Protocol Plugin Development

Pseudo UDP Over XAUI

This lesson introduces protocol plugin development interfaces. The example presented implements a pseudo UDP socket based protocol using the XAUI 10GE plug-in.

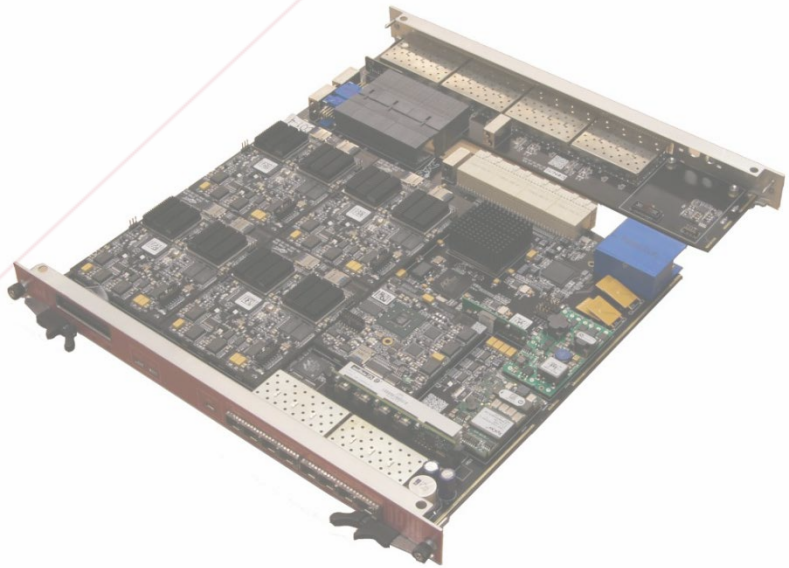
Objectives:

- Compile and link the pseudo UDP shareable library and transmit/receive executables.

- Explore the Socket Abstraction Services (SAS) programming interface using example code.

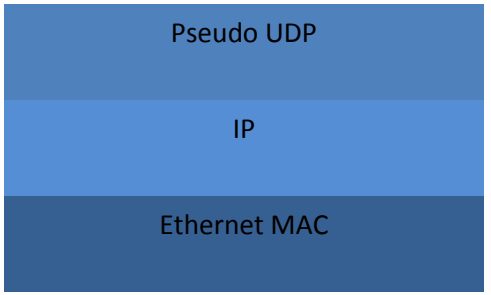
- Utilize the embedded console to run the examples on two separate RCEs

*Note that the focus of the exercise is **not** on how to implement a real UDP protocol.*

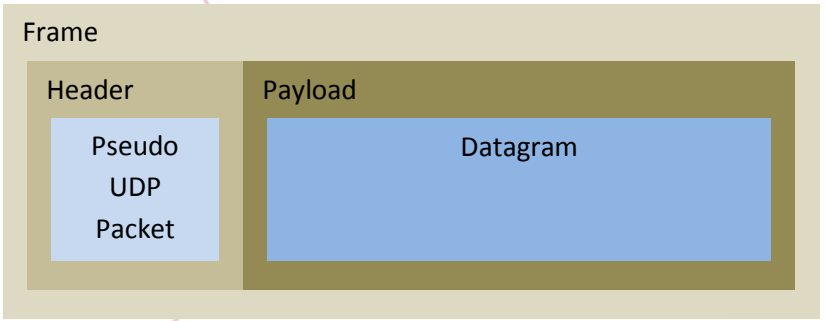


Introduction

A simple command and response is demonstrated by exchanging Pseudo UDP packets. The UDP packet consists of three separate protocol headers.



The frames used to implement the protocol consist of Pseudo UDP packets along with a user defined datagram payload.



The pseudoARP example is used here to translate the RCE IP address to a MAC address suitable for insertion into the XAUI MAC header.

Part One: Looking at the Pseudo UDP Example Code

1 Open a shell on the host machine and change directory to your workshop udp_example directory.

```
pb-d-128-141-165-118:~ Sergio$ ssh
smaldona@hpslac01
Password:
Last login: Thu Jan 29 15:54:34 2015 from pb-d-
128-141-165-118.cern.ch
[smaldona@hpslac01 ~] cd
workshop_examples/udp_example
[smaldona@hpslac01 ~] ls
Rx.c  Socket.c  Socket.h  Tx.c  build.sh
```

Outputs, Notes and Comments

Note the contents of the example directory. It contains several source files and an SDK build script.

2 Open Socket.h to inspect the class and struct definitions.

```
class Datagram {
public:
  Datagram() {};
 ~Datagram() {};
public:
  uint32_t  length()  {return  _content[0];}
  uint32_t  host()    {return  _content[1];}
  uint32_t  port()    {return  _content[2];}
  uint32_t* content() {return &_content[4];}
public:
  void free();
private:
  uint32_t _content[1 << SIZEOF_DATAGRAM];
  };

/*
**
*/

class Socket {
public:
```

Outputs, Notes and Comments

- The Socket class uses the PseudoARP class from the previous lesson.
- The Datagram class is used to encapsulate the source identifier and payload.

```
  Socket(unsigned numof_buffers);
  ~Socket() {};
public:
  void      sendTo(uint32_t dst_adr, uint32_t
dst_port, uint32_t* datagram, uint32_t
sizeof_datagram);
  Datagram* wait();
  void      close();
private:
  uint64_t    _own_mac;
  uint32_t    _own_adr;
  uint32_t    _own_port;
  uint32_t    _mib;
  uint32_t*   _inbound_datagram;
  SAS_Session _inbound_session;
  SAS_Mbx     _inbound_mbx;
  SAS_Session _outbound_session;
  SAS_ObMbx   _outbound_mbx;
  examples::PseudoARP* _arp;
  };

/*
**
*/

typedef struct {
  uint8_t   version;
  uint8_t   tos;      // Type Of Service
  uint16_t  length;   // Total datagram length;
  uint16_t  id;
  uint16_t  offset;
  uint8_t   ttl;
  uint8_t   protocol;
  uint16_t  checksum;
  uint32_t  src;
  uint32_t  dst;
  } IpHeader;

/*
```

```
**
*/

typedef struct {
  uint16_t src;
  uint16_t dst;
  uint16_t length;   // Total datagram length;
  uint16_t checksum;
  } UdpHeader;

/*
**
*/

typedef struct {
 Xaui_Header ethernet;
 IpHeader    ip;
 UdpHeader   udp;
 } UdpPacket;
```

3 Open the source file for the transmitter TX.c

Outputs, Notes and Comments

This file performs the following actions:

- Instantiate a socket
- Instantiate a datagram
- Populate the datagram with a string message
- Transmit the datagram to a receiver
- Wait for a response datagram to arrive
- Inspect the contents of the received datagram
- Close the socket

```
static const char Message[] = "Hello from xaui TX
Example";

void Task_Start(int argc, const char** argv)
  {
  Location* result;

  if(argc != 3) return;

  Address address(argv[1]);

  Client client;

  result = client.lookup(address);

  uint32_t dst = result->layer3.addr();

  Socket *socket = new Socket(32);
```

```
  Datagram *datagram = new Datagram();

  strncpy((char*)datagram-
>content(),Message,sizeof(Message));

  uint32_t port = strtoul(argv[2], 0, 0);

  printf("Tx example send to ip dst 0x%x port
0x%x\n",dst,port);

  socket-
>sendTo(dst,port,(uint32_t*)datagram,128);

  printf("Tx example socket waiting for
response...\n");

  Datagram *rx = socket->wait();

  if(!rx)
  {
  printf("Socket receive error\n");
  return;
  }

  printf("Tx example received new datagram 0x%x
from host 0x%x port 0x%x len %d\n",
          rx,rx->host(),rx->port(),rx->length());

  uint32_t *data = (uint32_t*)rx->content();

  printf("Datagram message: %s\n",(char*)data);

  socket->close();
  }
```

4 Open the source file for the receiver Rx.c

Outputs, Notes and Comments

This file performs the following actions:

- Instantiate a socket
- Wait for a datagram to arrive
- Inspect the contents of the received datagram
- Fill a response datagram with a string message
- Transmit the response datagram to the sender
- Close the socket

```
static const char Message[] = "Hello back from
xaui RX Example";

void Task_Start(int argc, const char** argv)
  {

  Socket *socket = new Socket(32);

  printf("Rx example socket waiting...\n");

  Datagram *rx = socket->wait();

  if(!rx)
  {
  printf("Socket recevie error\n");
  return;
  }

  printf("Rx example received new datagram 0x%x
```

```
from host 0x%x port 0x%x len %d\n",
          rx,rx->host(),rx->port(),rx->length());

  uint32_t *data = (uint32_t*)rx->content();

  printf("Datagram message: %s\n",(char*)data);

  uint32_t dst = rx->host();

  Datagram *datagram = new Datagram();

  strncpy((char*)datagram-
>content(),Message,sizeof(Message));

  uint32_t port = rx->port();

  socket-
>sendTo(dst,port,(uint32_t*)datagram,128);

  socket->close();
}
```

5

Outputs, Notes and Comments

Open the source file for the Socket implementation Socket.c

```
/*
**
*/

int lnk_prelude(void* prefs, void* elf)
 {

 Xaui_Bind(PSEUDO_IP, (SAS_IbHandler)_vector,
&_active);
```

```
 return 0;

 }

/*
**
*/

Socket::Socket(unsigned numof_buffers)
 {

 SAS_Session session = SAS_Open();

 const SAS_Attributes *attributes =
SAS_GetAttributes(Xaui, session);

 if(!attributes) return;

 _max_data_in_header = (attributes->mib << 3) -
sizeof(UdpPacket);

 SAS_Mbx     IbMbx   = SAS_Bind(_process_inbound,
&_inbound_datagram, session);
 SAS_MbxId   port    = SAS_Id(IbMbx);

 _inbound_datagram = (uint32_t*)0;
 _inbound_session  = session;
 _inbound_mbx      = IbMbx;

 SAS_ObMbx ObMbx   = SAS_ObBind(Xaui,
_process_outbound, (void*)0, session);

 _outbound_session  = session;
 _outbound_mbx      = ObMbx;

 _own_mac           = Xaui_Mac();
 _own_port          = port;

 printf("%s: using RX port
```

```
%d\n",__func__,_own_port);

 service::net::Interface interface;

 _own_adr          = interface.ipAddr();

 _arp              = common_pseudoarp_instance;

 _register(port, numof_buffers);

}

/*
**
*/

Datagram* Socket::wait()
 {

 SAS_EnableWait(_inbound_session);

 return (Datagram *)_inbound_datagram;
 }


void Socket::sendTo(uint32_t dst_adr, uint32_t
dst_port, uint32_t* datagram, uint32_t
sizeof_datagram)
 {

 if(!sizeof_datagram) return;

 SAS_ObMbx mbx   = _outbound_mbx;
 SAS_Frame frame = SAS_ObAlloc(mbx);

 if(!frame) return;

 SAS_Fd* fd = SAS_ObFd(frame,mbx);
```

```
 UdpPacket* packet = (UdpPacket*)fd->header;

 _mac(_arp->lookup(dst_adr) >> 16, &packet-
>ethernet.dst[0]);
 _mac(_own_mac,                    &packet-
>ethernet.src[0]);

 packet->ethernet.type  =
BSWP__swap16(PSEUDO_IP);
 packet->ethernet.mbz   = 0;

 packet->ip.version    = IPV4_VERSION;
 packet->ip.protocol   = IPV4_UDP;
 packet->ip.checksum   = 0;
 packet->ip.id         = 0;
 packet->ip.offset     = 0;
 packet->ip.length     = sizeof(IpHeader) +
sizeof(UdpHeader) + (sizeof_datagram << 2);
 packet->ip.tos        = 0;
 packet->ip.src        = _own_adr;
 packet->ip.dst        = dst_adr;
 packet->ip.checksum   = _checksum(&packet->ip);

 packet->udp.src       = _own_port;
 packet->udp.dst       = dst_port;
 packet->udp.length    = sizeof(UdpHeader) +
(sizeof_datagram << 2);
 packet->udp.checksum  = 0;

 uint32_t* payload = datagram;

 *(uint32_t*)(packet + 1) = *payload++;

 fd->payload = (void*)payload;
 fd->size    = (sizeof_datagram - 1) << 2;
 fd->mid      = SAS_ObId(mbx);
 fd->message = (void*)0;

 rtems_cache_flush_multiple_data_lines(fd-
```

```
>payload,
                                          fd->size);

 SAS_ObPost(SAS_OB_PAYLOAD_RUNDOWN,
SAS_ObSet(frame, XAUI_FRAME_TYPE, HEADER_SIZE),
mbx);

 SAS_EnableWait(_outbound_session);

 return;
 }

static void _vector(SAS_Frame frame, SAS_Arg arg,
SAS_IbMbx mbx)
 {
 if(SAS_IbError(frame)) goto rundown;
 SAS_Fd *fd;
 fd = SAS_IbFd(frame, mbx);

 UdpPacket* packet;
 packet = (UdpPacket*)fd->header;

 if(_checksum(&packet->ip))             goto
rundown;

 if(packet->ip.version != IPV4_VERSION) goto
rundown;

 if(packet->ip.protocol != IPV4_UDP)    goto
rundown;

 int length;
 length = (int)packet->udp.length -
sizeof(UdpHeader);

 if(length < 0) goto rundown;

 int max_in_header;
 max_in_header = _max_data_in_header;
```

```
 int remaining;
 remaining = SAS_IbPayload(frame) ? max_in_header
: length;

 length -= remaining;

 if(length < 0)                goto rundown;
 if(remaining > max_in_header) goto rundown;

 uint32_t port;
 port = packet->udp.dst;

 if(port > MAX_PORTS) goto rundown;

 void** active;
 active = (void**)arg;

 void*  buffer;
 buffer = active[port];

 if(!buffer) goto rundown;

 uint32_t* datagram;
 datagram = (uint32_t*)mem_rsAlloc(buffer);

 if(!datagram) goto rundown;

 datagram[0] = length;
 datagram[1] = packet->ip.src;
 datagram[2] = packet->udp.src;
 datagram[3] = port;
 datagram[4] = (uint32_t)buffer;

 uint32_t* src;
 src = (uint32_t*)(packet+1);
 src += 4;

 uint32_t* dst;
```

```
 dst = &datagram[4];

 while(remaining--) *dst++ = *src++;

 fd->payload = (void*)dst;
 fd->size    = length;
 fd->mid     = port;
 fd->message = (void*)datagram;

 SAS_IbPost(SAS_IB_PAYLOAD_RUNDOWN, frame, mbx);

 return;

 rundown:

 SAS_IbPost(SAS_IbPayload(frame) ? SAS_IB_FLUSH :
SAS_IB_FREE, frame, mbx);
 }

/*
**
*/

static uint32_t _process_inbound(SAS_Message
message, SAS_Arg arg, SAS_Mbx mbx)
 {

 uint32_t** result   = (uint32_t**)arg;

 if(SAS_Error(message))
  {
  printf("Inbound error detected\n");
  *result = 0;
  }
  else
  {
  uint32_t*  datagram = (uint32_t *)message;
  *result = datagram;
  }
```

```
 return SAS_DISABLE | SAS_ABORT;
 }

/*
**
*/

static uint32_t _process_outbound(SAS_Message
message, SAS_Arg arg, SAS_ObMbx mbx)
 {
 if(SAS_Error(message)) printf("Transmit error
detected\n");
 return SAS_DISABLE | SAS_ABORT;
 }
```

Part Two: Running the Pseudo UDP Examples

The execution of the examples takes place on two separate RCEs. One node runs the receiver examples, while the other node runs the transmitter. Both nodes make use of the socket.so and pseudoARP.so shared libraries.

Open a new shell on the host, source the environment script, cd to the udp_example directory, and execute the build script.

1	Open a new shell on the host, source the environment script, cd to the udp_example directory, and execute the build script. Leave this window open so it can be used later in the exercise.
	
	`pb-d-128-141-165-118:~ Sergio$ ssh smaldona@hpslac01`
	`Password:`
	`Last login: Thu Jan 29 15:54:34 2015 from pb-d-128-141-165-118.cern.ch`
		Outputs, Notes and Comments

```
[smaldona@hpslac01 ~]$ source
/home/workshop/envs.csh
Starting RPT environment for V0.7.1-WS
 [smaldona@hpslac01 ~]$ cd
/home/smaldona/workshop_examples/udp_example/
[smaldona@hpslac01 udp_example]$ ./build.sh
[smaldona@hpslac01 udp_example]$
```

2

Open a new shell on the host and source the environment script. Open an ssh session on your DTM (login root pw root), then open a serial console on the RCE.

Outputs, Notes and Comments

```
pb-d-128-141-165-118:~ Sergio$ ssh
smaldona@hpslac01
Password:
Last login: Thu Jan 29 15:54:34 2015 from pb-
d-128-141-165-118.cern.ch
[smaldona@hpslac01 ~]$ source
/home/workshop/envs.csh
Starting RPT environment for V0.7.1-WS
[smaldona@hpslac01 ~]$ ssh root@`atca_ip
snowbird/1/4/0`
root@192.168.210.254's password:
Last login: Wed Dec 31 18:38:59 1969 from
192.168.210.9

[root@snowbird/1/4/0 ~]# minicom -w bay0.0

Welcome to minicom 2.7

OPTIONS: I18n
Compiled on Jan 11 2014, 04:10:34.
Port /dev/ttyUSB1

Press CTRL-A Z for help on special keys
```

```
[/] #
```

3	Create a mount point, mount your nfs directory, and assign the examples namespace. Execute the sysinfo command to verify your RCE address.	**Outputs, Notes and Comments**

```
 [/] # mount -t nfs
192.168.210.9:/home/workshop/workshop_examples/compiled
/mnt/nfs
mounted
192.168.210.9:/home/workshop/workshop_examples/compiled
-> /mnt/nfs
[/] # ns_assign examples /mnt/nfs
[/] #
[/] # sysinfo
Firmware info:
  bitfile version: 0xdb00001a
  ARM module version: 0x12
  Dpm10G: Built Tue Jan 13 12:52:05 PST 2015 by rherbst
  refclk0 freq sel: 0x2
  refclk1 freq sel: 0x2
BSI info:
  HW ID: 0x120000007eb55a70
  RCE ID: snowbird/1/0/0
  BSI U-Boot version: rce.0.4.0
  BSI SW version: 3707
```

4	Partner with another participant, and chose to run the RX or TX example on your RCE. The RX example MUST be executed BEFORE the TX example. If executing the RX example, proceed to step 5. If executing the TX example, proceed to step 6.	**Outputs, Notes and Comments**

5	RX STEP: To run the receiver executable on this RCE, execute the run command.
	
	Make note of the RX port number printed by the receiver. This is the port number that will used by the TX example.
	
	`[/] # run examples:rx.exe`
	`Constructing PseudoARP DB.`
	`This may take some time if the shelf is not completely populated...`
	`Done.`
	`Socket: using RX port 2`
	`Rx example socket waiting...`
	`[/] # Rx example received new datagram 0x15800000 from host 0x5D2A8C0 port 0x4 len 428`
	`Datagram message: Hello from xaui TX Example`
	`Rx example send response to ip dst 0x5D2A8C0 port 0x4`
		Outputs, Notes and Comments

6	TX STEP: To run the transmit executable on this RCE, execute the run command. This executable takes two command line arguments: • The RCE ID string identifier of the datagram receiver (see output of step 3) • The port number on the receiver node (See output of step 5 from	**Outputs, Notes and Comments**

partner). This port number is printed to the receiver console when the example is executed.

```
[/] # run examples:tx.exe snowbird/1/0/0 2
[/] # Socket: using RX port 4
Tx example send to ip dst 0x1D2A8C0 port 0x2
Tx example socket waiting for response...
Tx example received new datagram 0x15900000 from host
0x1D2A8C0 port 0x2 len 428
Datagram message: Hello back from xaui RX Example
```