CHAPTER 5

HARDWARE REALIZATION OF PROTOCOLS

5.1 INTRODUCTION

This chapter addresses FPGA implementation of DSDV, DSR, AODV and TORA protocols. The design flow that starts from hand coding in VHDL to generating configuration data to be downloaded into RTS FPGA is explained. The design, implementation and results of the sub-modules are reported. The chapter discusses certain optimizations that lead to better performance in terms of space and time. Finally, all the individual modules of routing protocols in RTS FPGA and ATC FPGA as constituents of routing section are simulated and the results are obtained. The basic idea is to have a single routing protocol in the routing section on silicon at a given time, while sensing for any changes in the network throughput. In the subsequent sections the phrases like DSDV router, DSR router, AODV router and TORA router mean the routing section is currently embedded with that routing protocol in the node architecture. Here the node means source or destination and router node or intermediate node means the relaying nodes. The router or routing section is the same with a routing protocol, which is a part of node architecture. They are interchangeably used in this chapter.

5.2 RTS ARCHITECTURE USING DSDV

Figure 5.1 gives the architecture and the basic blocks of a MANET node’s routing section with DSDV routing protocol integrated inside. Initially the route request is initiated by the node (source), which needs to transmit
data packets. The Routing Section (RS) of the source node should have the Reconfiguring Target Section (RTS) with a specific protocol to continue the route finding. The selection of the routing protocol at source node is DSDV (default). But the routing protocol to be used by the routing node depends on the information contained in the route request broadcast packets. Consequently, the protocols may be changed to any of the routing protocol in the intermediate nodes. The routing section of node should have all the routing protocols in EEPROM in bit form so that any protocol that is needed can be configured in the routing section using RTS FPGA. The following section describes how DSDV protocol is implemented in RTS FPGA and corresponding DSDV-JTAG file is obtained for future storing in EEPROM.

The working procedure of DSDV protocol was explained in Chapter 1. RTS section architecture for DSDV protocol includes the various memory structures, DSDV router processor, input and output buffers for temporary storage and the control unit that helps to coordinate the various processes, etc.

![RTS architecture using DSDV protocol](image)

Figure 5.1 RTS architecture using DSDV protocol
The input control information from other nodes (that is transmitted by every node every second periodically) is received and stored in this node’s input buffer (Data-link In Buffer). The output control information that is waiting for transmission to the other nodes is stored in the node’s output buffer (Data-Link Out Buffer). The input from Data-Link In Buffer is processed by DSDV processor to obtain the routing information, namely, Destination-ID, Next Hop-ID, Metric(s) (Hop count), Sequence number and Install Time which are to be stored in the Route Information Memory (RIM).

The output control information that is to be transmitted to other nodes is stored in Data-Link out Buffer. The output control information that is to be transmitted in the form of a table contains the following information:

Destination id of the nodes (6 bits), originator source id (which is the IP address of the node itself) (6 bits), number of hops (Metric) to reach each destination from that node (6 bits) and sequence number of the individual nodes (8 bits).

The route information memory stores the information that is required to determine the next hop address for a given destination. The information fields present in the Route Information Memory are as follows:

Destination ID of the nodes; next hop ID for each destination; hop metric; sequence number and installation time, etc.

5.2.1 DSDV Processor

The architectural model of DSDV processor is shown in Figure 5.2. The processor includes the following entities, namely, control unit, input processor, output processor, sysclock and stale node processor.
The input processor processes the information stored in the data-link in buffer and the processed routing information is transferred and stored in the route information memory (RIM). When the input is received in the data link input buffer, the control unit generates interrupt and waits until processes are completed (OUP = 0 and IUP = 0) and after completion of the processes it issues iprdy = 1 (Input Processor Ready) signal to the input processor. The input processor then sets IUP = 1 which is an acknowledgement to the control unit and handles the process. Once the process is completed the IUP is again set to 0 as an acknowledgement to the control unit.

The output processor processes the information stored in Route Information Memory (RIM) and the processed information is transferred to data link output buffer. The update process, which is the periodic transmission of routing table information, is carried out once in every second. An interrupt (timer_int from the entity SysClock) is generated by the control unit when the timer counts one second, and the Control Unit waits until processes are completed (OUP = 0 and IUP =0). After completion of processes, it issues OUP = 1 signal to the Output Processor. Then the control unit issues
insproc = 1 (Input to stale node processor) signal to the stale node processor. The stale node processor then acknowledges the control unit by setting InstUP = 1 and handles the process. This processor calculates the difference in time between present and previous updating time of every node, which is called DEL value. If DEL value exceeds 5, then the corresponding metric for that particular node is set to infinite (“1111111”). When the process is complete the InstUp is again set to 0 as an acknowledgement to the control unit.

In order to provide protocols with high speed and less cost, hardware implementation is used with the help of Finite State Machine (FSM) model. At a low level of abstraction, a protocol is often most easily understood as a state machine. Design criteria can also easily be expressed in terms of desirable or undesirable protocol states and state transitions. FSM modeling of DSDV based MANET router is shown in Figure 5.3, which reveals the three basic operations. These processes are: 1. Table transmission (periodic update), 2. Receive Input and 3. Check input for the presence of stale nodes periodically.

![State diagram for control unit](image-url)
Whenever an input interrupt (IP_INTERRUPT) is detected, the input information has to be processed (IP_RECV_PROCESS) and once the process is completed the control unit is acknowledged by IP_ACK. The update process involves periodic transmission of routing table information (TABLE_TXN_PROCESS) for every 1 second. These control signals are handled by TIMER_INTERRUPT and OP_ACK. Using these processes we carried out VHDL modeling and verified using FPGA.

5.2.2 Simulation and Synthesis Report

The various blocks of DSDV architecture is coded using VHDL tool and the programs are simulated using Modelsim. The simulation result is shown in Figure 5.4. Using the simulation, the hardware functionality is verified by giving various inputs to the program.

![Figure 5.4 Simulation result of DSDV protocol](image-url)
The synthesis report, device utilization summary and the path delay are found using Xilinx project navigator. The result of RTL view is shown in Figure 5.5.

Figure 5.5 RTL view of DSDV protocol

Timing Summary:

Minimum period : 7.832 ns

Maximum Frequency : 127.685 MHz

Minimum input arrival time before clock: 10.835 ns

Maximum output arrival time after clock: 8.234 ns
Device utilization summary:

Selected Device: v812ebg560-8

Number of Slices : 4377 out of 9408  47%
Number of Slice Flip Flops : 8513 out of 18816  45%
Number of 4 input LUTs : 11834 out of 18816  62%
Number used as logic : 11814
Number used as Shift registers : 20
Number of IOs : 73
Number of bonded IOBs : 37 out of 404  9%
IOB Flip Flops : 61
Number of GCLKs : 4 out of 4  100%

5.3 RTS ARCHITECTURE USING DSR

Dynamic Source Routing (DSR) is one of the more generally accepted reactive ad-hoc routing protocols. As the name indicates, it utilizes source-based routing rather than table-based and source-initiated rather than hop-by-hop. When a node wishes to establish a route, or issues a route request to all of its neighbors, each neighbor rebroadcasts this request, adding its own address in the header of the route request packet. When the request is received by the destination or by a node with a route to the destination, a route reply is generated and sent back to the sender along with the addresses accumulated in the request header. The responsibility for assessing the status of a route falls to each node in the route. Each node must ensure that packets successfully cross the link to the next node. If it does not receive an acknowledgement, it reports the error back to the source, and leaves it to the source to establish a new route. While this process could use up a lot of bandwidth, DSR gives each node a route cache for them to use aggressively to reduce the number of
control messages sent. If it has a cache entry for any destination request received, it uses the cached copy rather than forwarding the request as shown in Figure 5.6.

In addition, it promiscuously listens to other control messages for additional routing data to add to the cache. DSR has the advantage that no routing tables need to be kept to route a given data packet, since the entire route is contained in the packet header. The primary disadvantages are two-fold. DSR is not scalable to large networks. The internet draft acknowledges that the protocol assumes that the diameter of the network is no greater than 10 hops. This means that the hop metric in the route header in the data packet should not exceed 10.

![RTS architecture using DSR protocol](image)

**Figure 5.6 RTS architecture using DSR protocol**

Additionally, DSR requires significantly more processing resources than most of the other protocols. In order to obtain routing information, each node must spend much more time in processing any control data it receives, even if it is not the intended recipient. Collectively DSR protocol consists of two main functions such as route discovery and route maintenance.
5.3.1 DSR Routing Processor and Other Modules

In the finite state machine model, DSR protocol can be represented using the following states: idle state, route check, send data, route discovery, error check, send acknowledgement and route reply. These states are represented with their functionality as arguments.

5.3.2 FSM Modeling of DSR Protocol

The DSR protocol as shown in Figure 5.7 consists of all FSM states along with FIFO and Cache memories. If there is a data packet to transfer, the route check state checks the availability of the route in the cache memory. If the route is available, data packet is transmitted in the send data state, else route discovery process is initiated by broadcasting route request packets with source id, destination id, and request id to the neighbor nodes until it reaches the destination node.

![Finite state machine model of DSR protocol](image)

**Figure 5.7** Finite state machine model of DSR protocol
The route request packet is stored in a table to avoid repeated route discovery. If the route discovery time exceeds, then the node goes to the idle state. The route reply state writes a route reply in a piggyback manner. The discovered route is stored in the cache memory and then the data packet is transmitted with header. If an error occurs in the route during data transmission, the intermediate node sends a route error signal and simultaneously deletes cached route and gives the alternate route if it is available. If the data packet reaches the destination properly, the receiver node sends an acknowledgement signal to the source node.

5.3.3 Design of Cache and FIFO Memory

Cache and FIFO memories are designed and used for each individual node. Cache memory stores the routes, which are discovered in on-demand basis. As the system is operated by battery, there is a trade off between power consumption and cache memory size. Due to this power consumption problem, cache memory size should be optimized. Power optimization not only saves the power but also provides better load balancing. The size of cache memory depends on the number of nodes in the system. This is shown in equation (5.5) of FIFO memory design.

FIFO memory is used as a buffer to store the data packet and hop count with time count when the nodes are busy. The greater is the size of FIFO memory, the lower is the packet loss. There is a trade off between quality of service of the system and FIFO memory.

In a network of N nodes, normally multiple routes are stored in the cache memory for a given destination in DSR algorithm. The number of routes $N_R$ to be stored at a particular source node can be given by,
\[ N_R = (\text{Number of destinations}) \times (\text{Number of disjoint routes}) \]
\[ = N_d \times N_j \tag{5.1} \]

where \( N_j = N - 1 \)

\[ N_R = (N-1) \times (N-1) \tag{5.2} \]

Memory size of a single route \( M_s \) is given by,

\[ M_s = \text{Number of Hop counts} \times \text{Number of bits required to represent a node} \]
\[ = N_h \times N_b \]
\[ = ((N/2) + 1) \times \log_2 N \tag{5.3} \]

Size of Cache memory

\[ M_c = N_R \times M_s \tag{5.4} \]

From equation (5.2) and (5.3),

\[ M_c = (N-1) \times (N-1) \times ((N/2) + 1) \times \log_2 N \tag{5.5} \]

In FIFO memory, the number of data packet storage depends on the requirement of efficiency of packet transmission. In this design the size of FIFO is taken as 10.

Size of FIFO memory = Number of data packet to be stored * data packet size.

5.3.4 Simulation and Synthesis Report

DSR protocol blocks are modeled using VHDL and the simulation is done using ModelSIM. The simulation result is shown in Figure 5.8. FPGA
The implementation procedure for this DSR protocol is the same as the previous procedure used for DSDV protocol.

**Figure 5.8 Simulation result of DSR protocol**

The synthesis of the routing protocol is done using Xilinx Project navigator. Figure 5.9 shows the synthesized RTL view of DSR protocol showing input, output terminal and control signals. The synthesized result is obtained as JTAG file. This is a stream of bits which has to be stored in a section of EEPROM. When these JTAG files in EEPROM are fed to RTS FPGA, all the devices and circuits rearrange themselves for the specified protocol (DSR). Finally, using the Xilinx project navigator the device utilization summary report and timing summary report are obtained.
Figure 5.9  RTL view of DSR protocol

Timing Summary:

Minimum period : 10.132 ns
Maximum Frequency : 98.695 MHz
Minimum input arrival time before clock : 9.984 ns
Maximum output required time after clock : 7.783 ns

Device utilization summary:

Selected Device : v812ebg560-8
Number of Slices : 945 out of 9408  7%
Number of Slice Flip Flops : 1232 out of 18816  5%
Number of 4 input LUTs : 1041 out of 18816  5%
Number of IOs : 42
Number of bonded IOBs : 42 out of 404  17%
Number of GCLKs : 1 out of 4  25%
5.4 RTS ARCHITECTURE USING AODV

In this section AODV protocol implementation is described. Procedure for this implementation is also like other two protocols described previously. The basic functional block diagram of AODV routing protocol is shown in Figure 5.10. Initially the route is to be obtained by any node (source), which needs to transmit data packets.

![Diagram of AODV architecture]

**Figure 5.10 RTS architecture using AODV protocol**

There is a buffer memory provided to store the incoming message and the addressing packets like RREQ, RREP, RERR and RREQ_ACK. There is a cache memory, which is used to store the addresses of all the other nodes of the network for which the route is already known. The incoming address packets are compared with the information available in the cache memory and the route is selected based on a perfect match after comparison. After formation of the route, message transfer takes place between the nodes. Route maintenance is done by constant exchange of HELLO message packets.
5.4.1 Routing Processor and Other Modules

The following points are taken into consideration while drawing the finite state machine that is shown in Figure 5.11. The initial state was taken to be the idle state where the node is in rest condition without any transaction but is logged to the network. A node can be source, destination, or an intermediate node and this condition is taken care of at the beginning.

![Finite state machine model of AODV Protocol](image)

Figure 5.11 Finite state machine model of AODV Protocol

As a source, the node starts the route request process by sending route request packet (RREQ) and waits for time duration for a route reply packet (RREP) to occur from the destination or from the intermediate node.
As a router node (intermediate node), it accepts the incoming RREQ packet and simultaneously sends a RREP to source and forwards RREQ to the destination.

As a destination, it receives RREQ either from source or intermediate node and sends a RREP message packet back to the source confirming the formation of the route. In the case of a breakage in the link, the node, which experiences the breakage, informs the source about it using the Route ERRor (RERR) message packet and hence a new route finding is initiated.

Absence of route to the destination is informed to the source by the intermediate nodes. The route maintenance is done by timely exchange of hello message packets between active route members.

### 5.4.2 AODV Memory Design

The memory design includes the two major memory units, namely, FIFO memory for storing information about actual data content and cache memory for storing information about the routes including the addresses of various nodes. Cache and FIFO memories are designed for each individual node.

Cache memory stores the route addresses, which are discovered using on demand basis. The size of cache memory depends on the number of nodes in the system. Cache memory is designed as follows:

Let size of one routing table entry = R bytes

Let the number of entries (previously available Routes) for each destination = A routes

Storage space for each destination = A *R bytes
Total number of possible destinations = (N-1)

Therefore total memory space required = (N-1) * A * R bytes

The size of FIFO memory depends upon the number of data packets to be stored and hop count with time count during route discovery process. The number of data packet storage depends on the requirement of efficiency of packet transmission without much delay. The total FIFO memory size is determined as follows:

Let the maximum number of data packets transmitted from a node be M Packets

Let size of a message packet = L bytes

Storage space required at 3 types of nodes:

Source = M*L bytes
Intermediate Node = (N-1)* M*L bytes
Destination = (N-1) *M*L bytes
Total Space required = (2N-1) *M*L bytes

5.4.3 Simulation and Synthesis Results of AODV

The various blocks of AODV protocol are coded using VHDL. The codes are simulated using ModelSIM. The procedure for this AODV protocol is the same as the previous procedure used for other two protocols already discussed.

The synthesis of the routing protocol is done using Xilinx Project navigator. The following are the simulation and synthesized reports of AODV protocol. The simulation result is shown in Figure 5.12 and RTL View of AODV Protocol is given in Figure 5.13.
Figure 5.12 Simulation result of AODV protocol

Figure 5.13 RTL View of AODV protocol
Timing Summary:

Minimum period : 9.132 ns

Maximum Frequency : 109.505 MHz

Minimum input arrival time before clock : 7.984 ns

Maximum output required time after clock: 5.783 ns

Device utilization summary:

Selected Device : v812ebg560-8

Number of Slices : 745 out of 9408    7%

Number of Slice Flip Flops : 1077 out of 18816   5%

Number of 4 input LUTs : 942 out of 18816     5%

Number of IOs : 69

Number of bonded IOBs : 69 out of 404         17%

Number of GCLKs : 1 out of 4               25%

5.5 RTS ARCHITECTURE USING TORA

TORA protocol is a reactive MANET routing protocol, which is explained in Chapter 1. It includes the various memory structures, main processor, buffers for temporary storage and the control signals that help to coordinate the various processes.

Figure 5.14 gives the architecture of the node with TORA. The input buffer is used to store and forward the packets received by TORA processor. The processor performs the functions according to the state modeling and updates the neighbor status memory and the link status memory. After the processing, the packet to be broadcast is sent to the output buffer and then to the network.
The input buffer is designed according to the maximum size of the packet to be received. In this case it is designed to be a two dimensional array of 6 rows of 8 bits each. At the initiation of the read control signal it reads the input port of the node and receives the packets.

5.5.1 Routing Processor and Other Modules

The three basic functions performed by RTS include: 1. Creating routes 2. Maintaining routes 3. Erasing routes. The protocol uses three distinct control packets to perform these three functions, namely, query (QRY) for creating routes, update (UPD) for both creating and maintaining routes and clears (CLR) for erasing routes. Figure 5.15 shows the control signals for maintaining route. In Figure 5.15, the various control signals represent various actions in the protocol implementation as follows, lk_fl – link failure, ck_pt – checkpoint, ref_lev – reference level, org_ref – original reference levels, etc.
Figure 5.15 FSM representation of control signals for route maintenance

This process is invoked by the function called UPDATE which is used to either create or maintain the route. When there is no downstream, it checks for the link failures first. In case there is a failure, it changes its height metrics correspondingly and generates an update packet for transmission and goes to idle state. If there is no link failure then it checks the reference level of all the neighbors. If they are not equal then it propagates the update packet and goes to the idle state. When the reference levels are equal, it checks whether the packet received is a reflection. If it is not a reflected one then the update packet is sent after necessary changes. If it is a reflected one, it checks for the original reference level and confirms that there is a link partition and sends the clear packet. If there is no original reference level detected, then it generates an update packet and transmits it.
The update packet is assumed to be consisting of the following details of the height metrics as shown in Figure 5.16. It is a two dimensional packet of 6 rows of 8 bits each. The version is the same as that of the query packet. The type field with three bits is assumed to be ‘010’ for the update packet and like wise for others. The reserved bits are for future use. The second row gives the address of the destination. The third and the subsequent rows give the height metrics of the sender of the update packet.

- HEIGHT.tau - Time the reference level was created.
- HEIGHT.oid - Unique id of the router that created the reference level.
- HEIGHT.r - Flag indicating if it is a reflected reference level.
- HEIGHT.delta - Value used in propagation of a reference level.
- HEIGHT.id - Unique id of the router to which the height metric refers.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
<td>4</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Version #</td>
<td>Type</td>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destination IP Address</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H. tau</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H. oid</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H. r</td>
<td>H. delta</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H. id</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.16 The structure of update packet of TORA
The routing processor is the main component that performs the operations on the packet elements. It reads every bit field of the packet received and checks for series of conditions as per TORA algorithm. This is represented as a series of comparators and multiplexers in the hardware. It also simultaneously updates the neighbor status memory and the link status memory. After the processing of the packet is over, it updates the packet with new values and passes it onto the output buffer and anticipates the arrival of the next packet.

The neighbor status memory shown in Table 5.1 is a two-dimensional array which is used to store the details about the height metrics of the various neighbor nodes. It is updated and the new values are retrieved whenever a new packet is received. At last the updates in this memory give the route required for the destination.

Table 5.1 Neighbor status memory

<table>
<thead>
<tr>
<th>Neigh Array</th>
<th>H. tan</th>
<th>H. vid</th>
<th>H. r</th>
<th>H. delta</th>
<th>H. id</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neigh(0)</td>
<td>0000000</td>
<td>0000000</td>
<td>00</td>
<td>000010</td>
<td>00000011</td>
</tr>
<tr>
<td>Neigh(1)</td>
<td>0000000</td>
<td>0000000</td>
<td>00</td>
<td>000001</td>
<td>00000111</td>
</tr>
<tr>
<td>Neigh(2)</td>
<td>0000000</td>
<td>0000000</td>
<td>00</td>
<td>000011</td>
<td>00001111</td>
</tr>
<tr>
<td>Neigh(3)</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

The link status memory is again a two dimensional array which is used to store the details regarding the linkage with the adjacent neighbor. A value of ‘1’ represents the presence of upstream and the value ‘0’ represents downstream while the absence of any wireless link is represented by a high impedance state. This memory is also updated and looked for whenever a route is searched. Table 5.2 gives an example of link status memory.
The output buffer consists of a similar configuration as that of the input buffer. The output buffer receives the new packet from the router processor only after processing and transmission of the previous packet to the network.

There are two control signals used in this configuration, namely, read and write. The read signal is used to indicate the input buffer so that it can now read the input port and store the packet temporarily. The processor to inform the neighbors that it is going to transmit the new packet to the network invokes the write signal. In effect the write signal of this node is connected to the read signal of the adjacent node.

### 5.5.2 Simulation and Synthesis Report of TORA

The simulation of TORA protocol is done using ModelSim after coding the various blocks of TORA architecture. By giving various inputs to the architecture, the functions of each block are verified. The simulation result is shown in Figure 5.17. The synthesis of the above program is done using the Xilinx Project navigator. The circuit containing RTS FPGA is able to support the routing section of a single node with an extendable memory for more neighbors. RTL view of TORA routing protocol is shown in Figure 5.18. By using this report, the speed of operation and device utilization are listed.

### Table 5.2 Link status memory

<table>
<thead>
<tr>
<th>Link Array</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link(0)</td>
<td>1</td>
</tr>
<tr>
<td>Link(1)</td>
<td>0</td>
</tr>
<tr>
<td>Link(2)</td>
<td>1</td>
</tr>
<tr>
<td>Link(3)</td>
<td>null</td>
</tr>
</tbody>
</table>

0 - Downstream
1 - Upstream
Figure 5.17 Simulation result of TORA protocol

Figure 5.18 RTL view of TORA protocol
Timing Summary:

Minimum period : 26.155 ns
Maximum Frequency : 38.234 MHz
Minimum input arrival time before clock : 24.15 ns
Maximum output required time after clock : 21.125 ns

Device utilization summary:

Selected Device : v812ebg560-8
Number of Slices : 1252 out of 9408 13%
Number of Slice Flip Flops : 376 out of 18816 1%
Number of 4 input LUTs : 2327 out of 18816 12%
Number of IOs : 27
Number of bonded IOBs : 27 out of 404 6%
Number of GCLKs : 3 out of 4 75%

The maximum frequency of the clock that can be used in the circuit is given by 38.234 MHz. In other words, the minimum time for period T is given by 26.155 ns.

5.6 COMPARISON OF FOUR PROTOCOLS

The four protocols were modeled using VHDL and the simulation using ModelSim tool verified their functionalities. Then VHDL models of four protocols were synthesized using Xilinx tool and the device utilization and timing summaries were obtained as shown in a nutshell in Tables 5.3 and 5.4 respectively. In the device utilization summary, the components like slices and slice flip flops are mentioned. The programmable element inside a FPGA is called CLB (Configurable Logic Block). If looked inside a FPGA, CLB
consists of LUTs (Look up Table) and Flip-Flops/Latches. Each CLB contains 2 flip-flops. In the device utilization, the number of slices used refers to number of CLBs and the slice flip flop refers to the total number of flip-flops used. This summary gives the overall usage of the device to implement the design.

Table 5.3 Device utilization summary

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Slices used (AREA)</th>
<th>Flip flops slices used</th>
<th>LUT’s used</th>
<th>IOB’s used</th>
<th>GCLK’s used</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSDV</td>
<td>4377</td>
<td>8513</td>
<td>11834</td>
<td>37</td>
<td>4</td>
</tr>
<tr>
<td>DSR</td>
<td>945</td>
<td>1232</td>
<td>1041</td>
<td>42</td>
<td>1</td>
</tr>
<tr>
<td>AODV</td>
<td>745</td>
<td>1077</td>
<td>942</td>
<td>69</td>
<td>1</td>
</tr>
<tr>
<td>TORA</td>
<td>1252</td>
<td>376</td>
<td>2327</td>
<td>27</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 5.4 Timing report of ARR

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Min input arrival time before clock (ns)</th>
<th>Max output required time after clock (ns)</th>
<th>Max frequency (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSDV</td>
<td>10.835</td>
<td>8.234</td>
<td>127.685</td>
</tr>
<tr>
<td>DSR</td>
<td>9.984</td>
<td>7.783</td>
<td>98.695</td>
</tr>
<tr>
<td>AODV</td>
<td>7.984</td>
<td>5.783</td>
<td>109.505</td>
</tr>
<tr>
<td>TORA</td>
<td>24.15</td>
<td>21.125</td>
<td>38.234</td>
</tr>
</tbody>
</table>

The device utilization summary of all the four protocols is used to select FPGA, which has the maximum capacity to accommodate any of the four protocols at one time. Then from the timing summary, the maximum
frequency of clock operation necessary is found to be 127 MHz. From Figure 5.19 it is found that AODV protocol has low area utilization and moderate frequency of operation. Finally, the synthesized result will provide JTAG files for each protocol implementation and they are permanently stored into four locations of EEPROM as shown in Figure 4.3.

![AREA AND FREQUENCY](image)

Figure 5.19 Maximum frequency and area comparison of the four protocols

5.7 SIMULATION OF ATC CONTROL ON ROUTING SECTION

The integration of ATC, EEPROM, SRAM and Reconfiguring Routing Section will provide the MANET node with ARR routing capability. The simulation is carried out using ModelSim. To simulate a condition of variable rate of packet arrival at the destination node, four packet-generating sources are modeled with different packet generation rates. Also changes in
the packet arrival rate are simulated artificially as if some packets are lost and are carried out by shifting to different packet generating sources. The simulated waveform results obtained for ARR with ATC and other routing protocols are shown in Figure 5.20.

The simulation results reveal that whenever the throughput plunges down to a value lower than the threshold, a new protocol in the suite is downloaded into RTS FPGA. The report shows the occurrence of change of protocols from DSDV to DSR, AODV and TORA in a round Robin fashion. This procedure has been explained in the ATC algorithm. As per the proposed ATC algorithm, when the initial throughput obtained is 40, the threshold is kept at 20 packets. In the subsequent run of simulation, whenever the throughput of a particular protocol is above the threshold, the same protocol is continued for transmission of the rest of the data packets. If the throughput falls below the threshold value of 20 packets, as it can be seen from the simulation, switching of protocol is effected. Monitoring and comparing of the throughput with threshold value are continuously carried out. Likewise, other protocols are also invoked as and when they are needed to maintain the throughput.
Figure 5.20 Simulation of ATC on RS using the four protocols

Figure 5.20 shows the occurrence of change of protocols from DSDV to DSR when throughput does not satisfy the condition of 20 packets as threshold. The change of protocol occurs at the instances of throughput values of 4, 4, 16 and 4 packets. These values are lesser than throughput threshold value of 20. The changeover is visualized at enabling or activation of EEPROM to release JTAG files using different addresses... e.g. 00 to 01, 10 and 11 etc. Likewise the protocols are switched to the best protocol, which provides throughput greater than threshold value so that it will always be higher. The prime objective of this research contribution was to achieve higher and consistent throughput in a changing network environment. It is demonstrated through simulation, that ARR routing technique provided better throughput of around 20 packets by changeover of protocols compared to conventional routing technique. The presented simulation results also validate our claim that multiprotocol routing provides better performance compared to a single protocol approach.
During the switching among these protocols, it can further be observed in the simulation; the data transmission is completely stopped and resumed only when new route is obtained using the next protocol in the order. This procedure is necessary because no packet should be processed during the protocol configuration in source node as well as in the relaying nodes.

Once it is discovered that the full cycle of new protocols {i.e. from DSDV to TORA} in line are not able to satisfy the throughput condition, a new threshold value is fixed. The present threshold value is obtained by subtracting 10% of T0 from memory (0) (threshold value) and new threshold value is set for future comparison, which becomes 17. Now the next round of protocol reconfiguration starts from default protocol DSDV. At times in case of throughput value becoming zero, the changeover of routing protocol is stopped. This is because the network may have encountered a disintegration problem or complete failure of wireless links to the destination node or failure of destination node itself. In this case, the whole process stops for a while and resumes after some time.

5.8 CONCLUDING REMARKS

This chapter discusses in depth FPGA implementation of DSDV, DSR, AODV and TORA protocols and their simulation using ATC algorithm. The design flow that starts from hand coding in VHDL to generating configuration data to download onto FPGA is explained. The design, implementation and results of the sub-modules are reported. The implementation brought out the device utilization ratio, comparison of the usage of various stages of flip-flops, slices, look-up tables and the clocks used. During the implementation, certain optimizations are carried out that lead to better performance in terms of space and time. Subsequently, all the individual modules of routing protocols and ATC are integrated in the routing section of node architecture. The simulation of routing section as a whole reveals in its ModelSim graph that whenever the packet
throughput falls below 20 of the threshold value, the protocol is swapped in turn to bring back the throughput to higher level. The idea here is to have a single routing protocol on silicon at a given time and also sensing any changes in the network throughput. During the change over of protocols from to one to another, the time consumed is called swapping time delay and will affect throughput. Swapping time between protocols also include the route finding time of the new protocol and therefore network throughput as a whole will be affected since throughput is the rate of receipt of data packets with respect to unit time. Though there is a slight reduction in throughput due to this delay, overall effect of throughput enhancement due to swapping to better protocol is an added advantage.