PROJECT-REPORT

IMPLEMENTATION OF SIMPLIFIED TCP ON INTEL DE2I-150 Board

For the course work

NETWORKED SYSTEM DESIGN (EE - 550)

By

Avinash Palamanda (50166647)
Aravind Srinivas Srinivasa Prabhakar (50166473)
Darshan Godabanahal Malleshappa (50170064)
Sandeep Shanmugarajan (50166471)

Introduction:

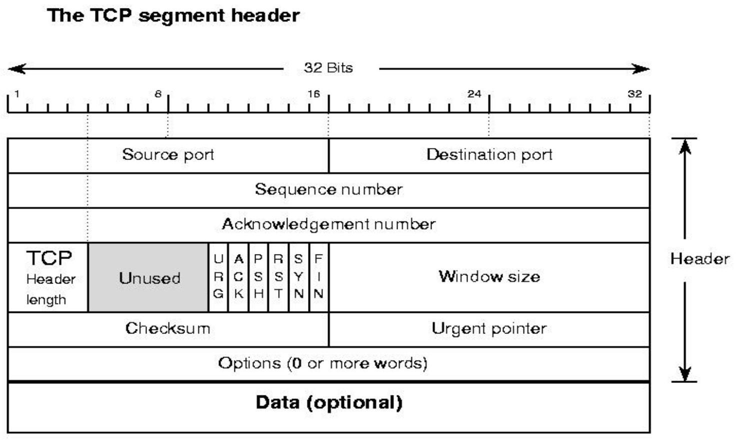
The TCP/IP protocol is designed such that each computer or device in a network has a unique "IP Address" (Internet Protocol Address) and each IP address can open and communicate over up to 65535 different “ports” for sending and receiving data to or from any other network device. The IP Address uniquely identifies the computer or device on the network and a "Port Number" identifies a specific connection between one computer or device and another (i.e. between two IP Addresses).

The Transmission Control Protocol (TCP) unlike User Datagram Protocol (UDP) is a reliable or a Connection-Oriented protocol, which ensures all the bytes of the data that is being sent from the sender is received correctly and in the same order in which it was sent. It provides a host-to-host connectivity at the transport layer of the Internet model. The connection between two hosts, i.e a sender and a receiver is established using a Three-way Handshake process. In order to guarantee the reliability of data transfers, TCP uses a technique called ‘Positive Acknowledgement with Re-transmissions’. This technique involves the Receiver to acknowledge the data which it has received. Every time the Sender sends data, it keeps its record and uses a timer at the exact moment the data is sent to the host at the receiving end. So if the data has not acknowledged at the right time by the receiver, i.e. if the timer expires by the time the data has been acknowledged, the Sender re-transmits the data.

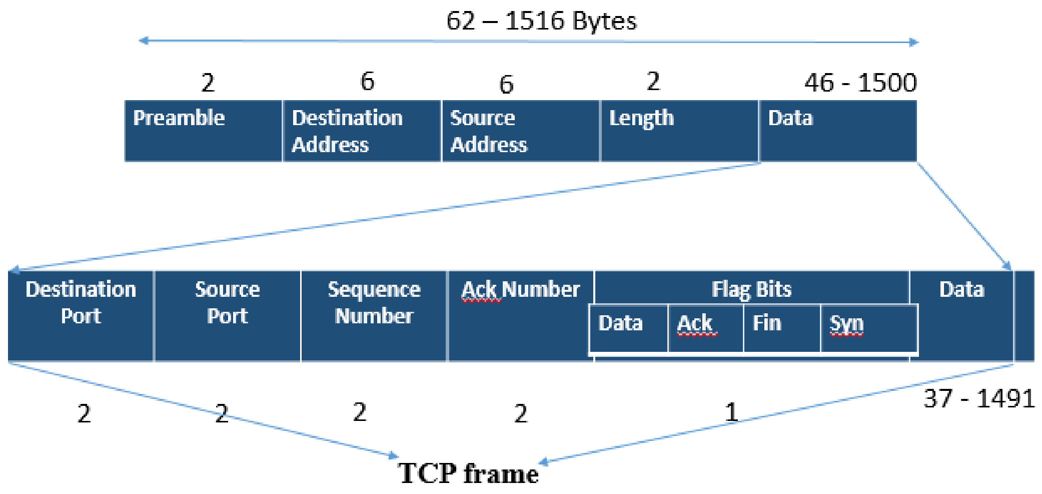
TCP although known for its high reliability and accuracy in the delivery of data, it has its own limitations. The TCP incurs long delays usually of the order of seconds while it waits for re-transmissions of the messages that has been lost. Therefore it is not advisable to use TCP when it comes to real time applications such as Media Streaming, Voice-over IP, etc. In such cases we can use protocols such as Real-time Transport Protocol (RTP) operating over User Datagram Protocol. TCP finds it applications in Email, Secure Shell, Peer-to-Peer file sharing, etc.

In our project we have implemented a simplified version of this TCP considering only the port numbers, sequence numbers and data. The flags used are SYN, FIN, ACK and DATA which are set to 1 upon the transmission /reception of the respective frames received over the Ethernet port communication between the two De2i-150 boards. Though TCP uses flow control, due to certain clocking constraints we faced, we decided to implement the stop and wait protocol along with the usual 3 way hand shake, using the altera timer module present. We did not implement the IP layer as we directly inserted the TCP header in the Ethernet payload of the link layer. We have used 2 boards connected to each other using a cross over cable. Both the boards can either be transmitter or receiver. A simple counter application has been developed to demonstrate the part of data transmission and these numbers being counted are indicated on the frame offset as well as by blinking the Red LEDs.

Frame Format of TCP:



Simplified Implementation of TCP:



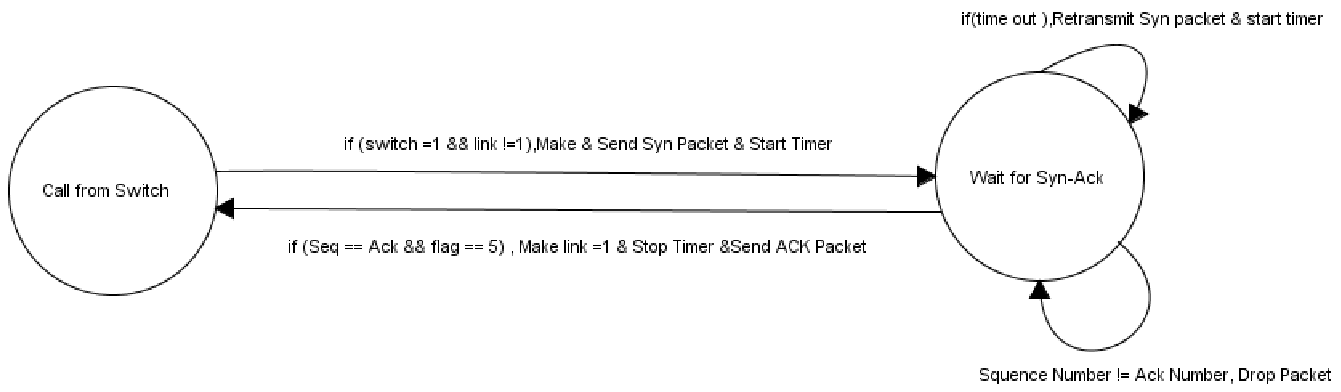
- **Source Port** : Identifies the sending port (2 Bytes)
- **Destination Port** : Identifies the receiving port (2 Bytes)
- **Sequence Number**: The Sequence Number is of 2 Bytes. Sequence number is incremented every time a new Ethernet frame is transmitted.
- **Acknowledgement Number:** The Acknowledgement Number is of 2 Bytes, then ACK number value is the next sequence number it is expecting from the other side.
- **Flags** :
    - ACK (1 Bit) - Acknowledgement Bit, when set, indicates segment carries ACK and its value is valid.
    - SYN (1 Bit) - Synchronize Bit, request to synchronize seq. num and Connection establishment.
    - FIN (1 Bit) - Finish Bit, sender requests that connection be closed.
    - DATA (1 Bit) – If the sender transmits a data this bit is set to 1
- **Data:** The Bytes of data that are to be sent.

The above figure is the representation of a simplified version of a TCP frame which we have used for our project. The size of the entire TCP frame is 48 Bytes in this project. In our case, we have made use of four flag bits, i.e. Data, Ack, Fin and Syn. Specific Flag Bits are set according to different sets of values of the flag. We have also not implemented the TCP checksum and urgent pointer to push the data for immediate requirement.

3 Way Handshake:

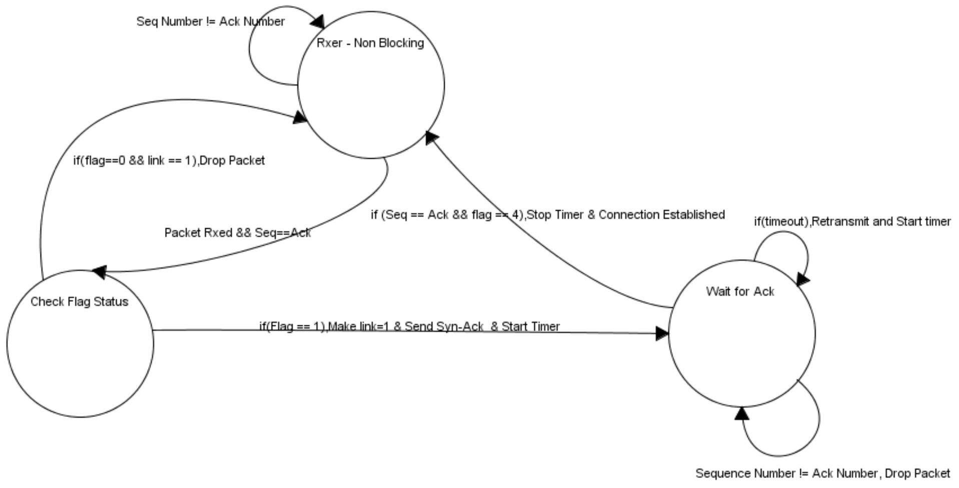
In this process of connection establishment, the TCP in general uses the size of the data transmitted as the sequence number. We have chosen to increment the sequence number linearly upon every data being transmitted post the successful establishment of the connection. Initially a SYN packet is sent and waits for a SYN-ACK post which an ACK is sent to complete the connection.

Transmitter End:


if(time out),Retransmit Syn packet & start timer
if (switch =1 && link !=1),Make & Send Syn Packet & Start Timer
Call from Switch
Wait for Syn-Ack
if (Seq == Ack && flag == 5) , Make link =1 & Stop Timer &Send ACK Packet
Squence Number != Ack Number, Drop Packet

SWITCH 1 is assigned for this process. When the Switch is ON a SYN packet is transmitted and the timer is turned on with timeout values set in the PERIODH and PERIODL registers. If the timeout occurs the frame which was sent would be retransmitted again. It also checks the link status which is declared in the TCP structure. If the link is not equal to 1, it validates the condition and transmits the SYN packet.
At the same time in the receiver, the system waits for a SYN and upon reception, it transmits the SYN-ACK along with starting the timer to handle the retransmission scenario. The transmitter then validates the condition of the sequence number being equal to the ack number being received and checks for the SYN-ACK flag being equal to 5 (0101) and makes the link equal to 1 to stop the timer and transmit an ACK packet to complete the 3 way handshake. At the receiver end if the sequence number is not equal to the acknowledgment number, the frame is dropped.

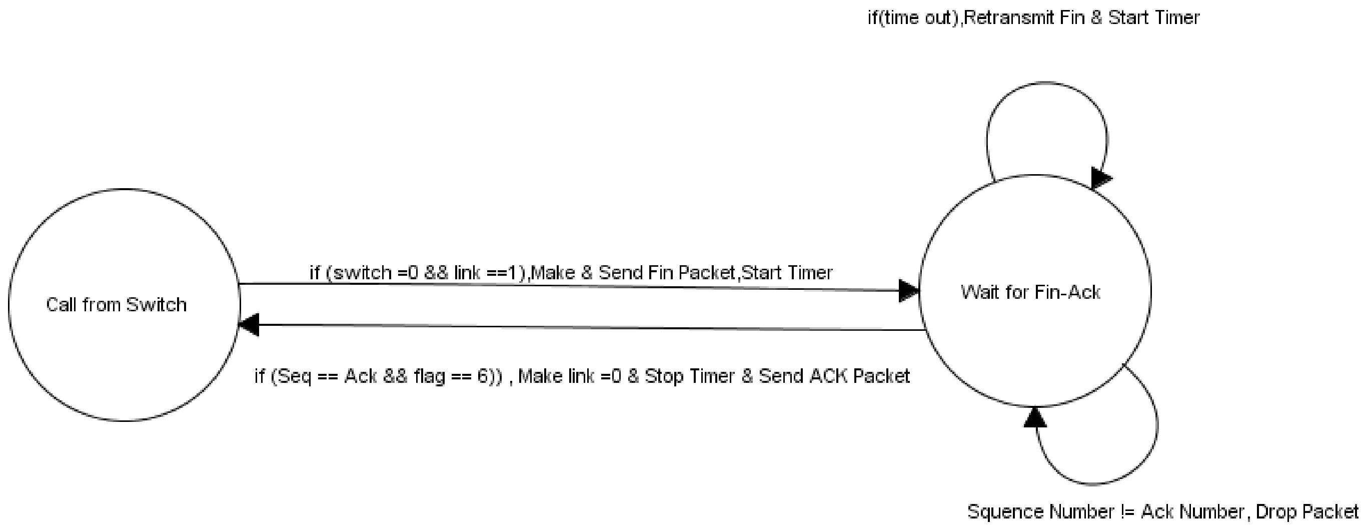
Receiver End:


Seq Number != Ack Number
Rxer - Non Blocking
if(flag==0 && link == 1),Drop Packet
if (Seq == Ack && flag == 4),Stop Timer & Connection Established
if(timeout),Retransmit and Start timer
Packet Rxed && Seq==Ack
Check Flag Status
Wait for Ack
if(Flag == 1),Make link=1 & Send Syn-Ack & Start Timer
Sequence Number != Ack Number, Drop Packet

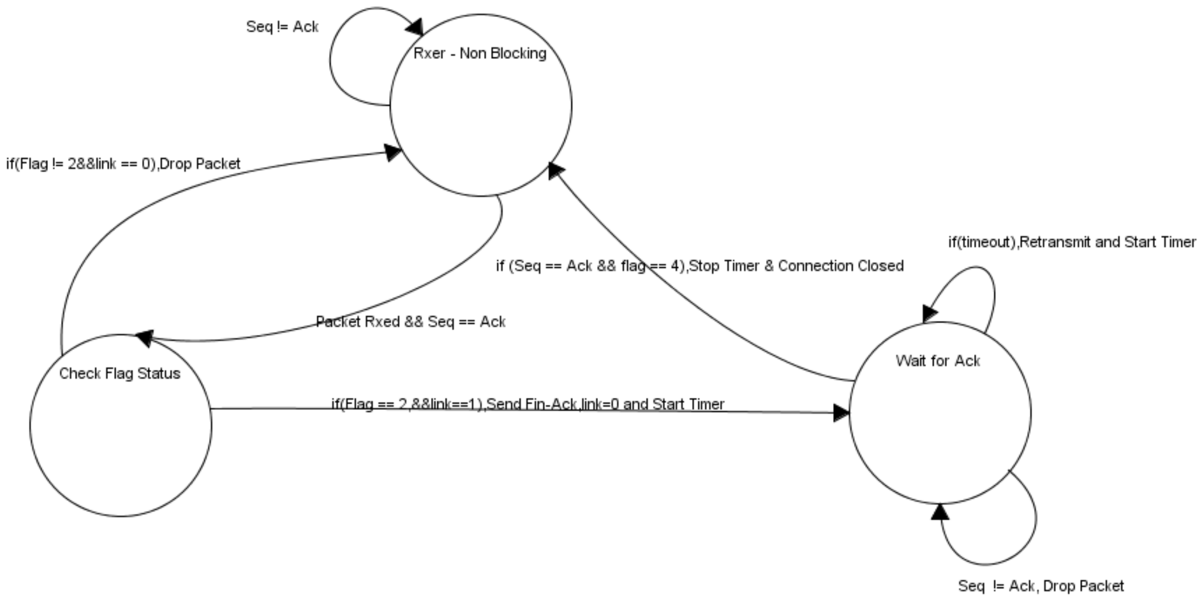
Here similar conditions are checked as in transmitter where link status are changed to handle frame accept and drop. If the acknowledgement number 4 (0100) received the connection is established and a print occurs stating the same indicating the user to start the data transmission. If the flag status == 1 , send SYN ACK and make the TCP link == 1 . If flag == 1 and link status == 0 we drop the packet allowing the other end to timeout and retransmit. Thus complete 3 way handshake procedure is carried out similar to the original TCP.

Similar procedure is carried out for the connection tear down process. The below Finite State Machine describes the procedure.

Transmitter side


if(time out),Retransmit Fin & Start Timer
Call from Switch
if (switch =0 && link ==1),Make & Send Fin Packet,Start Timer
Wait for Fin-Ack
if (Seq == Ack && flag == 6)) , Make link =0 & Stop Timer & Send ACK Packet
Squence Number != Ack Number, Drop Packet

Receiver side:


Seq != Ack
Rxer - Non Blocking
if(Flag != 2&&link == 0),Drop Packet
if(timeout),Retransmit and Start Timer
if (Seq == Ack && flag == 4),Stop Timer & Connection Closed
Packet Rxed && Seq == Ack
Wait for Ack
Check Flag Status
if(Flag == 2,&&link==1),Send Fin-Ack,link=0 and Start Timer
Seq != Ack, Drop Packet

Flag Value	**Flag Bits**				**Explanation**
	Data	*Ack*	*Fin*	*Syn*	
0x01	0	0	0	1	Syn Bit is Set
0x02	0	0	1	0	Fin Bit is Set
0x05	0	1	0	1	Syn-Ack bit is Set
0x06	0	1	1	0	Fin-Ack bit is Set
0x08	1	0	0	0	Data Bit is Set
0x0C	1	1	0	0	Data-Ack bit is Set

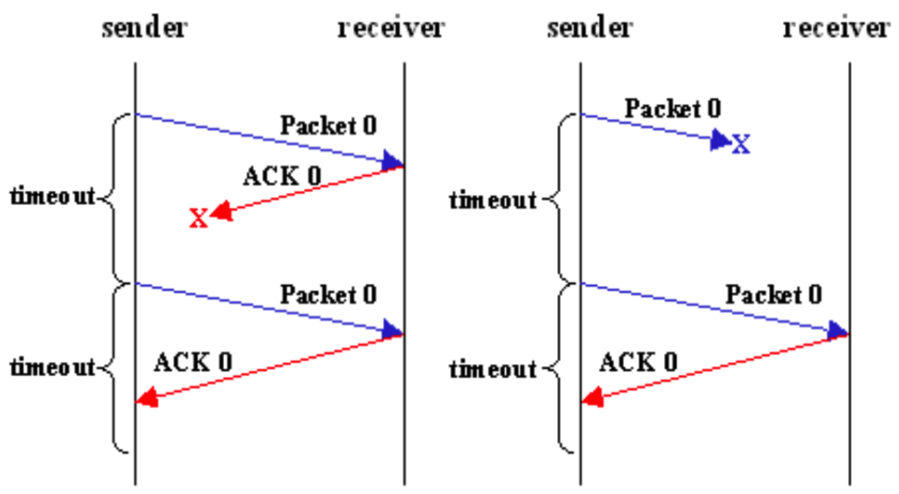
Fig: Table describing the flag bits used in simplified TCP implementation

Stop and Wait Protocol:

Since we haven't implemented the flow control, to make sure the data transmitted is reliable and successfully transmitted we have used stop and wait protocol. It makes use of two mechanisms which include:

- Acknowledgments
- Timeouts

The working of Stop and Wait protocol is explained using a schematic diagram below:



The timer starts at the exact moment the Sender sends the data/message to the Receiver. If the Sender does not receive the Acknowledgement for the previous sent data before timout, the Sender Retransmits the data it sent again. There are two instances the Sender does not receive acknowledgement from the Receiver, they are:

- When ACK is Lost.
- When the frame is itself lost.

During Every data transmission the sequence and acknowledgement numbers are incremented and checks for validity of the condition so that an ACK is received before the next data transmission. This ensures successful delivery of the transmitted frame and can be seen in the terminal.

Implementing the Nios II Timer Module:

There are 4 types of timer which can be implemented in the NIOS II system during the setup of Qsys. They are as follows.

- Full feature
- Watch Dog Timer
- Simple interrupt timer
- Custom Mode

The Timer module is a 32 bit timer which can be used as periodic pulse or a system watchdog timer. Upon the timeout value reaching zero, the interrupt can be either enabled or disabled. Either the 32 bit counter or the 64 bit counter can be chosen while adding the timer module in the Qsys. In our project we have chosen a full feature timer along with a 32 bit counter.

The Nios II timer runs off using a single master clock input (clk), which is the same that is provided to the Nios CPU and other peripherals. The timer register map is given below:

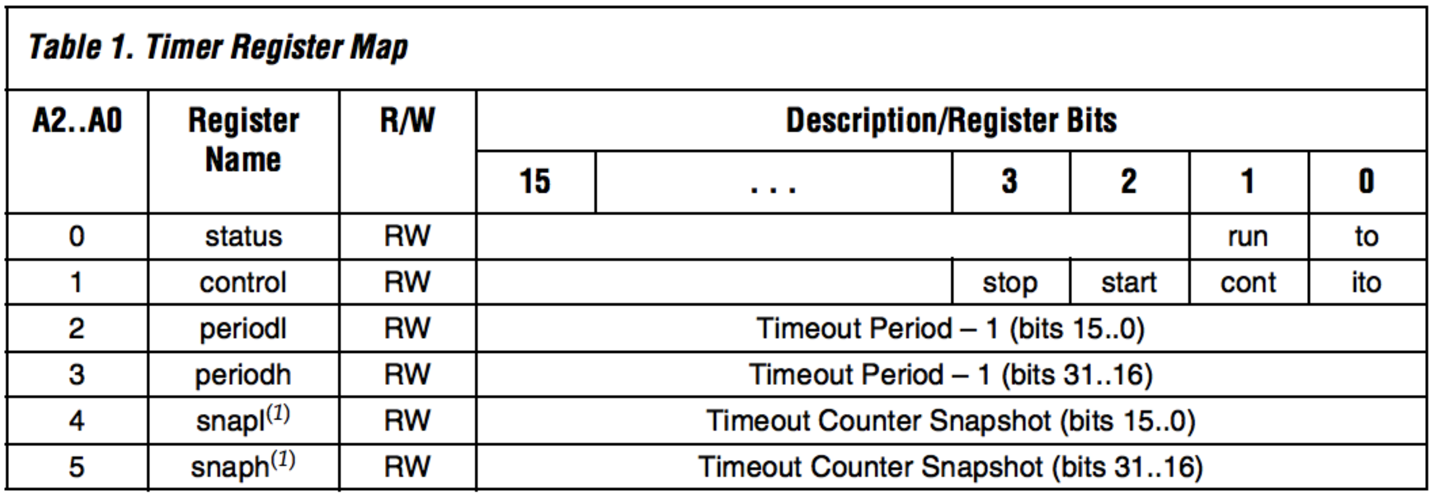


Table 1. Timer Register Map

A2..A0	Register Name	R/W	Description/Register Bits					
			15	. . .	3	2	1	0
0	status	RW					run	to
1	control	RW			stop	start	cont	ito
2	periodl	RW	Timeout Period – 1 (bits 15..0)					
3	periodh	RW	Timeout Period – 1 (bits 31..16)					
4	snapl[1]	RW	Timeout Counter Snapshot (bits 15..0)					
5	snaph[1]	RW	Timeout Counter Snapshot (bits 31..16)					

Status Registers:

- *TO bit* is set to 1 when countdown reaches to 0, it has to be explicitly cleared by writing 0 in the Status Register.
- *RUN* bit is internally set to 1 when running and 0 when not we don't have access to that bit

Control Registers:

- If *ITO* is set to 1, interrupt generates an IRQ when TO bit is 1
- *CONT* bit is set to 1 for the counter to run continuously. If needed to stop the clock , we have to set the stop bit to 1 in the Control Register.
- *START* bit is set to 1 to start the internal timer.
- *STOP* bit is set to 1 to stop the internal timer.

Writable Period and Readable Snap registers

- *PERIODH/PERIODL* is used to load timeout period value
- *SNAPH/SNAPL* gives the current counter value

Header files required:

```
#include<altera_Avalon_timer.h>
#include<altera_Avalon_timer_regs.h>
```

The following functions have been used in our program in order to READ or WRITE the timer:

```
IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_0_BASE, 0x0000);  //set TO bit to 0 if  1

IOWR_ALTERA_AVALON_TIMER_PERIODL(TIMER_0_BASE ,0xAAAA); //timeout value

IOWR_ALTERA_AVALON_TIMER_PERIODH(TIMER_0_BASE , 0x00FF);//timeout value

IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_0_BASE, 0x0007); //start timer

IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_0_BASE, 0x0008); //stop timer

IORD_ALTERA_AVALON_TIMER_STATUS(TIMER_0_BASE);  //Read Status of the timer

IORD_ALTERA_AVALON_TIMER_SNAPL(TIMER_0_BASE ); //Counter Snapshot

IORD_ALTERA_AVALON_TIMER_SNAPH(TIMER_0_BASE); //Counter Snapshot
```

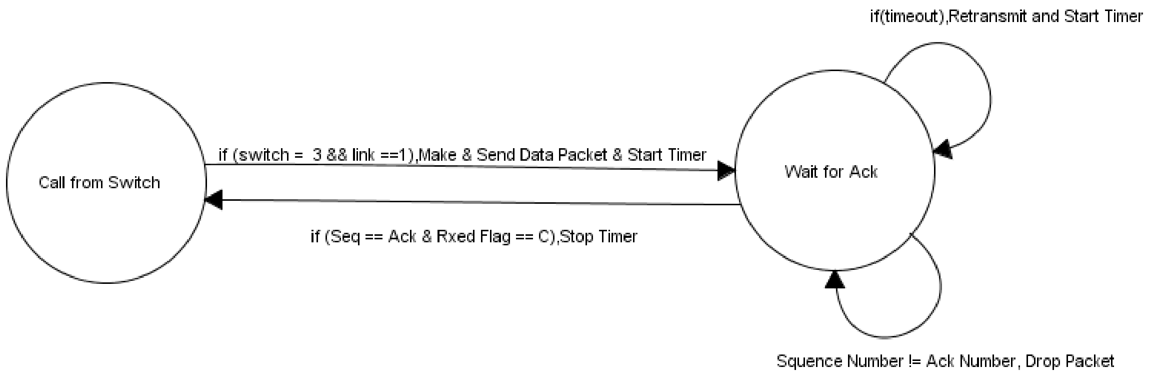
Application - Simple Counter:

We have implemented a simple counter as an application of simplified TCP. The frame that has been transmitted by the sender, has a count value that increments when it has successfully transmitted. When the receiver board receives the data, specific LEDs blinks or are turned on. In order to show the working of timer and retransmission, we have used SWITCH 3(turn ON) to set up dropping of frames. When the frames are dropped, retransmission from the sender side occurs. When the SWITCH 3 is turned OFF, retransmission of lost frames is stopped and the data being sent by the Sender has the next sequence number in the sequence number field of the TCP header. We have assigned the following functions to be executed for the three switches

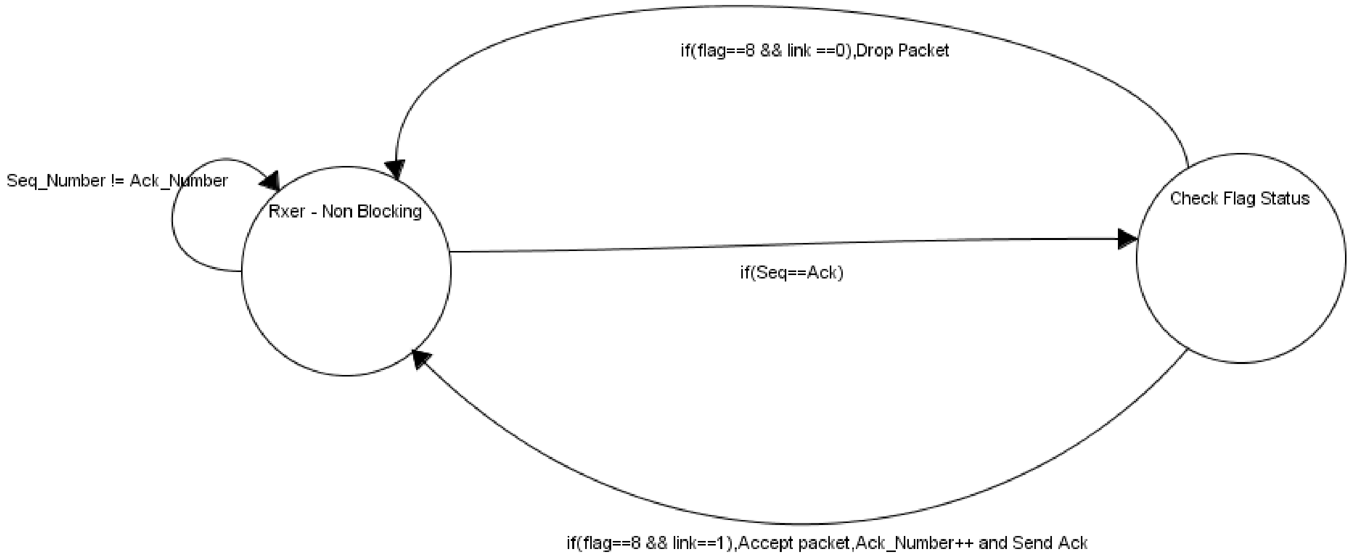
SWITCH	**FUNCTION**
Switch 1	Three way handshake
Switch 2	Data Transmission
Switch 3	Drop Receiving Frames

Further we are printing the statistics of the frames being transmitted and received when we initiate the connection close process. The TCP header statistics along with the Ethernet frame statistics are printed where we could see the number of erroneous frames, lost frames and successfully received. These help in further analyzing the working of our simplified TCP model.

Transmitter Side:


if(timeout),Retransmit and Start Timer
Call from Switch
if (switch = 3 && link ==1),Make & Send Data Packet & Start Timer
Wait for Ack
if (Seq == Ack & Rxed Flag == C),Stop Timer
Squence Number != Ack Number, Drop Packet

Receiver Side:


if(flag==8 && link ==0),Drop Packet
Seq_Number != Ack_Number
Rxer - Non Blocking
Check Flag Status
if(Seq==Ack)
if(flag==8 && link==1),Accept packet,Ack_Number++ and Send Ack

Implementation Psuedo Code:

For each,
Transmit Packet: Sequence Number = Sequence Number +1
Receive Packet: Ack Number = Ack Number + 1

If the connection has already been established from one side, then if we try to inititate a connection again from the other end, we would skip the connectivity part by printing the connection has already been established and request to directly transmit the data by turning on the swich 2, which is also one of the condition present. Everytime data has to get transmitted, the link status should be equal to 1 i.e. connection established. Once it receives a packet, the timer is stopped calling the appropriate acknowledgement frame to be sent.

<u>Conclusion:</u>

A simplified TCP was implemented on a De2i-150 Intel board using Verilog as the hardware programming language and C as the software programming language. A simple counter application by blinking the LEDs was done to demonstrate this implementation.

The following were implemented in the project:

- Host-to-Host connection establishment was executed using three-way handshake technique.
- Simple Stop and Wait Protocol.
- Implementation of Timer for retransmission of frames in case of timeouts.
- A simple counter was implemented as a application, where in LEDs glow as per the sequence of the data that has been received by the host.
- Statistics of the Ethernet frames and the TCP frames are that are sent and received were printed.

The following were not implemented:

- Flow control : due to clocking issues , we decided to implement stop and wait protocol
- Congestion control
- IP Layer: we have transmitted the TCP header directly into the payload of the Ethernet frame.
- Urgent pointer and push flag along with TCP checksum

<u>References:</u>

- Computer Networking a Top-Down Approach 6th edition by James F. Kurose University of Massachusetts, Amherst Keith W. Ross Polytechnic Institute of NYU.

- Altera Embedded IP core manual.

- Nios II reference guide

- http://www.isi.edu/nsnam/DIRECTED_RESEARCH/DR_HYUNAH/D-Research/stop-n-wait.html

- http://www.taltech.com/datacollection/articles/a_brief_overview_of_tcp_ip_communications