CHAPTER 6

RESULTS AND DISCUSSION

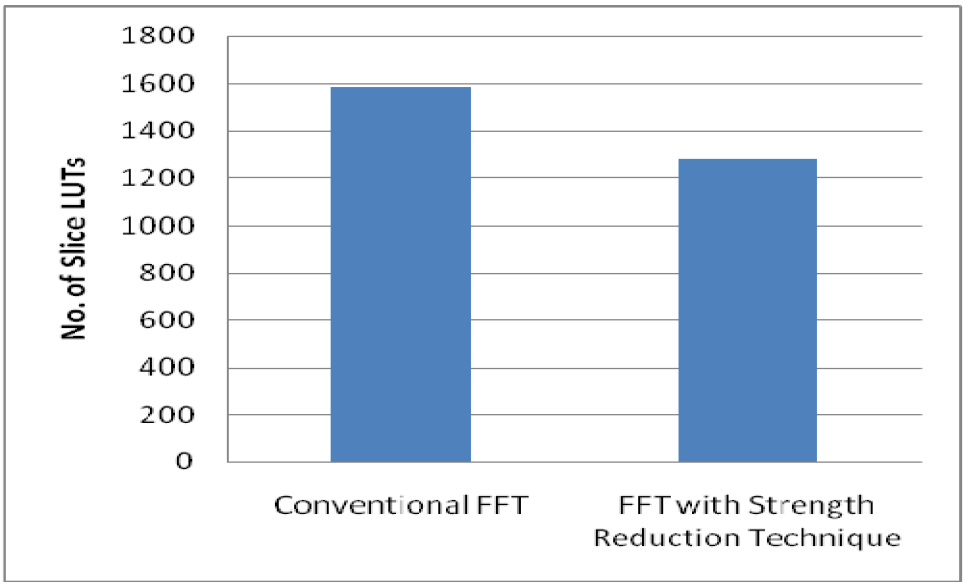
The proposed architectures described in Sections 4, 5 and 6 have been simulated using ModelSimSE v6.5, coded in VHDL and mapped onto a Virtex-6 FPGA device (xq6vlx130t-rf784) with speed grade (-2) using the tool Xilinx ISE 12.2 and synthesized (Xilinx Inc. 2010). The metrics extracted from Xilinx ISE after synthesis and implementation are,

(1) Number of used slice LUTs and
(2) Gate count.

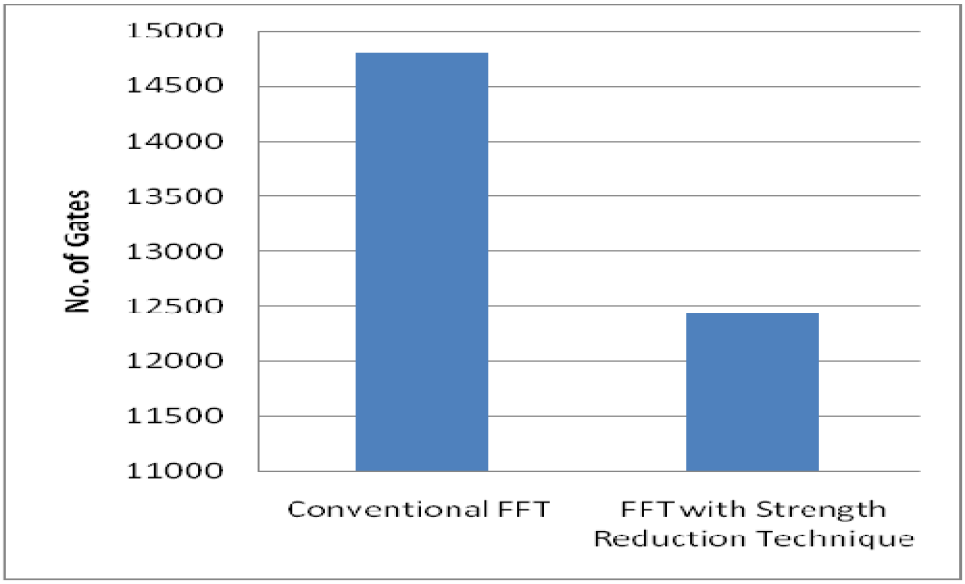
Table 6.1 shows the implementation results of the optimized FFT implementation which uses algorithmic strength reduction transformation technique. These results are obtained using the design technique explained in Chapter 4. The comparison results are given in Figure 6.1. These results illustrate that the algorithmic strength reduction transformation technique of FFT algorithm has significant improvements in terms of area saving compared to the conventional FFT implementation. On average, the optimized approach shows an improvement about 19.27% in terms of number of slice LUTs used and 15.93% in terms of number of gates used, as illustrated in Table 6.5.

Table 6.1 Resource utilization of conventional FFT and FFT with strength reduction transformation technique

Resources Utilized	Conventional FFT	FFT with Strength Reduction Transformation Technique
Number of Slice LUTs	1588 out of 80000	1282 out of 80000
Gate Count	14796	12440



(a) Comparison of the slice LUTs used



(b) Comparison of the no. of gates utilized

Figure 6.1 Comparison results of resources utilized by conventional FFT and FFT with strength reduction transformation technique

In OFDM based WiFi standard FFT block receives data every 4µs for the duration of 3.2µs in serial manner. The input data slots are stored in the memory every 4 µs and the 64 point FFT module fetches the data from the memory as soon as the computation of 64 point FFT for a particular data slot is completed. Time delay of 64 point FFT is only 40.16ns which is within the acceptable range.

64 point FFT using strength reduction technique is compared with the existing techniques in terms of no. of slices. From the comparison of the implementation results as shown in Table 6.2, the total slices of the proposed FFT implementation is 60% less than the $R2^2$ 64-point FFT proposed in Sukhsawas and Benkrid (2004) and 57% less than the $R2^4$ 64-point FFT proposed in Hang Liu and Hanho Lee (2008).

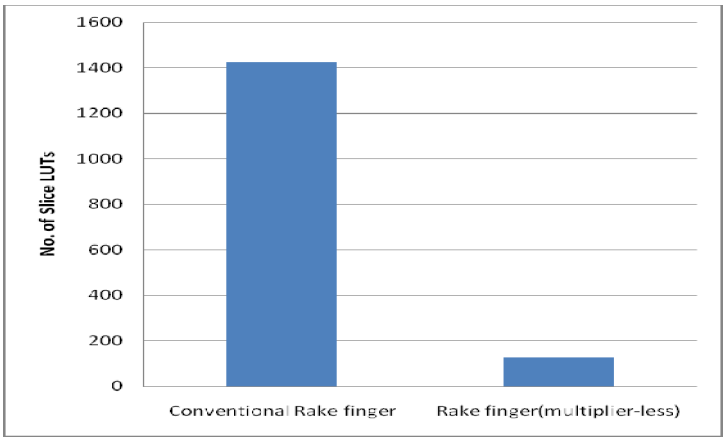
Table 6.2 Resource comparison of 64 point FFT architectures

Resource utilized	FFT using strength reduction technique	$R2^4$ FFT	$R2^2$ FFT
No. of Slices	1282	3017	3470

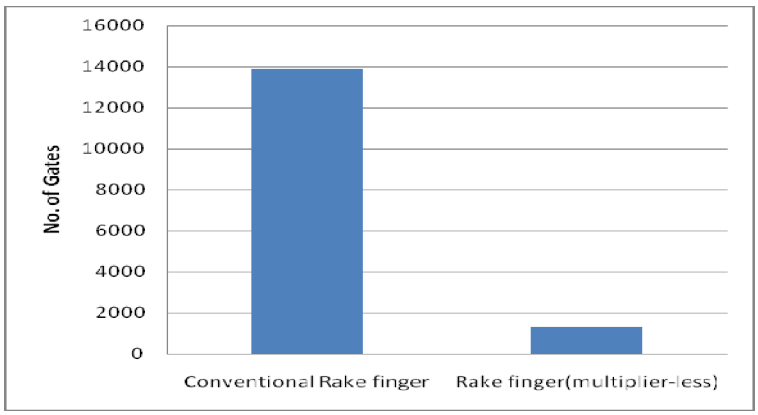
Table 6.3 and Figure 6.2 show the comparison of the results of the optimized multiplier-less RAKE receiver implementation and conventional RAKE receiver implementation. From this result it is clear that the proposed approach outperforms the conventional RAKE receiver in terms of the number of slice LUTs used and the number of gates utilized. On average, the optimized approach shows an improvement about 91.03% and 90.45% in terms of number of slice LUTs used and the number of gates utilized, as illustrate in Table 6.5.

Table 6.3 Resource utilization of conventional RAKE receiver and multiplier-less RAKE receiver

Resources Utilized	Conventional RAKE receiver	RAKE receiver (multiplier-less)
Number of Slice LUTs	1426 out of 80000	128 out of 80000
Gate Count	13898	1328



(a) Comparison of the slice LUTs used



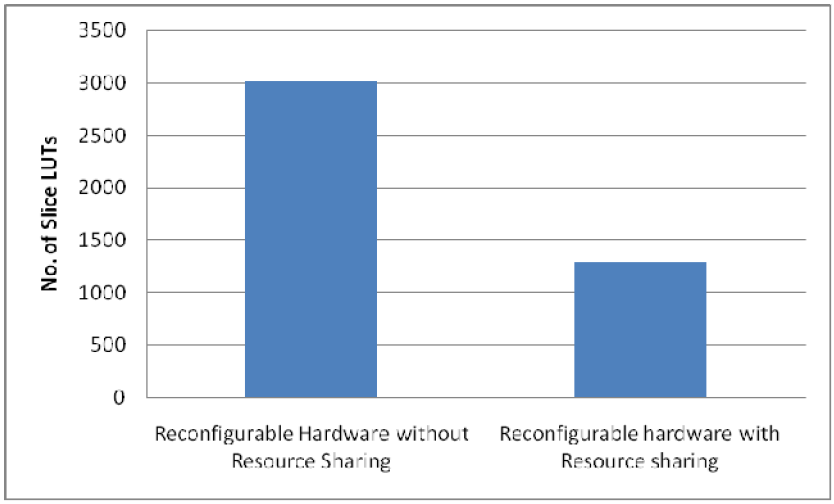
(b) Comparison of the No. of gates utilized

Figure 6.2 Comparison of the percentage of resources utilized by conventional RAKE receiver and RAKE receiver (multiplier-less)

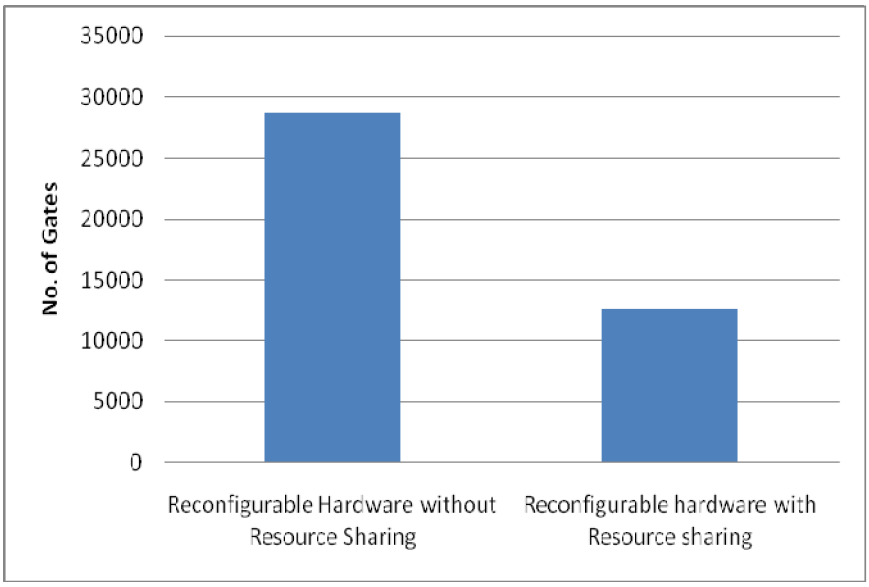
Finally implementation results of the proposed reconfigurable architecture with resource sharing and reconfigurable architecture without resource sharing techniques in terms of the number of slice LUTs used, the number of gates utilized and the critical path delay(in ns) are listed in Table 6.4.

Table 6.4 Resource utilization of reconfigurable architecture without and with resource sharing

Resources Utilized	Reconfigurable Architecture without Resource Sharing	Reconfigurable Architecture with Resource sharing
Number of Slice LUTs	3014 out of 80000	1290 out of 80000
Gate Count	28694	12540
Delay in ns	36.30	40.16



(a) Comparison of the slice LUTs used (continued)



(b) Comparison of the No. of gates utilized

Figure 6.3 Comparison of the percentage of resources utilized by reconfigurable architecture without resource sharing and with resource sharing

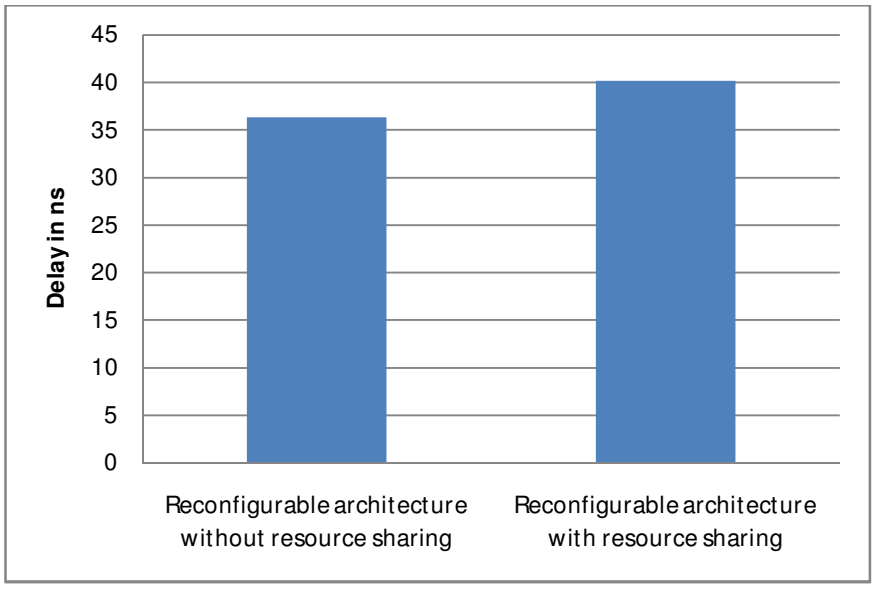


Figure 6.4 Comparison of delay(in ns) of reconfigurable architecture without resource sharing and with resource sharing

Figure 6.3 and 6.4 show the comparison of the proposed architecture (reconfigurable architecture with resource sharing) and the reconfigurable architecture without resource sharing in terms of the number of slice LUTs used, the number of gates utilized and the delay(in ns). Figure 6.4 indicates that the proposed architecture shows a slight increase in delay of 9.4% compared with reconfigurable architecture without resource sharing which is optimal. But average reductions of the computational resources in terms of slice LUTs and no. of gates are 57.2% and 56.3% respectively in the proposed architecture, as shown in Table 6.5. From this result it is clear that the proposed resource sharing reconfigurable architecture is better than the standard reconfigurable architecture.

Table 6.5 Comparison of results summary of the conventional architecture and the proposed method targeting Virtex-6 FPGA

Proposed method	Slice LUTs saving	Gate Reduction
FFT with Strength Reduction Transformation Technique	19.27%	15.93%
Rake finger (multiplier_less)	91.03%	90.45%
Reconfigurable Hardware with Resource Sharing	57.2%	56.3%

From the results presented earlier it seems that there is a significant reduction in large number of computational resources which forms the proposed architecture which is more efficient than the conventional architecture. These efficient realizations require fewer FPGA resources, so not only reduce the area, but also the total power consumption as described below.

Table 6.6 shows the summary of estimated power of the conventional reconfigurable architecture (without resource sharing method) and the proposed reconfigurable architecture (with resource sharing method). These results are obtained using Xilinx PlanAhead 12.2 tool. The power performance of the two types of architecture is illustrated in Figure 6.5.

Table 6.6 Power comparison of conventional architecture and proposed architecture

method	Logic Power in mW
Reconfigurable Architecture without Resource Sharing (Conventional)	5.7
Reconfigurable Architecture with Resource Sharing (Proposed)	2.6
Power saving	54.4%

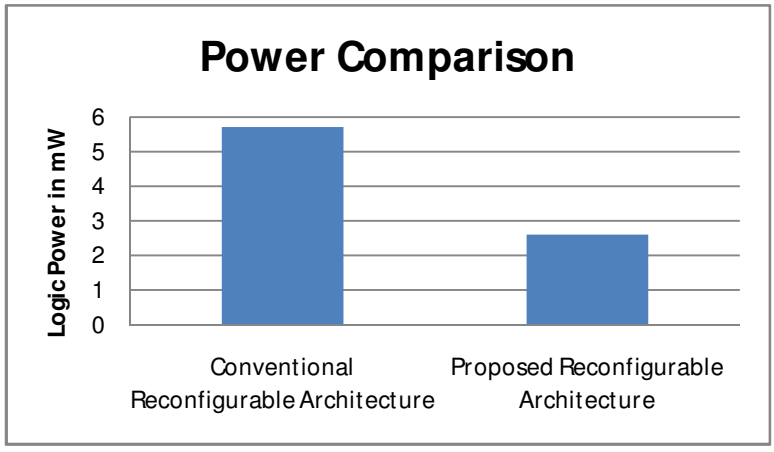


Figure 6.5 Logic power comparison of proposed reconfigurable architecture (with resource sharing) and conventional architecture (without resource sharing)

From Table 6.6, it is shown that the proposed architecture saves 54.4% of power by resource sharing method compared with the conventional

architecture. The major source of power saving is the reduction of multipliers in the architecture. The implementation results confirm the salient aspect of the design; it reduces area as well as the power consumption.

Table 6.7 Quantitative comparison between the proposed and existing architecture

Architecture	Area (# slices)
Proposed	1290
ADRES	2570
Montium	2850
Reconfigurable Architecture [Lasse Harju and Jari Nurmi (2005)]	1796

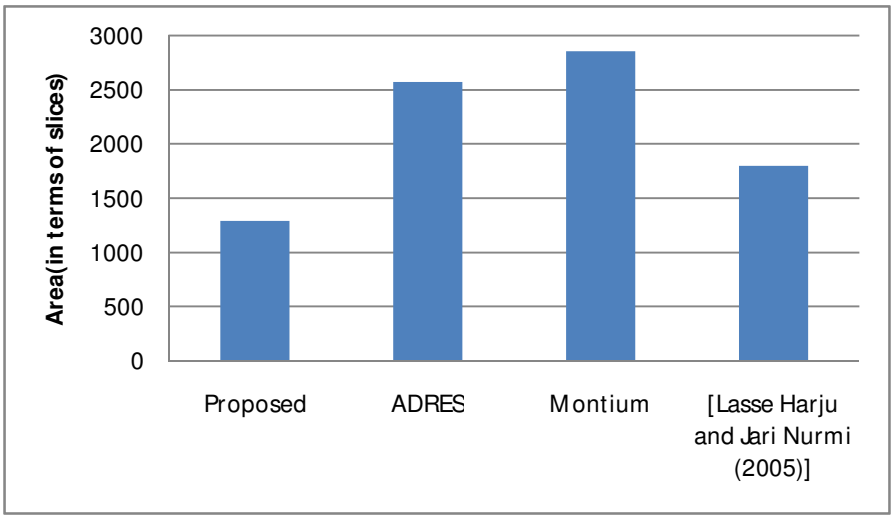


Figure 6.6 Comparison of the proposed and existing reconfigurable architectures

The proposed reconfigurable architecture has been quantitatively compared with few of the coarse-grained architectures presented in Chapter 2, for RAKE finger and FFT implementation. The comparative results are reported in Table 6.7 and illustrated in Figure 6.6. The second column of Table 6.7 shows the area utilization (in terms of slices) of ADRES architecture and that of Montium architecture. The proposed architecture has

also been compared with reconfigurable architecture presented by Lasse Harju and Juri Nurmi (2005) and the slices utilized is given in Table 6.7.

In comparison with the existing architectures such as ADRES, Montium and Lasse Harju and Jai Nurmi (2005), the proposed architecture delivers better results in terms of computational resources utilized. The hardware complexity of the proposed architecture is 49.81% smaller than ADRES, 54.75% smaller than Montium, and 28.18% smaller than Lasse Harju and Jai Nurmi (2005) .