International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2015): 6.391

Project (HIDA)-High Speed Communication Interface for Distributed Avionics Design of Switching Software for the Implementation AFDX Protocol

J. Victor Paul

School of Electronics and Communication, Veltech Dr RR & Dr SR Technical University, Avadi-600062, Chennai, Tamil Nadu, India

Abstract: ***With the rapid development of advanced data transmission technologies in avionics AFDX (Avionics Full Duplex Switched Ethernet) shows great potential in on-board avionics system because of its high bandwidth and real-time property. In this paper, we present a feasible software framework to implement AFDX protocol on VxWorks Operating System. With the rapid development of advanced data transmission technologies in avionics fields, AFDX (Avionics Full Duplex Switched Ethernet) shows great potential in on-board avionics system because of its high bandwidth and real-time property. In this paper we design switching software using the linux kernel on the freescale power architecture processor for the implementation of filtering and policing logics on the AFDX protocols. The simulative result proves that the framework works effectively with appropriate performance and meets the real-time requirements of avionics data transmission.***

Keywords: AFDX, End systems, VL scheduling, BAG, jitter

1. Introduction

As one of the most advanced data transmission technologies in avionics field, AFDX (Avionics Full Duplex Switched. Ethernet) features high bandwidth, low cost, extensive system integration, real-time property, reliability and shows great potential in on-board avionics system. The performance of AFDX protocol has been thoroughly studied and implemented.

In this paper, we first makes a brief overview of AFDX protocol stack and its determinism, then further Concentrate on the design of switching software for the T1040 Reference design board for the end system using the configured linux kernel and using the freescale open source standard development kit to the implementation of filtering and policing logics on the AFDX protocols and testing them with different parameters.

2. AFDX Protocol

2.1 Introduction

Aircraft Data Networks (ADN) primarily utilizes the ARINC 429 standard. This standard, developed over thirty years ago and still widely used today, has proven to be highly reliable in safety critical applications. ARINC 429 networks, which can be found on a variety of aircraft from both Boeing and Airbus, utilize a unidirectional bus with a single transmitter and up to twenty receivers. A data word consists of 32 bits communicated over a twisted pair cable. There are two speeds of transmission: high speed operates at 100 Kbit/s and low speed operates at 12.5 Kbit/s.

Another standard, ARINC 629, introduced by Boeing for the 777 aircraft provides increased data speeds of up to 2 Mbit/s and allowing a maximum of 120 data terminals. ARINC 629 network operates without the use of a bus controller, thereby increasing the reliability of the network architecture. One of the primary draw-backs of this network type is that it is very specific to civil aircraft applications, requiring custom hardware which can add significant cost and development time to the aircraft.

ARINC 664 Part 7 [1] is defined as the next generation aircraft data network (AFDX). It is based on IEEE 802.3 Ethernet, enabling greater potential use of Commercial Off-The-Shelf (COTS) hardware, thereby reducing aircraft cost and development time. AFDX was developed by Airbus Industries for the A380, has since been accepted by Boeing and used on the Boeing 787 Dreamliner, and is being used or considered today for other applications.

2.2 AFDX Overview

Typically, an AFDX network is a star topology of end systems that are connected through a centralized AFDX switch. By utilizing this form of network structure, AFDX is able to significantly reduce wire runs on the aircraft when compared to ARINC 429 or 629 as examples, thus reducing overall aircraft weight. Additionally, AFDX provides dual link redundancy and Quality of Service (QoS).To support more end systems, AFDX switches can be cascaded to form a larger network with a more complicated network topology. AFDX protocol comprises five layers, as shown in Figure 1.

Figure 1: OSI model of AFDX

End System

An AFDX end system [1] provides services to achieve deterministic communication with other end systems. It guarantees a secure and reliable data exchange on the AFDX network. A guaranteed service provides a deterministic (mathematically provable) worst case on end system to end system frame transit delay over the AFDX network. In order to guarantee a fixed transit delay, bandwidth is managed at the link level through logical communication channels called Virtual Links (VL). Each VL is responsible of guaranteeing its Ethernet frame flow based on its allocated bandwidth.

In order for a network application to use the end system, it transmits its data to communication ports residing on the end system. These ports constitute the entry and exit points of data to or from the end system. In transmission, the end system uses a traffic shaping function to manage configured VLs based on at least one VL parameter: the Bandwidth Allocation Gap (BAG); The BAG configures the frequency at which transmission can occur.

An AFDX compliant end system manages outgoing traffic before sending data over the Ethernet network. This is to guarantee a deterministic behavior on each transmit VL. An end system is equipped with two Ethernet adapters, referred to as "Network Adapter A" and "Network Adapter B", which are respectively connected to AFDX Network A and AFDX Network B. Communication on the physical network is dependent on the VL's configuration. When using both adapters, the VL uses redundancy, meaning it transmits the same frame over both networks simultaneously. Each end system is identified by a unique identifier composed of the Equipment Domain, Side and Location. This identifier is used for Ethernet and IP addressing to connect to different end systems during the virtual link transmission for the successful communication

Virtual Links

A virtual link (VL) describes a logical unidirectional "point to multipoint" communication channel between end systems over an AFDX Network, see Figure 3. Each VL is configured with a fixed bandwidth. The Bandwidth Allocation Gap (BAG) and the Lmax are the AFDX parameters responsible of allocating the bandwidth for a given VL. Virtual links describe network communication at the Ethernet level.

More precisely, each VL has a unique identifier that is used in the destination MAC address of an Ethernet frame. This address uses the Ethernet multicast format. As for the MAC source address field, it contains the Ethernet adapters configured address.

Traffic Shaping

Each VL is allocated some bandwidth that is managed by the scheduler according to the Bandwidth Allocation Gap (BAG), which specifies the frequency at which a VL can transmit on the network(s). The traffic shaping function (scheduler) is the core AFDX entity, providing regulated and deterministic traffic flow to the AFDX network.

When a contention occurs between two or more VLs, this induces scheduling latency, referred to as jitter. According to the standard, jitter must not exceed 500 microseconds at any time. A jitter management mechanism can be used to minimize or eliminate jitter effects Traffic shaping is only performed in transmission. In a transmitting end system with multiple VLs, the scheduler multiplexes the different flows coming from the regulators, as illustrated in Figure 2

At the output of the scheduler, for a given VL, frames can appear in a bounded time interval. This interval is defined as the maximum admissible jitter. Scheduling jitter occurs when multiple VLs are to be scheduled at the same time. This situation happens when BAG values coincide with each other. In the perspective of one VL, the jitter effect can be illustrated in Figure 3.

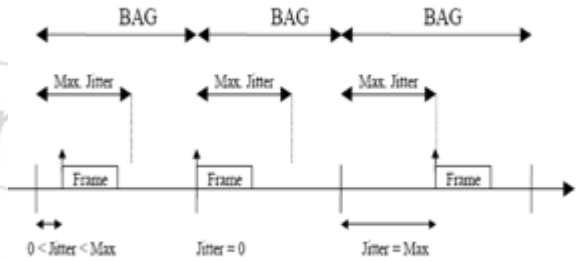

Figure 2: The Jitter Effect for a Maximum Bandwidth Data Flow

Actual traffic shaping is thus performed on a per VL basis. The scheduler uses each VL's Bandwidth Allocation Gap (BAG) and shapes the flow of frames so that no more than one frame is transmitted in each interval of BAG milliseconds. Using a BAG scheduler provides a logical isolation with respect to available bandwidth among VLs it supports. Regardless of the attempted utilization of a VL by one application, the allocated Bandwidth on any other VL is unaffected. The maximum usable bandwidth of each VL is characterized by its BAG and its authorized Lmax (maximum VL frame size). The maximum usable bandwidth = Lmax / BAG in Kbytes per second. The end system should accommodate VL frames up to the maximum Ethernet frame size of 1518 bytes, or as specified by its Lmax, in both transmission and reception.

Integrity Checking

The integrity checking function is responsible for maintaining the ordering of data as delivered by the application, for both transmission and reception (ordinal integrity). In transmission, the integrity checking function adds, per VL, a Sequence Number (SN) for each transmitted frame on the AFDX network. The frame SN is one byte long with a range of 0 to 255. The frame SN is incremented by one for each consecutive frame of the same VL and wraps around to 1 following the value 255.

On the receiving side, under fault-free network operation, the integrity checking function simply passes the frames that it has received on to the redundancy management function (see next section), independently for each network. If there are faults (based on sequence number), the integrity checking function

has the task of eliminating invalid frames, and informing the network management function accordingly. The integrity checking function is configurable on a per VL basis.

Redundancy Management

Redundancy is used as a fall-back mechanism in case of a failing or damaged network. When data becomes unreliable on one network, that same data can be received from the second network if the transmitting VL uses both networks. When both networks are fine, some sort of logic needs to be applied to discard the second copy of a frame.

3. Design of Switching Software for the AFDX switch

The project undertaken is for the design of switching software for the 10 port AFDX switch having ISL ports(Inter switch Linking port-where Virtual link can be exchanged between any ports) using the freescale T1020 network power pc. The basic theme of the project is the User space datapath acceleration architecture where it is a software framework that allows Linux user space applications to directly access the portals in a high-performance manner. This session provides an overview of USDPAA. Topics include USDPAA threads, queue manager and buffer manager drivers, and example applications. So before implementing the automated setup a manual setup is prepared where the freescale T1040 reference design board is used to implement the AFDX protocols consists of fman and L2switch(Responsible for the transmitting of vl's between end systems). The T1040 RDB has the 4-core e5500 power core, the design board is ported with u-boot from the host PC to the board (Target) for initiating the board, the Linux kernel 12.2 is ported on the board which is responsible for the memory management and other processor related functions, device tree block which consists of the layout of the total hardware and the Ethernet interfaces. In this host to target interface the(The manual test setup) Linux host PC is connected to the target board through the TEP port(Echoing port) which loads the configuration file on to the target board and another windows host machine is connected to the end systems(n in number taking 3 end systems in this setup) where it loads configuration file to the respective end systems where during the test the end system configuration file and the switch side configuration files are compared and accordingly the test plan is carried out for different parameter's.

3.1 T1040 Reference Design board

The QorIQ T1040 Reference Design Board (T1040RDB) is a high-performance computing evaluation, development and test platform supporting the QorIQ T1040/20, T1042/22 and T2081 processors built on Power Architecture® technology. The board, with its 1.4 GHz T1040 processor and rich I/O mix, is intended for evaluation of the QorIQ T1 family of processors in networking and Ethernet-centric applications, such as mixed control and data plane in fixed routers, switches, Internet access devices, firewall and other packet filtering applications, as well as general-purpose embedded computing. The board consists of 1 TEP port 2 Fman (Frame manager ports) ports and 8 L2(Layer 2 switch) switch ports. Accesses DPAA through hardware components called portals. USDPAA is a software framework that permits Linux user space applications to directly access the DPAA queue manager and buffer manager software portals in a high performance manner. Here in USDPAA there are Queuing manager (QMan), Buffer manager (Bman) which is responsible for input to the frame manager for processing the frames and storing the processed frames.

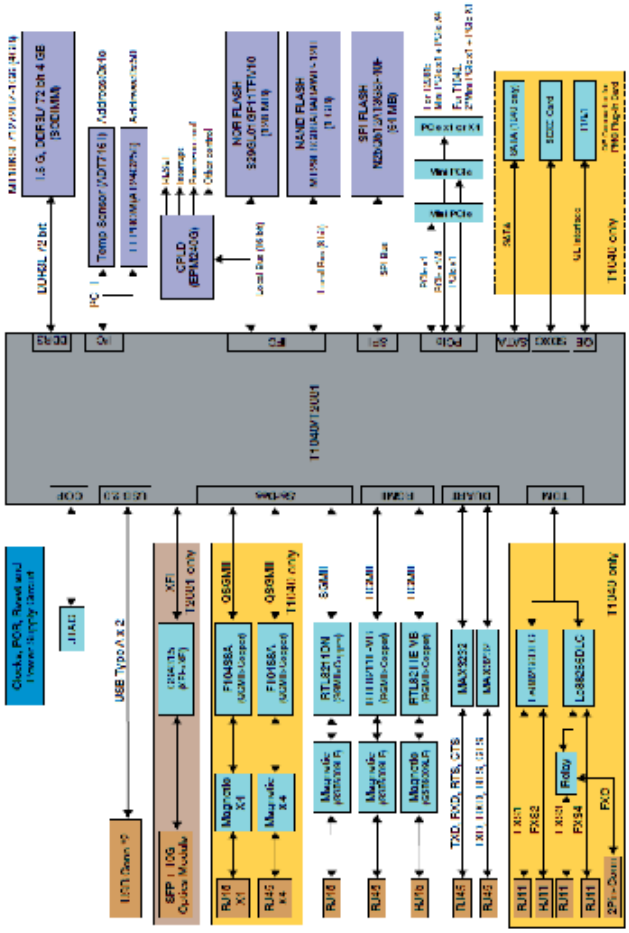

Figure 3: Block Diagram of T1040 RDB

Feature of T1040 RDB:

Processor:

- QorIQ T1040, 1.4 GHz core with 1600 MT/s DDR3L data rate
- Multiple SysClk inputs for generating various device frequencies.

Memory:

- 2 GB un buffered DDR3L SDRAM UDIMM/ RDIMM (64-bit bus), 1600 MHz data rate.
- 128 MB NOR flash, 16-bit .
- 2 GB SLC NAND flash.
- SD connector to interface.
- SATA interface.

3.1 User space data path acceleration architecture

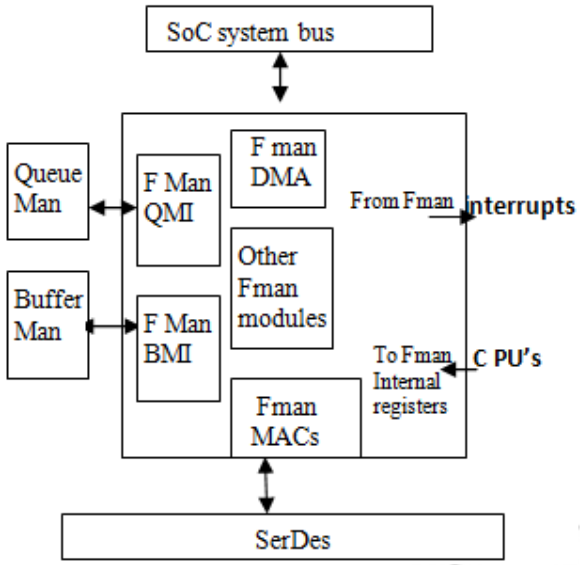

Figure 4: Fman Interfaces Block Diagram

Queuing manager:

Receives the incoming transmitted frames within the virtual link and sends and receives to and from the frame manager through the queuing manager interface.

Buffer manager:

Routes the processed packets to the L2 switch and also receives the packets from the L2 switch to the inter switch linking of the ports.

Frame manager:

Frame manager processes the incoming frames from the queuing manger process them and releases to the BMI next to the L2 switch.

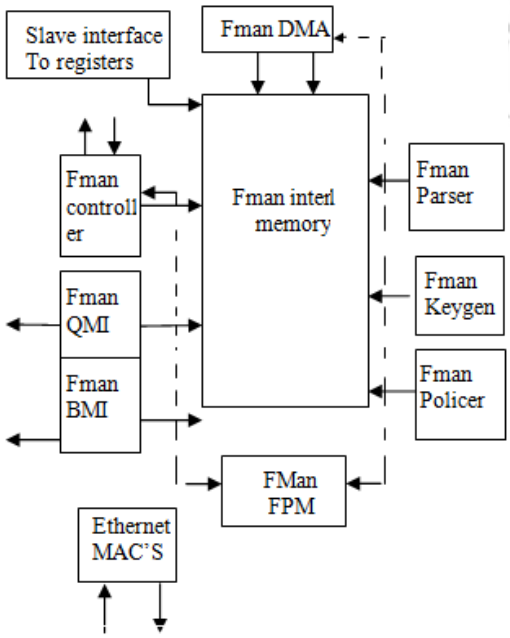

Figure 5: Fman Block diagram

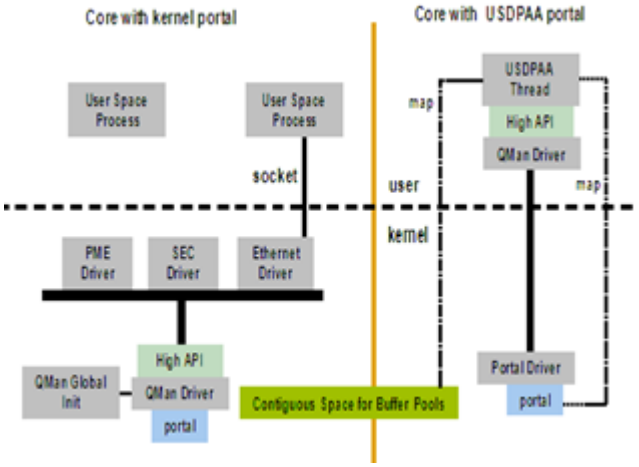

Figure 6: USDPAA Driver Architecture

3.2 Pipelining

Functional of Rx pipelining:

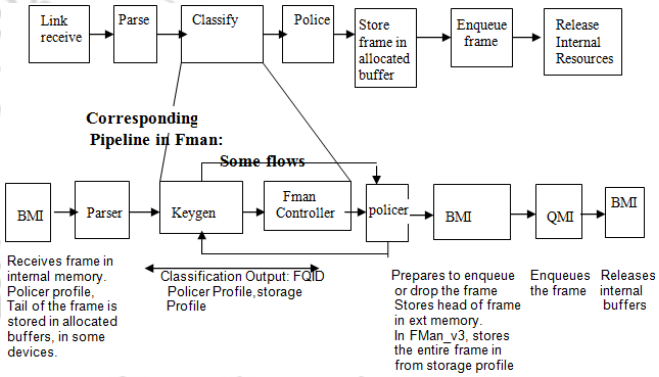

Figure 8: Pipelining architecture

Parser:

The parser reads and initiates the incoming packets entry by entry.

Policer:

Allows the only required frames to be processed by the frame manager.

4. Design of the switching software for the AFDX Switching software (Manual method by porting the u-boot image, Linux kernel, File system images (Device Tree Block) on to T1040 Reference design board)

4.1 Procedure

The whole setup consists of a Linux host machine
Which is connected to the T1040RDB and three different Pc's are connected to as the output port to the switch which acts as the end systems. Another host Pc consisting the windows operating systems which is also connected to the whole setup through the data center where configuration file is loaded through the pearl script to the end systems port and TEP port for loading configuration file on to the switch side and

different end systems are connected to the single port data center where the data can be exchanged and be communicated between any same as in the AFDX environment. It's 64-bitoperation

The T1040 processor has 4 e5500 power pc cores where in our set we use only 2 cores where one of the core will be dedicated for the linux operation where they will dedicated for the memory allocations for the Fman memory and interrupt related functions and the second core will be dedicated for the protocol functionalities like the SNMP, TFTP(The Trivial File Transfer Protocol (TFTP) is used to serve the boot image to the client. Theoretically, any server, on any platform, which implements these protocols, may be used . The **JFFS2** is the log-structured file system for use with flash memory devices. The project is deployed by using the freescale standard development kit(SDK) v1.7 developed on the Yocto project distribution for the root file system deployment. (The **project is undertaken at Research center imarat A.P.J abdul kalam Missile complex, Hyderabad for the RCI design AFDX switch the test setup is done on T1040RDB.)**

4.2 Host setup

Yocto requires some packages to be installed on host. The following steps are used for preparing the environment for Yocto running.
1. $ cd <yocto_install_path>
2. $. /scripts/host-prepare.sh

4.2.1 Builds

To set up a cross compile environment and perform builds
.$cd<yocto_install_path>/build_<machine>release
2. $ bitbake <image-target>
Where <image-target> is one of the following:

- fsl-image-minimal: contains basic packages to boot up a board
- fsl-image-core: contains common open source packages and Freescale-specific
- packages.
- fsl-image-full: contains all packages in the full package list.
- fsl-image-flash: contains all the user space apps needed to deploy the fsl-imagefull
- Image to a USB stick, hard drive, or other large physical media.
- fsl-image-kvm: contains guest rootfs in qemu
- fsl-toolchain: the cross compiler binary package
- package-name(usdpaa): build a specific package

4.2.2 Configure and rebuild the U-Boot

1. Modify U-Boot source code
   a. $ bitbake -c cleansstate u-boot
   b. $ bitbake -c patch u-boot
   c. $ cd <S> and modify the source code
2. Modify U-Boot configuration.
   a. $ modify UBOOT_MACHINES
      e.g. UBOOT_MACHINES = "T1040RDB"
3. Rebuild U-Boot image
   a. $ cd build_<machine>_release
   b. $ bitbake -c compile -f u-boot
   c. $ bitbake u-boot

4.2.3 Configuring Linux Kernel

1. Modify kernel source code
   a. $ bitbake -c cleansstate virtual/kernel
   b. $ bitbake -c patch virtual/kernel
   c. $ cd <S> and change the source code

Use bitbake -e <package-name> | grep ^S= get the value of<S>.

2. Change the kernel defconfig
   a. $ update KERNEL_DEFCONFIG variable in meta-fsl-ppc/conf/machine/<machine>.conf
   machine =T1040RDB
3. Change dts
   a. $ update KERNEL_DEVICETREE variable in meta-fsl-ppc/conf/machine/<machine>.conf
4. Do menuconfig
   a. $ bitbake -c menuconfig virtual/kernel
5. Rebuild Kernel image
   a. $ cd build_<machine>_release
   b. $ bitbake -c compile -f virtual/kernel
   c. $ bitbake virtual/kernel

4.2.3 Customize a Root Filesystem

Packages included in a rootfs can be customized by editing the corresponding recipe:
fsl-image-flash:meta-fsl-networking/images/fsl-image-flash.bb
fsl-image-core:meta-fsl-networking/images/fsl-image-core.bb
fsl-image-full:meta-fsl-networking/images/fsl-image-full.bb
fsl-image-kvm:
meta-fsl-networking/images/fsl-image-kvm.bb
fsl-image-minimal:meta-fsl-networking/images/fsl-image-minimal.bb
fsl-toolchain:meta-fsl-networking/images/fsl-toolchain.bb
The rootfs type can be customized by setting the IMAGE_FSTYPES variable in the
above recipes.
Supported rootfs types include the following:
cpio
cpio.gz cpio.xz
cpio.lzma
cramfs
ext2
ext2.gz
ext2.gz.u-boot
ext2.bz2.u-boot
ext3
ext3.gz.u-boot
ext2.lzma
jffs2
live
squashfs
squashfs-lzma
ubi
tar
tar.gz
tar.bz2
tar.xz

Specify the preferred version of package:

<PREFERRED_VERSION_pkgname> is used to configure the required version of a package.
If <PREFERRED_VERSION> is not defined, Yocto will pick up the recent version.
to downgrade Samba from 3.4.0 to 3.1.0: add PREFERRED_VERSION_samba = "3.1.0" in meta-fslppc/conf/machine/<machine>.conf
machine: T1040RDB

Rebuild rootfs:
$ bitbake <image-target>

4.2.4 Extract Source Code
To extract the source code of a package, do the following:

1)$ bitbake -c cleansstate<T1040RDB-fsl_networing-linux/u-boot>
2)$ bitbake -c patch < T1040RDBRDB-fsl_networing-linux/u-boot >
3)$ cd <S>Use bitbake -e <package-name> | grep ^S= to get the value of <S>.
4)$ bitbake -c cleansstate u-boot For example, to do a U-boot of a T2080RDB processor.
5)$ bitbake -c patch u-boot.

4.2.5 Standalone toolchain
Build and install the standalone toolchain with Yocto:
1. $ source/fsl-setup-poky -m <T1040RDB>
2. $ bitbake fsl-toolchain
3. $ cd build_<T1040RDB>_release/tmp/deploy/sdk
4.$./fsl-networking-eglibc-<host-system>-<e5500>-toolchain-<release>.sh

The default installation path for standalone toolchain is /opt/fsl-networking/. The install folder can be specified during the installation procedure. To use the installed toolchain, go the the location where the toolchain is installed and source the environment-setup-<core> file. This will set up the correct path to the build toolsand also export some environment variables relevant for development (eg. $CC, $ARCH, $CROSS_COMPILE, $LDFLAGS etc).

To invoke the compiler, use the $CC variable (eg. $CC <source files>).

This is a sysrooted toolchain. This means that GCC will start to look for target fragments and libraries (eg. crt*, libgcc.a) starting from the path specified by the sysroot. The default sysroot is preconfigured at build time to point to /opt/fsl-networking/QorIQ-SDK-<sdk_version>/sysroots/<target_architecture>.If the toolchain is installed in a location other than the defaultone (/opt/fsl-networking/), the--sysroot=<path_to_target_sysroot>
parameter needs to be passed to GCC. When invoking thecompiler through the $CC variable, there is no need to pass the --sysroot parameter as it is already included in the variable (check by running echo $CC).

4.3 The Proof of code (POC) or Reflector program developed using U-boot on Host machine:

U-boot settings required for HIDA PoC
==========
Execute below commands at u-boot prompt.

#below commands disables two CPUs to emulate T1020
cpu 2 disable
cpu 3 disable

#below command set boot arguments to disable qportal and bportal interrupts and isolates cpu 1 from linux scheduling
setenv bootargs root=/dev/ram rw ramdisk_size=3000000000 console=ttyS0, 115200 usdpaa_mem=256M qportals=s0 bportals=s0 isolcpus=1

save U-boot settings
Save env

steps to run HIDA switch PoC application
==========

Copy all files from app folder to T1040RDB file system <folder>.

Run below commands at Linux shell prompt to start HIDA switch PoC application.
#tftpboot 0x100000 u-Image.bin
#tftpboot 0x400000 Kernelimage 12.3
#tftpboot 0x200000 fsl-image.dtb

#bootm 0x1000000 0x4000000 0x2000000**(Store on to the NOR flash)**
root: xyz
password: xyz

ifconfig -a

#ifconfig fm1-gb2 172.22.22.10 netmask 255.255.255.0 up**(Setting up the network to copy the configuration file to the target board, fm1-gb2 –Ethernet port on the target board)**

#mkdir /usr/etc
#cd /usr/etc
#cp hidaconfig.txt /usr/etc

#scproot@172.22.22.4:/srv/tftpboot/hidaconfig.txt

frame manager configuration.
#fmc -c usdpaa_config_t1_serdes_0x66.xml -p usdpaa_policy_hash_ipv4.xml –a

(kill already running switch daemon and delete all its files.)
#killall l2sw_bin

#rm -rf /tmp/il2sw /tmp/ol2sw

#chmod +x hidaswitchpoc.binbhida_switch_app.bin

#/hida_switch_app.bin –n 1 –c usdpaa_config_t1_serdes-0x66.xml –p usdpaa_policy_hash_ipv4.xml**.(Run HIDA Switch PoC**

application. Name(ipfwd_app) will be changed to hida_switch_app further).

5. **AFDX Protocols test cases and Results**

1. Policing test:

The Configuration file is loaded into the both end system and the switch when the BAG of the end system is greater than the BAG value of the switch then test fails if the BAG of the end system is less than the switch BAG then the test passes. This test is considered for 10 virtual links port speed of 1000Mbps.

Input to Switch End system:
(hidaconfig.txt)-Configuration file
Input to Switch End system:
(hidaconfig.txt)-Configuration file

#PORTCONFIG, "(portno, maxdelay(in), portstate(1-up, 0-down), portspeed(100-100Mbps, 1000-1Gbps, 10-10Gbps), opbufsizelowprio, opbufsizehiprio)", , , , , , , ,
1, 100, 1, 1000, 128, 256, , , , , , , , , , ,
2, 100, 1, 1000, 128, 256, , , , , , , , , , ,
3, 100, 1, 1000, 128, 256, , , , , , , , , , ,
4, 100, 1, 1000, 128, 256, , , , , , , , , , ,
5, 100, 1, 1000, 128, 256, , , , , , , , , , ,
6, 100, 1, 1000, 128, 256, , , , , , , , , , ,
7, 100, 1, 1000, 128, 256, , , , , , , , , , ,
8, 100, 1, 1000, 128, 256, , , , , , , , , , ,
9, 100, 1, 1000, 128, 256, , , , , , , , , , ,
10, 100, 1, 1000, 128, 256, , , , , , , , , , ,
#VLCONFIG, "(vlid, Smin, Smax, bag(0.5-128ms), jitter(0-10000us), inputport, outputport(op)1, op2, op3, op4, op5, op6, op7, op8, op9, op10, ACi, priority 0 - low 1 - high)", , , , , , , , , , , , , , , , , ,
0, 100, 200, 33, 1000, 3, 2, 1, , , , , , , , , 0, 0,
2, 100, 200, 32, 1000, 3, 2, 1, , , , , , , , , 0, 0,
65533, 100, 200, 32, 1000, 1, 3, 2, , , , , , , , , 0, 0,
65534, 100, 200, 50, 1000, 1, 3, 2, , , , , , , , , 0, 0,
350, 100, 200, 32, 1000, 2, 3, 1, , , , , , , , , 0, 0,
500, 100, 200, 49, 1000, 2, 3, 1, , , , , , , , , 0, 0,
, ##END, , , , , , , , , , , , , , , , , ,

End system configuration

#VLCONFIG, "(vlid, Smin, Smax, bag(0.5 - 128ms), jitter(0 - 10000us), inputport, outputport(op)1, op2, op3, op4, op5, op6, op7, op8, op9, op10, ACi, priority 0 - low 1 - high)", , , , , , , , , , , , , , , , , , , ,
0, 100, 200, 32, 1000, 3, 2, 1, , , , , , , , , 0, 0,
2, 100, 200, 28, 1000, 3, 2, 1, , , , , , , , , 0, 0,
65533, 100, 200, 40, 1000, 1, 3, 2, , , , , , , , , 0, 0,
65534, 100, 200, 49, 1000, 1, 3, 2, , , , , , , , , 0, 0,
350, 100, 200, 36, 1000, 2, 3, 1, , , , , , , , , 0, 0,
500, 100, 200, 32, 1000, 2, 3, 1, , , , , , , , , 0, 0,

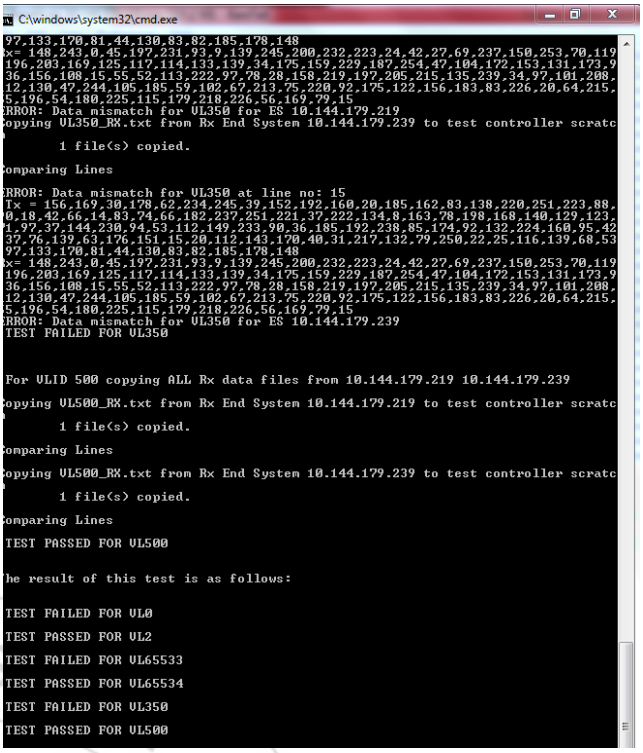

Figure 8: Test Result for the Policing test

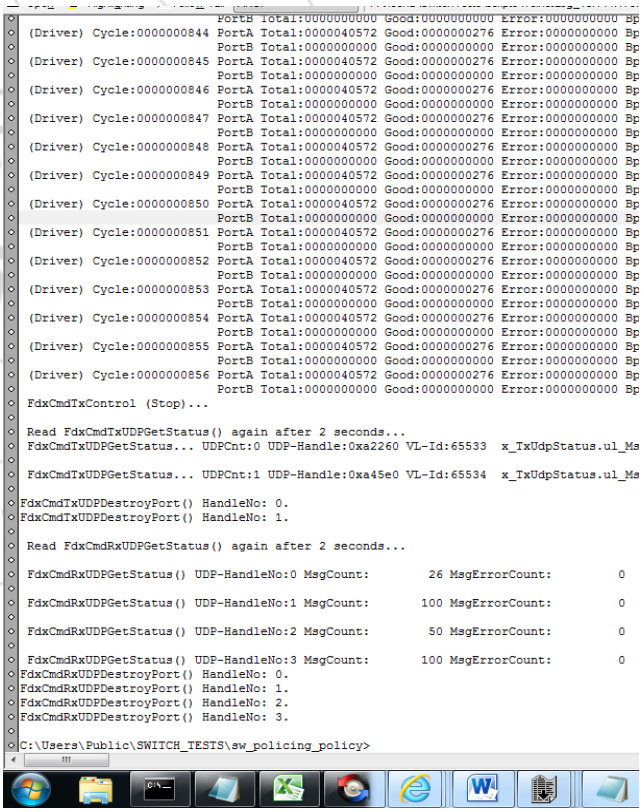

Figure 9: Packet flow status on the Bare tail tool

5. Filtering Test

a. Smin Filterning Test:

All the frames with size below 84 bytes or Smin value as programmed in configuration table to be discarded and MIB to be updated port wise

Configurtion table(hidaconfig.txt)

bufsize low prio, opbufsize hi prio)", , , , , , , , , , , , , , , , ,

1, 100, 1, 1000, 128, 256, , , , , , , , , , ,
2, 100, 1, 1000, 128, 256, , , , , , , , , , ,
3, 100, 1, 1000, 128, 256, , , , , , , , , , ,
4, 100, 1, 1000, 128, 256, , , , , , , , , , ,
5, 100, 1, 1000, 128, 256, , , , , , , , , , ,
6, 100, 1, 1000, 128, 256, , , , , , , , , , ,
7, 100, 1, 1000, 128, 256, , , , , , , , , , ,
8, 100, 1, 1000, 128, 256, , , , , , , , , , ,
9, 100, 1, 1000, 128, 256, , , , , , , , , , ,
10, 100, 1, 1000, 128, 256, , , , , , , , , , ,
#VLCONFIG, "(vlid, Smin, Smax, bag(0.5 - 128ms), jitter(0 - 10000us), inputport, outputport(op)1, op2, op3, op4, op5, op6, op7, op8, op9, op10, ACi, priority 0 - low 1 - high)", , , , , , , , , , , , , , , ,
0, 100, 200, 32, 1000, 3, 2, 1, , , , , , , , , 0, 0,
2, 100, 200, 32, 1000, 3, 2, 1, , , , , , , , , 0, 0,
4, 100, 200, 32, 1000, 3, 2, 1, , , , , , , , , 0, 0,
65533, 100, 200, 32, 1000, 1, 3, 2, , , , , , , , , 0, 0,
65534, 100, 200, 32, 1000, 1, 3, 2, , , , , , , , , 0, 0,
65535, 100, 200, 32, 1000, 1, 3, 2, , , , , , , , , 0, 0,
350, 100, 200, 32, 1000, 2, 3, 1, , , , , , , , , 0, 0,
500, 100, 200, 32, 1000, 2, 3, 1, , , , , , , , , 0, 0,
1350, 100, 200, 32, 1000, 2, 3, 1, , , , , , , , , 0, 0,
##END, , , , , , , , , , , , , , , , , ,

```
inside option Pinside option I 1 : 1inside option SGLOBAL STATSPORT STATS of PORT : 1
lmin_err : 300  filtercnt : 300 dropped : 300   rx : 300VL STATS of VL : 0stats command done successfullyinside
option Pinside option I 2 : 2inside option SGLOBAL STATSPORT STATS of PORT : 2  lmin_err : 100
filtercnt : 100 dropped : 100   tx : 200 rx : 300VL STATS of VL : 0stats command done successfullyinside
option Pinside option I 3 : 3inside option
SGLOBAL STATSPORT STATS of PORT : 3     lmin_err : 100  filtercnt : 100 dropped : 100   tx : 200
rx : 300VL STATS of VL : 0stats command done successfully
```

Figure log file of smin and port status

Smax Filtering Test:

All the frames with size above 1538 bytes or Smax value as programmed in configuration table to be discarded and MIB to be updated port wise

```
inside option Pinside option I 1 : 1inside option SGLOBAL STATSPORT STATS of PORT : 1
lmin_err : 300  filtercnt : 300 dropped : 300   rx : 300VL STATS of VL:0stats command done successfullyinside option
Pinside option I 2 : 2inside option SGLOBAL STATSPORT STATS of PORT: 2lmin_err : 100filtercnt : 100dropped : 100
tx : 200 rx : 300VL STATS of VL : 0stats command done successfullyinside option Pinside option
 I 3 : 3inside option SGLOBAL STATSPORT STATS of PORT : 3
lmin_err : 100  filtercnt : 100 dropped : 100   tx : 200rx : 300VL STATS of
VL : 0stats command done successfully
```

Figure 10: Log file S max and the respective port status

6. Conclusion

After implementing the testing the Filtering and policing logics on the T1040RDB reference design board for a speed for a speed of 1000Mbps the testing will be implemented on the RCI designed AFDX switch.

7. Acknowledgement

I would like thank Scientist C. Ramesh Reddy and Chandra Bhusan Kumar at Research center imarat A.P.J Abdul Kalam Missile Complex, Hyderabad, for their support and helpful comments.

References

[1] A Software Implementation of AFDX End System Xin Chen, Xudong Xiang, Jianxiong Wan and Technology, 2007. Xin Chen, Xudong Xiang, Jianxiong Wan.
[2] AFDX Software Network Stack Implementation – Practical Lessons Learned, *Imad Khazali, Marc-André Boulais, and Phil Cole, Embvue Inc., Montreal, Quebec, Canada.*
[3] The Design and Implementation of the AFDX Network Simulation System.
[4] Freescale Linux SDK T2080 v1.1, T1040v0.3