advance, the carry propagation chain is reduced to just the selection of the correct outputs at each stage using multiplexers. The critical path will now just consist of multiplexers at the output of each bit. Figure 3.5 shows the structure of 4-bit CSA. The speed of CSA is achieved at the cost of doubling the area, because two adders per bit are required, one adder to calculate the sum with a carry-in of 0 and another adder to calculate the sum with a carry-in of 1. In addition, multiplexer is needed for every bit to choose the result based on the actual carry value. As a consequence of this duplication of logic, this design also consumes more power.

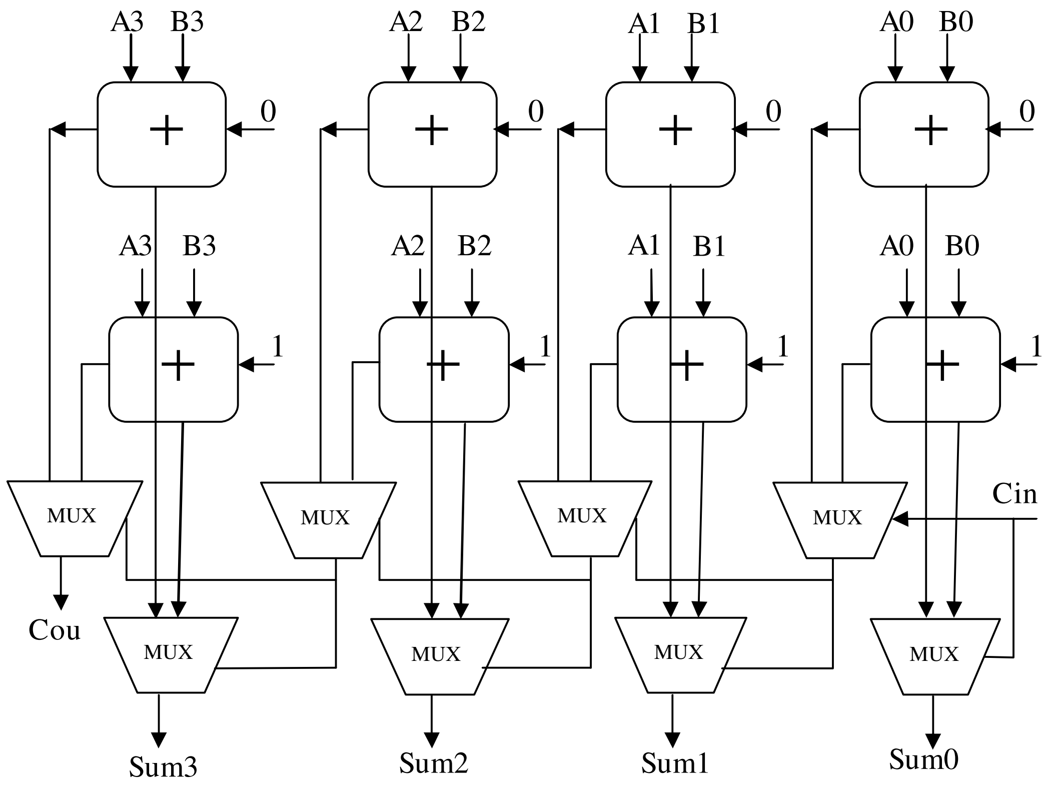


Figure 3.5 Structure of 4-bit CSA

Table 3.3 Area requirements of 8-bit CSA

Hardware Requirements	Area (sq. µm)	Quantity	Total Area (sq.µm)
full adder	62.18	16	994.88
multiplexer	9.81	16	156.96
Net Area (sq.µm)			1151.84

Area requirement of 8-bit CSA is calculated as given in Table 3.3. From the table it is observed that the area occupied by 8-bit CSA is 1151.84

square µm. The total delay is 0.624 ns which is due to 8 multiplexers since the circuit delay of each multiplexers is 0.078 ns.

3.3.3.3 Carry Look-ahead Adder (CLA)

Carry-ripple delay grows linearly with the size of the input operand for the RCA, but these delays can be shortened by generating the carries of each stage in parallel. The CLA adder is theoretically one of the fastest methods for addition. This adder is based on carry generation and propagation logic. Figure 3.6 shows the block diagram of 4-bit CLA. The delay time of the CLA architecture exhibits logarithmic dependency on the size of the adder, which allows the propagation delay of the carry signal to be minimized. Therefore, the CLA comes in handy for better delay-reduction performance. It is not advantageous, however, the CLA consumes more area and power because of its large number of logic gates

.

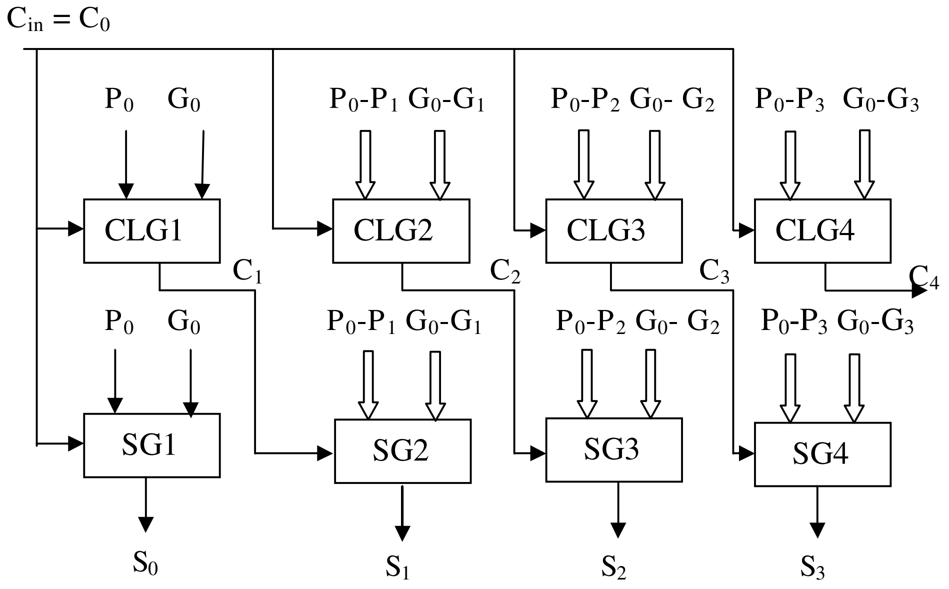


Figure 3.6 Block diagram of 4-bit CLA

Area requirement of 8-bit CLA is calculated as given in Table 3.4. The list of hardware and the area allocated by each hardware element is given in the table 3.4. From the table it is observed that the area occupied by 8-bit CLA is 640.16 square µm. The delay time of 8-bit CLA is 0.903ns, which is the logarithmic dependency on the size of the adder.

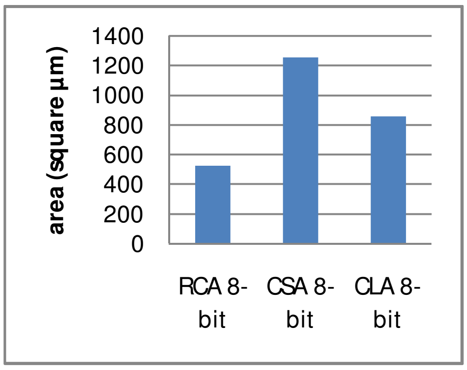
Table 3.4 Area requirements of 8-bit CLA

Hardware Requirements	Area (sq. µm)	Quantity	Total Area (sq.µm)
full adder	62.18	8	497.44
AND Gate	8.82	8	70.56
XOR Gate	9.02	8	72.16
Net Area (sq.µm)			640.16

The RCA, CSA, and CLA adder designs have been created in Verilog and synthesized in a 0.18 micron standard CMOS cell library using Leonardo Spectrum of Mentor Graphics Corp. in order to validate the theoretical results. Area and timing results of synthesized adders are listed in Table 3.5. These results has included the area and delay due to interconnect lines between hardware elements. As shown by the results given in Table 3.5 and Figure 3.7, the CLA is the fastest scheme for implementing addition, whereas the RCA is the smallest. CSA is the fastest area-efficient adder, as it is slightly smaller than a CLA, but significantly faster than an RCA. Since the goal is to target area efficiency and low power consumption for the proposed architecture, the simple and area efficient RCA structure has been chosen for addition.

Table 3.5 Area and timing results of all synthesized adders

Adder Design	Area in (square µm)	Min. Clock delay (ns)
RCA 8-bit	524.62	1.33
CSA 8-bit	1251.84	1.07
CLA 8-bit	857.02	1.09



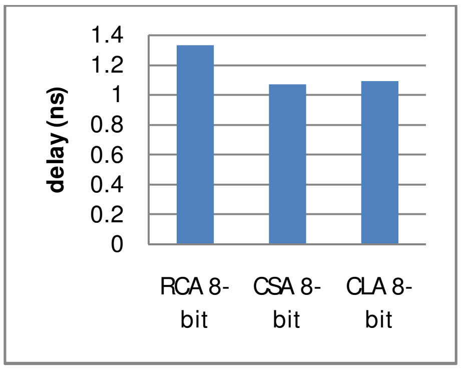


Figure 3.7 Area and speed comparisons of 8-bit RCA, CSA and CLA adder designs

3.3.4 Multiplier Block

Multiplication is an important fundamental arithmetic function currently implemented in multistandard wireless communication applications such as FFT, RAKE Receiver operation and others. They usually contribute significantly to the time delay and take up a great deal of silicon area in the architecture. Since multiplication dominates the execution time of most digital signal processing applications, using a high speed multiplier is very desirable. With an ever-increasing quest for greater computing power on battery operated mobile devices, design emphasis has shifted from optimizing conventional delay time and area size to minimizing power dissipation while still maintaining high performance. Therefore it is needed to select area efficient and high speed multiplier, which is to be used in the architecture.

3.3.4.1 Array multiplier

This is one of the simplest techniques for implementing multiplication. The idea is to add all the N partial products sequentially using N-1 adders. In order to multiply N bit values then in effect it requires N-1 of N-bit adders or N×(N-1) single adder cells. The structure of the array multiplier is shown in Figure 3.8. Array multipliers are very slow as their critical path is very long. The blocks shaded in grey show one such critical path. The advantage of array multiplier is that because of its regular structure it is easy to design.

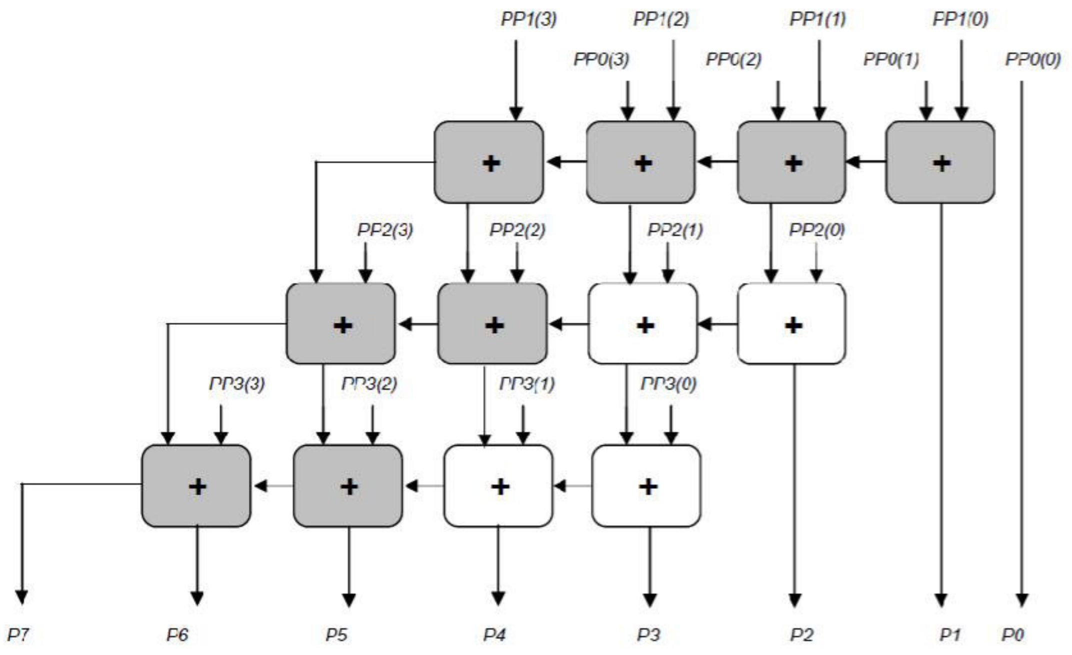


Figure 3.8 Structure of N×N array multiplier

Area requirement of 8-bit array multiplier is calculated as given in Table 3.6. The list of hardware and the area allocated by each hardware element is given in the table. From the table it is observed that the area occupied by 8-bit array multiplier is 3736.18 square μm. The delay time of 8-bit array multiplier is 2.38ns, which is due to the critical path of 20 full adders.

Table 3.6 Area requirements of 8-bit array multiplier

Hardware Requirements	Area (sq. µm)	Quantity	Total Area (sq.µm)
full adder	62.18	49	3046.82
Half adder	17.84	7	124.88
AND Gate	8.82	64	564.48
Net Area (sq.µm)			3736.18

3.3.4.2 Wallace tree multiplier

C. S. Wallace propounded a fast technique to perform multiplication in 1964. The Wallace tree multiplier belongs to a family of multipliers called column compression multipliers. The underlying principle in this family of multipliers is to achieve partial product accumulation by successively reducing the number of bits of information in each column using full adders or half adders.

The dot diagram of 8×8 Wallace tree Multiplier is shown in Figure 3.9. The eight rows of eight dots each at the top of the figure represent the partial product matrix formed by the AND gates. Four reduction stages are required with matrix heights of 6, 4, 3 and 2. In the figure, two dots joined by a diagonal line indicate that these two dots are the outputs from a (3:2) adder. Similarly two dots joined by a crossed diagonal line indicate that these two dots are the outputs from a (2:2) adder. Wallace tree multiplier requires 64 AND gates, 38 (3:2) adders and 15 (2:2) adders. This multiplier uses as much hardware as possible to compress the partial product matrix as quickly as possible into the final product. Although the amount of hardware required to perform this style of multiplication is large, the delay is near optimal.

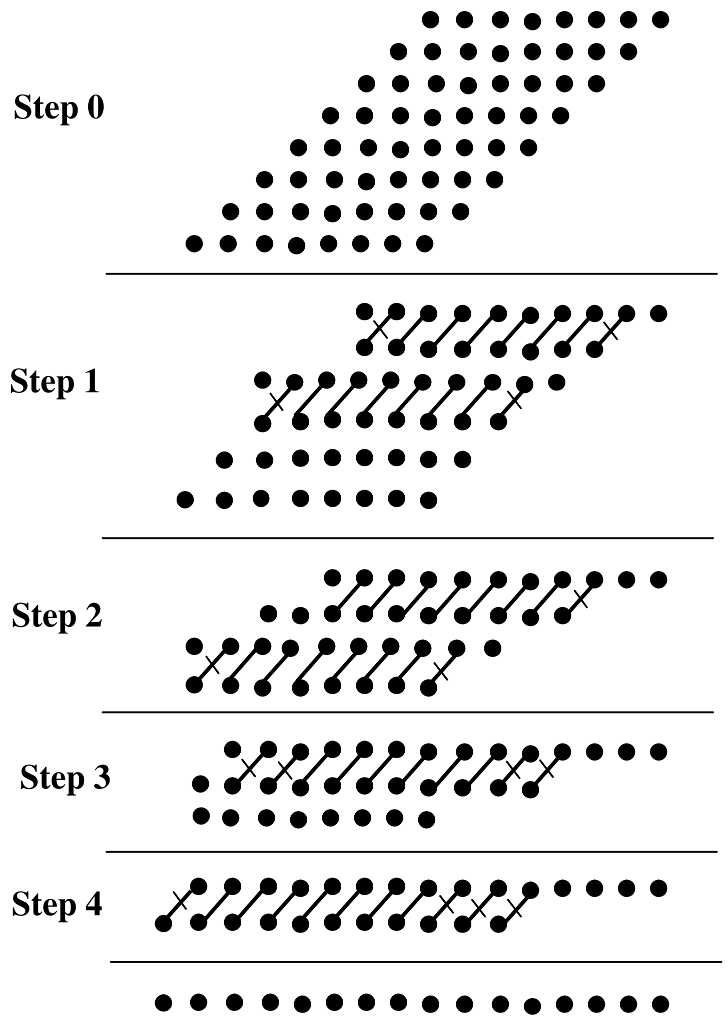


Figure 3.9 Dot diagram for an 8×8 Wallace tree multiplier

Table 3.7 Area requirements of 8-bit Wallace Tree multiplier

Hardware Requirements	Area (sq. µm)	Quantity	Total Area (sq.µm)
full adder Half adder AND Gate	62.18 17.84 8.82	38 15 64	2362.84 267.6 564.48
Net Area (sq.µm)			3194.92

Area requirement of 8-bit Wallace Tree multiplier is calculated as given in Table 3.7. The list of hardware and the area allocated by each hardware element is given in the table. From the table it is observed that the area occupied by 8-bit array multiplier is 3194.92 square µm. The delay time of 8-bit Wallace Tree multiplier is 1.785ns, which is due to the critical path of 15 adders.

3.3.4.3 Dadda multiplier

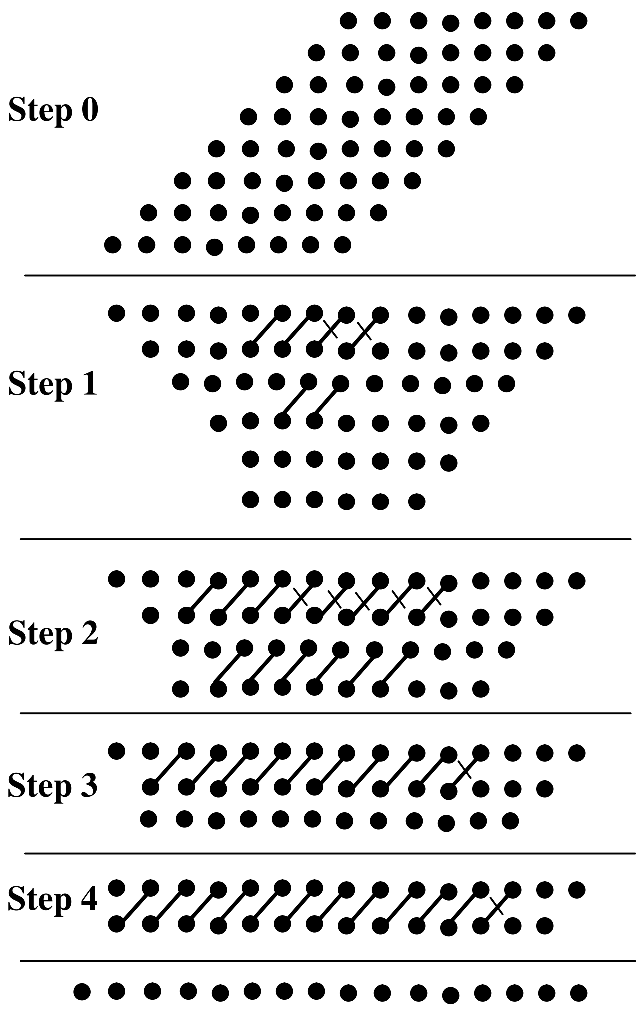


Figure 3.10 Dot diagram for an 8×8 Dadda multiplier

Dadda multipliers have been described by L. Dadda in 1965. They also belong to the column compression family of multipliers like the Wallace tree multiplier. The Dadda multiplier compresses columns differently from the Wallace-tree multiplier. Dadda observed that the maximum compression factor possible was 3/2, which is the compression factor of the full adder. Thus fewer columns are compressed in the first few steps of the column compression tree, and more columns in the later levels of the multiplier. At each stage just enough compressors are used to obtain depth equal to the next number from the sequence above. Dadda multiplier uses a minimal number of (3:2) and (2:2) adders at each level during the compression to achieve the required reduction. The dot diagram for an 8×8 Dadda multiplier is shown in Figure 3.10. This multiplier requires 64 AND gates, 35 full adders and 7 half adders. This multiplier requires fewer adders during the compression stage than Wallace tree multiplier. The result of this compression scheme for the partial product matrix is that the delay of the structure is nearly equal to that of the Wallace tree multiplier, but with a smaller area.

Table 3.8 Area requirements of 8-bit Dadda multiplier

Hardware Requirements	Area (sq. µm)	Quantity	Total Area (sq.µm)
full adder	62.18	35	2176.3
Half adder	17.84	7	124.88
AND Gate	8.82	64	564.48
Net Area (sq.µm)			2865.66

Area requirement of 8-bit Dadda multiplier is calculated as given in Table 3.8. The list of hardware and the area allocated by each hardware element is given in the table. From the table it is observed that the area occupied by 8-bit Dadda multiplier is 2865.66 square µm. The delay time of this type of multiplier is 1.428ns, which is due to the critical path of 12 adders.

Table 3.9 Synthesis results of multiplier structures

Multiplier Design	**Area in (square µm)**	**Min. Clock delay (ns)**
Array multiplier 8-bit	4488	3.35
Wallace tree 8-bit	4170	2.42
Dadda multiplier 8-bit	3558	2.17

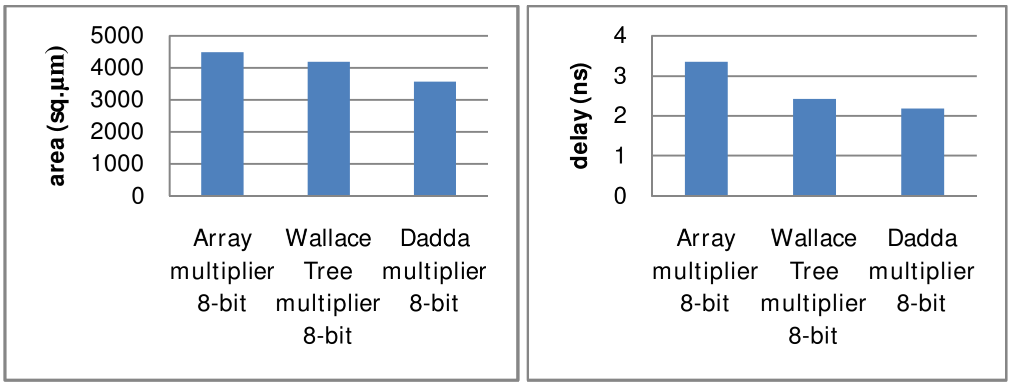


Figure 3.11 Area and speed comparisons of 8-bit Array, Wallace Tree and Dadda multipliers

The multiplier structures discussed above have been coded in Verilog, and synthesized in a 0.18 micron standard CMOS cell library using Leonardo Spectrum of Mentor Graphics Corp., and the results have been given in Table 3.9. From the comparison it is observed that the Dadda multiplier is faster than the Wallace tree multiplier of the same bit-width, possibly due to the lower fan-outs of intermediate signals. The smaller area of the Dadda multiplier is obtained from a reduced number and size of compressors. So, the Dadda tree outperforms the Wallace scheme in both area and timing. The Dadda multiplier is also faster and smaller than all other multiplier schemes examined above of the same bit-width. The area-wise comparison of multiplier designs have been given in Figure 3.11. The smallest multiplier by area is the Dadda multiplier with ripple carry adder, which is used in the proposed architecture for multiplication.

3.3.5 Interconnect Lines

The interconnect lines, as shown in Figure 3.12, are configured to route data between the computational units of PEs, between the different type of PEs and between the PESs. Nearest-neighbour communication is as simple as it sounds in this architecture. Looking at an array of PESs, one can see that the only need in this neighbourhood are input and output connections in each direction i.e, north, south, east, and west. This allows each PES to communicate directly with each of its immediate neighbours.

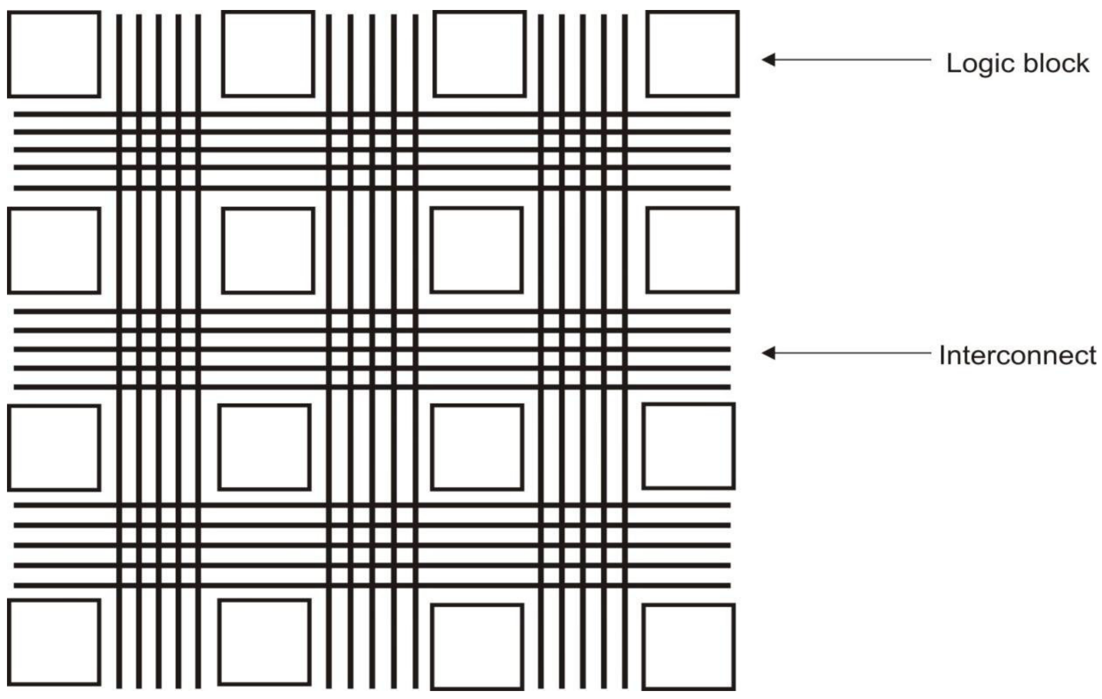


Figure 3.12 Interconnect lines between PESs

In addition, each PES can be connected to any PES of its nearest column. Each PES accesses nearby PES and its resources through the connection block. The connection block connects the logic block input and output terminals to routing resources through programmable switches or multiplexers. The connection block allows logic block inputs and outputs to be assigned to arbitrary horizontal and vertical tracks, and thereby increases the routing flexibility. In the most general sense, it is simply a matrix of programmable switches that allow a signal on a track to connect to another track. Depending on the design of the switch block, this connection could be, for example, to turn the corner in either direction or to continue straight.

3.3.6 I/O Block

I/O Blocks are programmable, which allow designs inside the architecture to configure a single interface pin as input, output or bidirectional. This I/O Block used for external communications has buffers to store the received pre-processed data, and then to deliver out the data.

3.3.7 Memory Block

The memory block is a storage device used during RAKE receiver operation of WCDMA technology and FFT operation of OFDM based WLAN standard. It has a register file that can store 64 16-bit data used for storing spreading codes of the RAKE receiver and 32 16-bit twiddle factor coefficients of FFT. Each RC has an internal memory in order to load the intermediate data used by that RC instead of loading them on the memory block. For applications where each RC has a different set of input (and/or output) data, being able to quickly load or unload the internal RAM will have a significant impact on the system performance.

The memory block contains context cache which stores the context words to determine the way the PESs, register files, interconnect lines, demultiplexers and multiplexers are configured. It stores the whole configuration responsible for setting up the architecture in order to execute the different types of applications. Context Caches distributed inside the PESs are used for the fast reconfiguration of this architecture. Context cache stores a few configuration contexts locally and can be loaded on a cycle-by-cycle basis. Configuration contexts can alternatively be loaded from the configuration memory at the cost of extra delay, in the case when the context cache is not large enough to store configuration. Transferring the configuration contexts from the configuration memory to the PESs' Context Caches is realized with the use of a dedicated set of buses. Each of them can be shared either among PESs residing in the same row or in the same column.

3.3.8 Controlling Block

The controlling block is responsible for generating control signals of the local memories, multiplexers, interconnect lines and PESs. The controller can be shared across multiple PESs which reduce the overall hardware complexity. The host processor communicates with the controller through user registers to initiate the controller.

3.4 SUMMARY

This chapter has presented a new reconfigurable architecture, for flexible system solutions in adaptive air interface standards such as WCDMA and OFDM based WiFi standards. This chapter has described the reconfigurable components of the architecture. The various designs of adder and multiplier have been compared and selected the optimal design. All other blocks of the architecture have also been described in this chapter.