Universal Scheduler for AFDX Based on BAG Concept

Sanyogita Sathe[1], Ch. Praveen Kumar[2]

[1]M.Tech (VLSI Design),, GITAM University, Hyderabad, India

[2]Assistant Professor, ECE Department, GITAM University, Hyderabad, India

Abstract: ***Avionic Full-Duplex Switched Ethernet (AFDX), is a specification for a deterministic aircraft data network bus for aeronautical, railway and military systems. The network is based on standard IEEE 802.3 Ethernet technology. AFDX extends the Ethernet standard by adding Quality of Service (QoS) and deterministic behavior with a guaranteed dedicated bandwidth. An AFDX network consists of so called End Systems and switches. An End System is a component connected to the AFDX network and capable of handling all AFDX related protocol operations. One or more switches, depending on the network hierarchy, are located on the data path between two End Systems. The point-to-point and point-to-multipoint connections are represented by virtual links (VL). The aim of the paper is to develop Transmitter scheduling Mechanism using the concept of BAG in verilog.***

Keywords: AFDX, standard IEEE 802.3, end system, virtual links, BAG

1. Introduction

A typical AFDX network generally consists of avionics subsystems; interconnect networks and source/destination end systems. Specifically, the avionics subsystems include traditional on-board aircraft system, such as global position system (GPS) and flight control system (FCS). The interconnect networks use a full-duplex switched Ethernet consisting of links and switches to support data exchange among different avionics systems. The end systems actually serve as an interface between the subsystems and the interconnect networks to guarantee real-time and reliable data transmission by using deterministic Virtual Links (VLs). A virtual link constructs a virtual communication connection from one source end system to one or more destination end systems, forming a mono-sender multicast path. According to the AFDX specification, we can identify a VL by setting its available 16-bit ID, Bandwidth Allocation Gap(BAG) and the largest length of VL frames (i.e., *Lmax*),where the BAG represents the minimum interval between two consecutive frames sent to a VL.

An AFDX network specifies the BAG duration from 1ms to 128ms to serve as the bandwidth control mechanism for virtual links. In order to guarantee transmission reliability in an AFDX network, one of the most important characteristics is the redundant management on virtual links.

2. Block Diagram

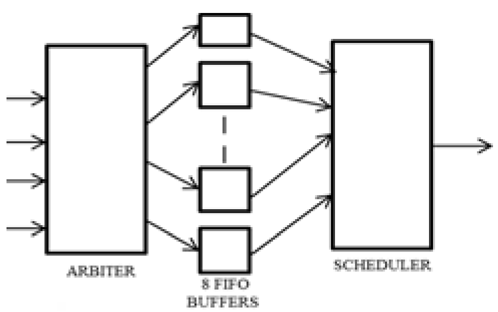

Figure 1: Block Diagram

3. Working Methodology

3.1 Arbiter

The arbiter module has been coded using case statements. Each case corresponds to a separate bag value. There is a round robin based algorithm running within the arbiter module that checks each incoming signal one after the other by cycling through each one. If an incoming signal is detected, then the arbiter receives the signal, separates the frame into two i.e. BAG frame and request frame. The bag frame is 8 bits long and the request frame is 24 bits long. BAG value can be anything from 20 to 28 in binary form. The BAG frame is forwarded to another system that receives will tell the processor which output signal corresponds to which BAG. The request frame is passed onto the buffer module.

3.1.1 Inputs – Arbiter Module

Frm1 – 32'b input from first incoming Virtual Link
Frm2 – 32'b input from second incoming Virtual Link
Frm3 – 32'b input from third incoming Virtual Link
Frm4 – 32'b input from fourth incoming Virtual Link

3.1.2 Outputs – Arbiter Module

Out1, Out2, Out3, Out4, Out5, Out6, Out7, Out8 – Wires that connect as inputs to each of the buffers
Bag1, Bag2, Bag3, Bag4 – Corresponding bag values from each of the inputs frm1 to frm4 sent to the concerned module
Write1, Write2, Write3, Write4, Write5, Write6, Write7, Write8 – Write flags to activate write operation on the buffers

3.1.3 Algorithm

1. Check for "rst" condition. If "rst == 1" then all wires and registers are reset to zero condition
2. If "rst == 0" then check inputs frm1 to frm4 for input value in Round Robin fashion
3. Separate BAG value (Bits 31 to 24) from VL ID (Bits 23 to 0)
4. if(cntr == 2'b00) // Counter that points to each input port begin

```
case(bag1)
8'b00000001: // One of the possible BAG values
begin
out1 <=frm1[23:0]; // transmit VL ID to corresponding buffer
end
```

5. Repeat preceding code for all cases (BAG values)

3.2 FIFO buffer

Once the data is stored in the buffer, it waits for the scheduler module to request it by sending a read flag. The read flag, like the write flag is a single bit, that when high, will allow the buffer to send forward any packets of data stored within it. Once the read flag is high, the read pointer starts rising from the bottom. The buffer checks if there is any data stored at the address that is being pointed at by the read pointer and if the value is not 0, it sends the message forward. The read pointer runs until the difference between the write and read pointers (ptr_gap) is 0. This means that the read pointer has reached the last input message.

This operation of the buffer is based on FIFO, or first in first out concept. Since the data sent to the buffer first will be sent forward first to the scheduler module. There are a total of 8 buffers operating within the TOP module. Each one corresponds to a separate BAG value. This sort of operation sorts all incoming packets according to their BAG and since from a particular buffer all the data will go forward at the same time this means that all the data with one particular BAG value that comes in within one cycle of the scheduler will be sent forward as requests at the same time. Additional spaces for messages can be added to the buffers at any time.

3.2.1 Inputs – Buffer Module

Datain – connected to 'out' from arbiter. Each buffer has one datain and named datain1 to datain8 for the 8 buffers
Write – Flag sent by the arbiter to activate write operation for a particular buffer
Read – Flag sent by the scheduler to activate read operation for a particular buffer

3.2.2 Outputs – Buffer Module

Dataout – connected to input of scheduler module. Each buffer has one dataout named dataout1 to dataout8 for the 8 buffers.5

3.2.3 Registers – Buffer Module

'i' – This register stores the 24'b input that is input and is cross referenced before writing to the buffer every time.

3.2.4 Algorithm

1. If 'rst == 0' Check if "write_to_stack" i.e. the write flag is high
2. Receive 24 bit VL ID from arbiter
3. Write at location "write_ptr" is pointing using the following code

```
else if(write_to_stack&& (i!=datain)) // if write flag high and same data has not already been received
begin
stack[write_ptr] <= datain; // write at write_ptr location
write_ptr<= write_ptr+3'b001; // increment write_ptr
ptr_gap<= ptr_gap+3'b001; // increment ptr_gap
 i<=datain; // save new value in 'i' register
end
```

4. Check if "read_from_stack" i.e. the read flag is high
5. Read value from buffer and send forward using the following code

```
 if(read_from_stack) // if read flag is high
begin
dataout<= stack[read_ptr]; // transfer data to round robin
 stack[read_ptr] <= 24'bx; // revert stack value to don't care condition
 read_ptr<= read_ptr+3'b001; // increment read pointer
 ptr_gap<= ptr_gap-3'b001; // decrement write pointer
end
```

3.3 Round-Robin Scheduler

The round robin scheduler consists of a two level counter – one that counts from 0 to 255 (ctr) and the other, depending on the top level counter, changes from 000 to 111. This is named ptr. It operates in such a way that 000 corresponds to buffer 1 (1 ms BAG), 001 to buffer 2 (2 ms BAG) and so on to 111, which points to buffer 8 (128 ms BAG). The pointer increments by 1 for each of the 31 bits that the counter (ctr) points to. For each buffer that the pointer (ctr) points to, the buffer simultaneously sends the value forward to the redundancy management system which then sends it forward as a request to the processor as a request.

3.3.1 Inputs – Scheduler

datain1 to datain8 – These are connected to dataout1 to dataout8 that are outputs from the buffer respectively

3.3.2 Outputs – Scheduler

Dataoutt – output for the overall system that serially sends the received message packets in the required format.
Read1 to read8 – Read signals that correspond to each buffer from 1 to 8 and activate the read operation in the buffer when received

3.3.3 Algorithm

1. If condition "rst == 1" is true, then reset all values to zeros
2. If condition "rst == 0" is true, then start the counter "ctr"
3. Run the following loop to check if there are any incoming values

```
else if((ctr<= 8'b11111111) && (ptr == 3'b000)) // If ctr< 255, start with ptr == 000(first buffer)
begin
 dataoutt<= datain1; //forward data from first buffer to output
 if(ctr == 8'b00100000) // if counter has reached upper limit for first buffer, enter loop
 begin
 ptr<= 3'b001; // change ptr to point at second buffer
 read1 <= 0; // change read flag for buffer 1 to 0
 read2 <= 1; // change read flag for buffer 2 to 1
 end
 end
```

4. Repeat loops for all buffers till buffer no. 8
5. Repeat entire cycle in round robin fashion

4. TOP Module

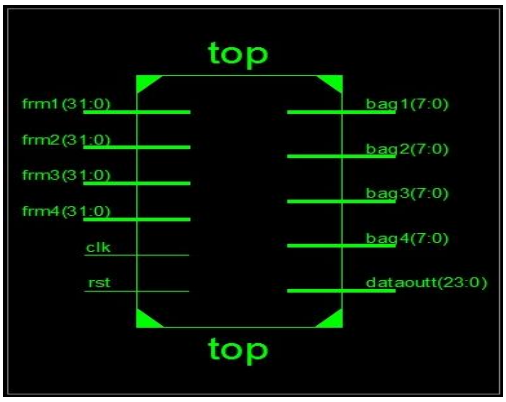

Figure 2: RTL Schematic of TOP module

4.1 Inputs – Top Module

Frm1 – 32'b input from first incoming Virtual Link
Frm2 – 32'b input from second incoming Virtual Link
Frm3 – 32'b input from third incoming Virtual Link
Frm4 – 32'b input from fourth incoming Virtual Link
clk – Gives a clock input
rst – Reset high and low condition

4.2 Outputs- Top module

Bag1 – 8'b bag value separated from frm1
Bag2 – 8'b bag value separated from frm2
Bag3 – 8'b bag value separated from frm3
Bag4 – 8'b bag value separated from frm4
Dataoutt – 24'b output signal in serial form that is forwarded

5. Results and RTL Schematics

Figure 3: TOP module

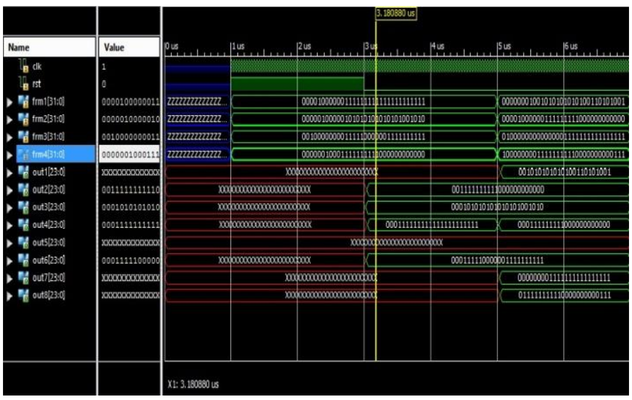

Figure 4: Arbiter

Figure 5: FIFO buffer

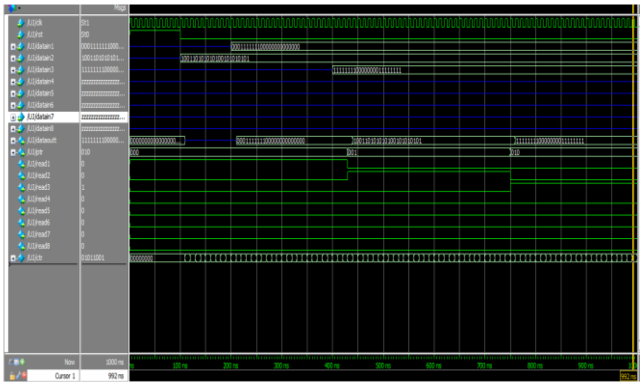

Figure 6: Scheduler

6. Synthesis Report

Target Device: XC5VLX330T

# RAMs	8	24-bit register	41
8x24-bit dual-port RAM	8	3-bit register	1
# Adders/ Subtracors	26	8-bit register	10
24-bit adder	16	# Multiplexers	90
24-bit addsub	8	1-bit 8-to-1 multiplexer	8
3-bit adder	1	24-bit 2-to-1 multiplexer	72
8-bit adder	1	24-bit 8-to-1 multiplexer	9
# Registers	68	8-bit 8-to-1 multiplexer	1
1-bit register	16	# FSMs	1

7. Conclusion

A scheduler that is optimized for the purpose of real time internal communication was successfully designed using Xilinx ISE tools and implemented in Spartan Kit. Building on the existing concepts already used within AFDX, and optimizing the pre-existing scheduler, a new scheduler design using both FIFO and Weighted Round Robin techniques was developed. The design concentrates on the advantages and strengths of both algorithms while minimizing data loss or delays from the weaknesses of both algorithms. Other than avionics, this scheduler can be used for any generalized purposes that require operations in real time environments.

References

[1] Condor Engineering; "AFDX Protocol Tutorial", May 2005
[2] Yu Hua; Xue Liu; "Scheduling Design and Analysis for End-to-End Heterogeneous Flows in an Avionics Network", IEEE INFOCOM 2011

[3] ManolisKatevenis; StefanosSidiropoulos; Costas Courcoubetis, "Weighted round-robin cell multiplexing in a general-purpose ATM switch chip", IEEE Journal on Selected Areas in Communications, (Vol. 9, Issue:8), October 1991
[4] H.S. Behera; Simpi Patel; Bijayalakshmi Panda, "A New Dynamic Round Robin and SRTN Algorithm with Variable Original Time Slice and Intelligent Time Slice for Soft Real Time Systems", International Journal of Computer Applications (0975 – 8887) Volume 16– No.1, February 2011
[5] Xin Chen; Xudong Xiang; Jianxiong Wan, "A Software Implementation of AFDX End System", 2009 International Conference on New Trends in Information and Service Science
[6] A. Mifdaoui, F. Frances, C. Fraboul, "Real-Time Communication over Switched Ethernet for Military Applications" In Proceedings of the conference on Emerging network experiment and technology, pp. 45-56, ACM, 2005.
[7] AEEC; Aeronautical Radio Inc.; "Aircraft Data Network Part 7 Avionics Full Duplex Switched Ethernet Network (ARINC SPECIFICATION 664 P7-1)", September 23, 2009
[8] Jonathan Woodruff, "Deficit Round Robin Input Arbiter for NetFPGA"
[9] M. Shreedharand G. Varghese. Efficient fair queuing using deficit round-robin. IEEE/ACM Transactions on Networking (TON), 4(3):385, 1996.