Voice-over-IP Telephone

-System Design-

Embedded Systems

Spring, 2005

Raj Bakhru {rb2137}
Colin Gilboy {crg2012}
Sam Jenning {smj2008}
@columbia.edu

Table of Contents

Project Overview

Our team set out to create a VoIP phone interface using the available peripherals on the Spartan 2E FPGA board. In our design, incoming audio goes through the AK4565 codec, is converted to u-law format via the Microblaze and then placed in packets in SRAM. At the same time, the ethernet controller establishes a connection with the destination address and initiates SIP protocol. When the connection is acknowledged, packets containing audio data are sent to the other end of the line where they are decoded.

What We Accomplished

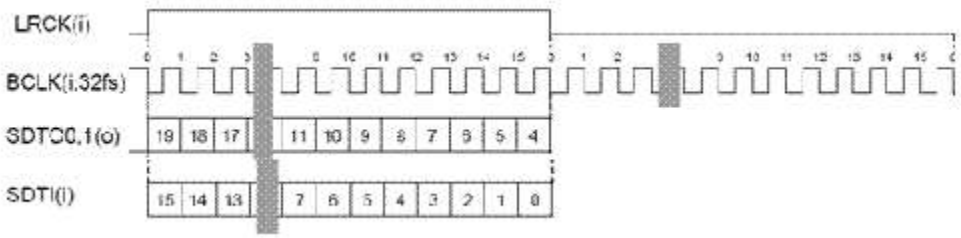
Transmission of audio data is functional with steadily-increasing, if low, fidelity. Reception is not yet reliable, but control structure for reception and audio decoding exists. At present, the system is able to "dial" any of the ROLM phones on campus (which are VoIP-capable) and initiate a unidirectional conversation.

Audio Overview

We used the XESS tool, gxssetcodec, to configure the AK4565 codec to take input from the microphone input (labeled “ext” in the documentation), though this was not entirely necessary. The codec defaults to taking input from two pins accessible on the board directly (intL0, intR0), but using the microphone input, we lessened the number of wires hanging off of the board.

We wrote a peripheral to handle the input of the codec and communicate with the Microblaze. The audio is downsampled in the peripheral to 8 kHz (the standard sampling frequency for G.711 u-law, as defined by the ITU recommendation). Each sample is sent via interrupts to the Microblaze microprocessor. This audio data is converted from 16-bit linear to 8-bit u-law using Sun's freely available conversion code (ftp://ftp.cwi.nl/pub/audio/ccitt-adpcm.tar.gz). This data is stored in a byte array in the processor's data memory until 160 samples have been gathered. It is then transferred to the memory on the ethernet interface.

Since the audio peripheral interrupts the processor every time it has a sample ready (8000 times per second), the processor uses the same interrupt routine to write data from packets received via ethernet (i.e. from the other phone) when they are available. The processor converts the incoming packet data from 8-bit u-law to 16-bit linear, and then the AK4565 peripheral copies that value 6 times as sequential samples, so that it has enough samples to run at 48 kHz (the sampling rate of the AK4565 codec).



Audio Interface

This is the timing diagram for AK4565 Audio Codec, ignoring stereo input (taken from the Asahi Kasei datasheet, modified for our purposes). When LRCK is

low, the codec samples from the right channel, but we are only interested in monophonic recording, so we will ignore the right channel and use the left as the only signal. Running the bit clock (BCLK) at 32 times the sampling frequency (LRCK) we can force the 20-bit codec to serially output 16-bits. We read and write on LRCK high (so the peripheral interrupts the processor on LRCK low), which in theory gives the processor plenty of time to handle ethernet.

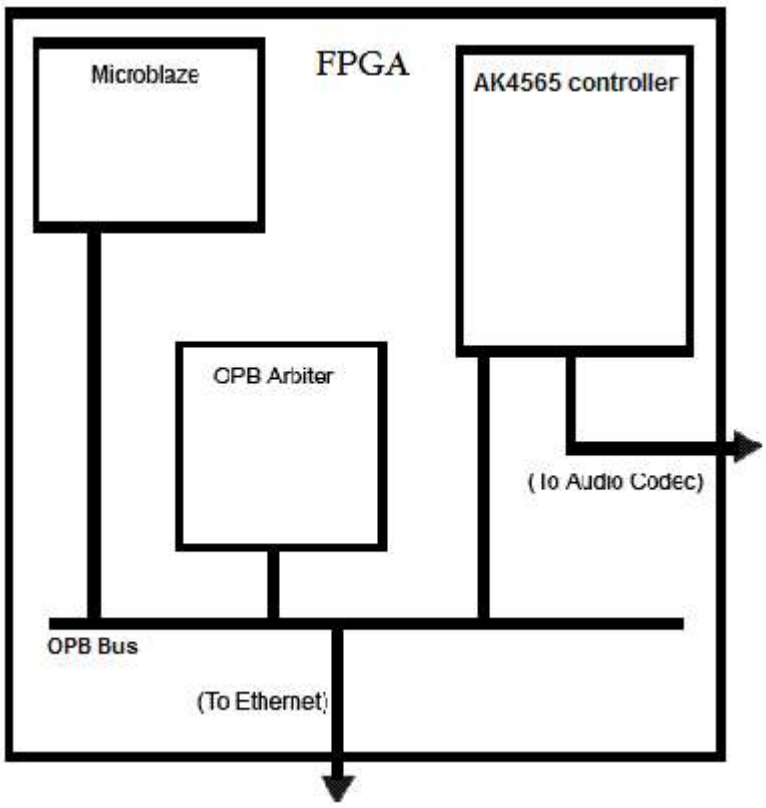


Illustration 1System Overview

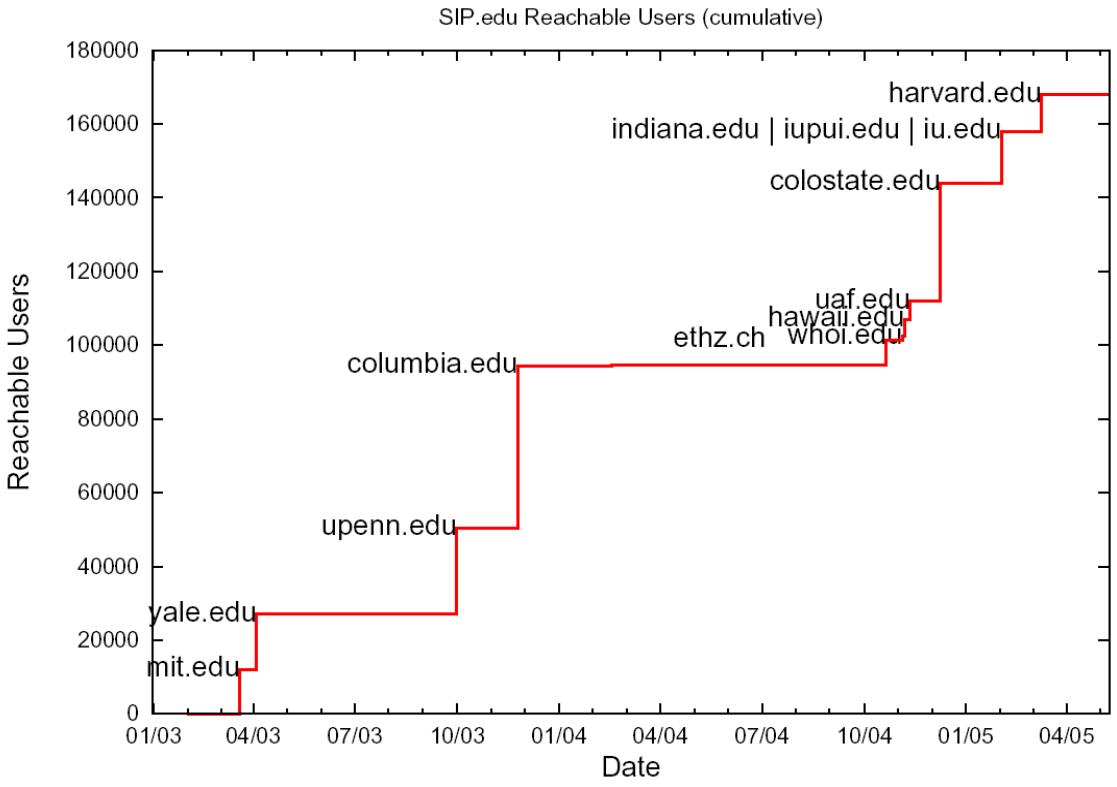
Ethernet and VoIP Protocol

Voice-over-IP Background / Columbia's SIP System

Columbia participates in an experimental SIP Voice-over-IP setup utilizing Internet2. This program incorporates 11 universities and research institutions each with local SIP gateways that all connect to each other. In this way, VoIP calls from Columbia can be routed to MIT, and vice-versa. Because our system is directed through the university gateway, modifying the caller address from "@columbia.edu" to "@colostate.edu" would modify the destination of the user's call.

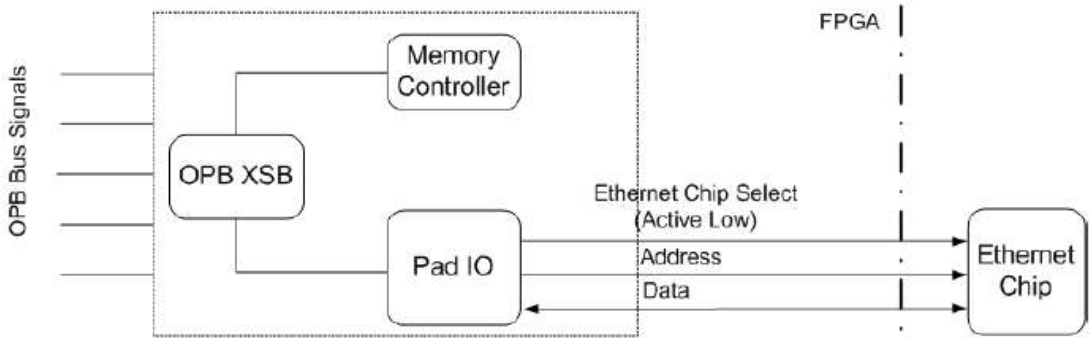
SIP gateways provide routing for IP to IP calls. Additionally, most act as a gateway to the POTS network (plain-old-telephone system). For example, ROLM phones at Columbia, while digital, are not based on Voice-over-IP. VoIP calls for a ROLM number are converted to analog and re-converted onto the ROLM proprietary digital system.

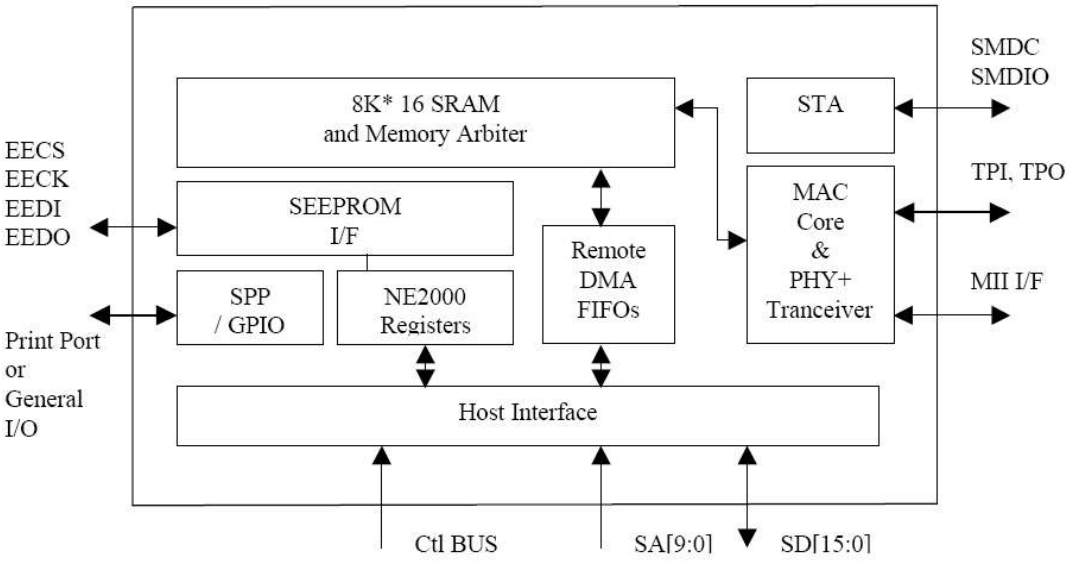
Because SIP is a standard protocol, modifying the system to use a different gateway is not difficult. Columbia's Computer Science department, for example, runs their own VoIP system which connects to the university's system.



Ethernet Controller

Team JAYCam, from last year's class, was able to successfully use the ethernet controller. Our implementation of the Ethernet was based on theirs. Modifications were made to remove extraneous bloat from their code that proved irrelevant to our project. Additionally, minor modifications were necessary to allow the Ethernet to properly receive packets.





Memory Addressing:
NE2000 registers – 0xA00400 – 0xA0041F
SRAM Local Memory – 0xA04400 – 0xA0BFFF

Ethernet Buffers

The Ethernet 8k x 16 SRAM acts as a buffer for both incoming and outgoing data. Because we only queue/buffer a single packet type for retransmission (RTP), the necessary outgoing buffer size was small; the majority of the SRAM buffer was

dedicated to receiving. Data is transmitted by performing a Remote DMA write. Data is received by performing a Remote DMA read. Data is written as a 16-bit word with the Ethernet controller operating in ISA mode; when transmitting, the latter byte should take the upper 8 bits of the word. When receiving, data is received in order, with the latter byte in the lower 8 bits.

The page registers are set as follows:
Transmit start buffer: 0x41
Transmit end buffer: 0x49
Receive start buffer: 0x4A
Receive end buffer: 0x80

When the last receive buffer is encountered, the buffer wraps around, creating a receive ring. Packets that are longer than the maximum size of the buffer (256 bytes) are stored in linked pages. The 4 byte NIC-appended packet header indicates the existence of linked pages, along with the amount of data received. Given this information, a remote DMA read is performed to obtain the packet data on the page. For our particular use, the linked data can be ignored in call cases except that of a SIP 200/OK, in which case we must parse the tag identification value in the SIP header "To" field. The "To" is not in the upper 256 bytes of the packet, and thus, we increment the address appropriately to obtain the linked data, which is then parsed for the tag.

To receive packets, we constantly poll the ISR (and, for debugging, the RSR, which contains more detailed information on the type of packet received) NE2000 registers. This can also be implemented with interrupts, but, because of lack of time to create an interrupt controller (necessitated by having both audio and ethernet operating with interrupts), we avoided this. Once a packet is received, the ISR holds a 1 in the packet received field (the least significant bit) and the packet receive handler is run to determine the packet type and process it appropriately.

RTP packets need to be sent very quickly (ideally, every 0.016 seconds); thus, instead of re-loading the framework of the RTP from the Toshiba SRAM and appending the sequence, timestamp, and payload, the framework is loaded a single time from the Toshiba SRAM and stored in a separate buffer page for re-use and faster transmission. The time difference between transmissions dropped drastically

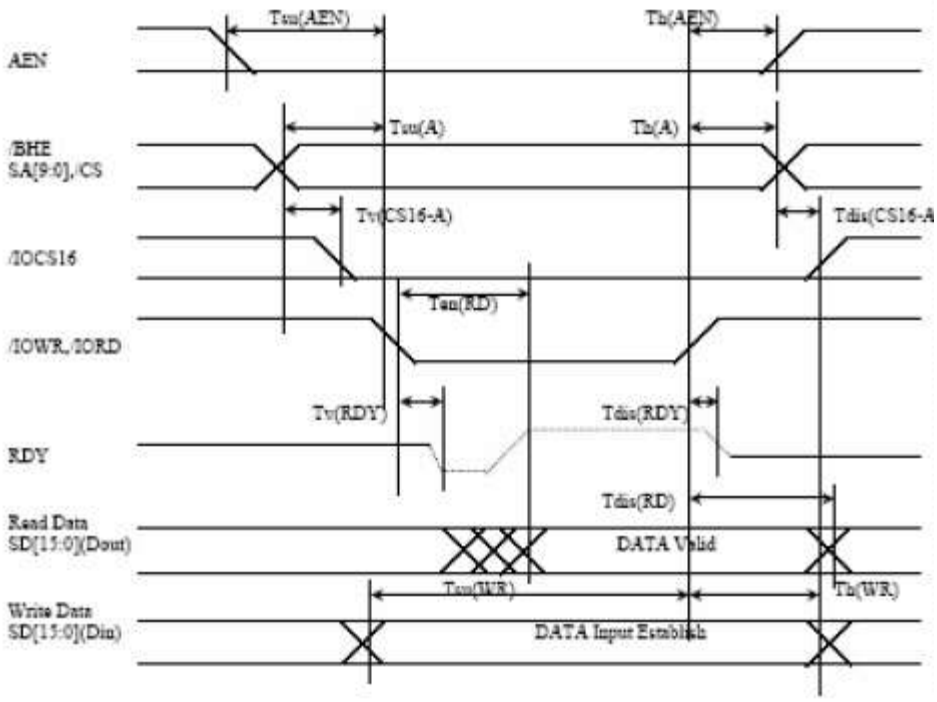
by using this policy, as the number of OPB transfers was more than halved; our actually transmit time between RTP packets is approximately 0.026 seconds.

Timing – General

The 100MHz clock implemented by JAYCam was tested and provided no extra benefit, aside from complicating the padding procedures, so we removed it and reverted to the OPB/System clock.

As noted by JAYCam, the timing diagram provided by the manufacturer is incorrect. The CS, BHE, AE, Add, and WE signals cannot be asserted at the same time, but rather require delay, with CS coming before WE, and all after an overall 5ns initial setup time. This yields a 4-step FSM: 1) All signals except WE asserted 2) WE asserted (making the address and data signals valid) 3) Wait cycle (because DMA may be too slow) 4) Data disabling

Timing Diagram



Ethernet Driver

The Ethernet controller is configured through NE2000 registers. JAYCam developed an initialization sequence that sufficed for transmission, but additional

configuration was required to enable reception. In particular, the NIC MAC address had to be set, as well as the reception mode. Globally, the current receive page and location had to be stored for proper retrieval of received packets (and removal from the receive buffer ring to prevent a buffer overrun), while the registers holding the pointers to the current write, read, and binary pages had to be written and updated at the NIC.

OPB Arbitration

The SRAM resides on the OPB, with the Ethernet, sharing PB_A and PB_D ports. Thus, simple arbitration of the OPB takes place to provide control to either the Ethernet or SRAM. The OBP arbiter resides on the memory range of 0x800000 to 0xffffff. This range includes both the active SRAM and Ethernet memory address ranges.

Network Transport

The Ethernet packets are preloaded in the SRAM, as discussed later. The location of these packets in the SRAM is known and static. At runtime, when the packets are needed for transmission, a procedure integrating the Toshiba SRAM and Ethernet SRAM is used – two bytes of the packet are brought from the Toshiba SRAM to the Microblaze at a time, generating a 16-bit word that is then transferred with a DMA write to the Ethernet SRAM.

RTP packets are handled differently due to the time sensitive nature of their transmission. The RTP packet framework is loaded at the start of the audio sending sequence, which commences either once an ACK packet is sent in response to the SIP/183 Session Progress or SIP/200 OK, or once a call is initiated. The RTP packet is also modified to send to the proper gateway audio port by waiting until it receives an RTP packet and can parse out the source port (this proved to be much easier and just as effective as parsing the SDP of the 183 or 200 requests).

Additionally, the ACK packet requires special consideration because it must include an extra identifier (tag) that is sent along with the 183 or 200 requests. This identifier is randomly generated, alphanumeric, and of variable length. This variable length requires that we push back the rest of the packet framework appropriately where it is inserted, as well as modify the checksum, packet length, and UDP length

fields of the ACK packet. The checksum change is handled through a lookup table, as the tag is always between 11 and 15 characters long. The length field positions are hardcoded for quick changes.

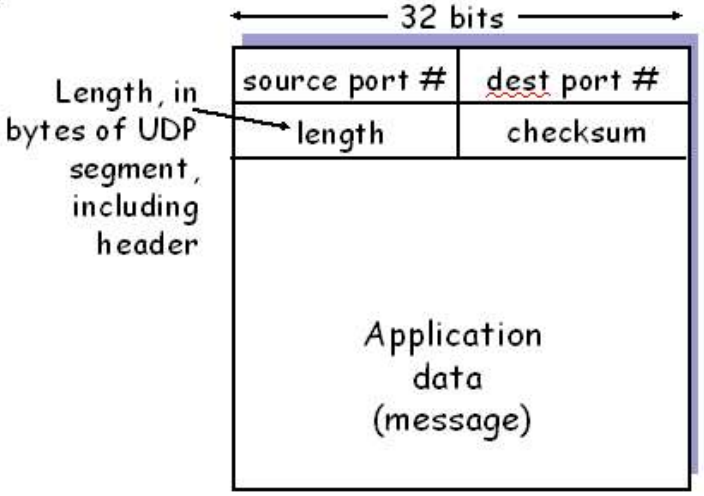
The structure of the IP packet is:

Destination Address	6 Bytes
Source Address	6 Bytes
Length / Type	2 Bytes
Data (Pad if < 46 Bytes)	46 Bytes Min.

General Transmit Packet Format

This is not the standard structure for an IP packet, but rather the structure of the packet as received by the ethernet controller, where the data is taken and formed into a standard IP packet. The checksum, etc. are automatically handled by the controller.

The UDP packet is placed within the IP packet and has the following structure:



The application data message is specific to the protocol.
The protocols we implemented minimally are: ARP, SIP, SDP, and RTP.

ARP Packets

ARP (Address Resolution Protocol) packets are sent within IP (not UDP) frames, and they are used to provide routing on the local subnet. The source is requesting to know the physical hardware address of the destination. To be able to receive packets we must properly register the board on the network with ARP. Normally, an ARP request is broadcast to the network and a response is sent via unicast to the requester. Instead of implementing the procedures to detect and respond to ARP requests, we utilized gratuitous ARP broadcasts to broadcast the existence and MAC of the board.

A gratuitous ARP packet differs from a normal ARP packet in that the source and destination MAC addresses are the same. The ARP packet is broadcast to the network, and all interested parties (in our case, the gateway/router) will update their ARP cache to include an entry for our MAC and IP.

Alternatively, because we assigned the Ethernet controller the same MAC address as one of our laptops, we could simply connect the laptop for a few seconds, allowing all ARP queries to be satisfied, and then switch the Ethernet cable to the FPGA board.

SIP/SDP

SIP (Session Initiation Protocol) is implemented over UDP/IP, and it will be used to communicate properly with standard VoIP services. To accomplish this, we also need to implement SDP and RTP. SDP (Session Description Protocol) is used upon connection to agree on a codec, transmission port, rate, etc. In our case, we support only G.711 u-law at 8 kHz, and the SDP response will indicate this. RTP (Real-Time Transport Protocol) is used for audio data transfer during phone calls. RTP is implemented by passing a sequence number, timestamp, identifier, and payload audio data in a UDP/IP packet. Both incoming and outgoing RTP audio for 711mu-law at 8 kHz will include 160 samples/packet. Actual packet sniffed data from our project is provided in the appendix.

SIP Reference: http://www.faqs.org/rfcs/rfc3261.html

SIP Sequence of Events:

-- Gratuitous ARP packet (attached packet #1) --

#2: “INVITE” (attached packet #2)
Phone to gateway:
Tells the gateway who to add to the phone call; implements SDP in message body to dictate CODECs available

#3: “100 Trying” (attached packet #3)
Gateway to phone:
Acknowledges an attempt to call the specified address

#4: “183 Session Progress” with SDP (attached packet #4)
Gateway to phone:
Equivalent to knowing that the phone is ringing; SDP indicates selected CODEC

--RTP incoming stream starts --

#5: “200 OK” with SDP (attached packet #5)
Gateway to phone:
Phone answered; SDP confirms/changes selected CODEC

#6: “ACK” (attached packet #6)
Phone to gateway:
Tells the gateway its about to send the audio UDP/RTP stream

--RTP outgoing stream starts (packet #7) --

#8: “BYE” (attached packet #8)
Either direction:
closes the call

Memory

BRAM

It was critical that we extended our instruction memory space. It was also very simple. The initial RAM space allocated in lab 6 (the skeleton off which we built some of our project) is 0x0000 to 0x0FFF address lines each for instruction and data. We ultimately doubled our instruction and data memory.

There are multiple ways to extend memory without using the instruction cache (the icache never did anything to help fit our main.c file on the microblaze anyway). One way is to extend the address range in the mhs file for instruction and data memory. This will change the way RAM is implemented in the FPGA- in our case, we extended the address range up to 1FFF lines, which resulted in the use of s2_s2 (2 bit input/output) RAMs. Another option is to add extra RAM elements in the mhs file, making sure they are connected properly to control units and the the LMB. This will cause the system to implement two separate blocks of memory. You can then concatenate them in the system_bd.bmm file. The resultant memory covers the same address range but with longer words at each address. Testing has shown that the favorable configuration to use is block memory with smaller data width. As of this writing, we are at roughly 60% storage capacity.

Toshiba SRAM

The Toshiba TC55V16256J 256k x 16 byte SRAM module provides a total of 512 KB of storage. This storage space is used to store the various Ethernet packets utilized by the phone. The memory offsets for the locations of these packets are below.

The packets are placed in the SRAM through a separate preloader module. A tool provided by XESS, xsload, allows us to preload the SRAM. The SRAM is then accessed at runtime and the data is transferred to the Ethernet in a Remote DMA write as described in the Ethernet software chapter.

The SRAM resides on the memory address range beginning at 0xC00000

Packets:

Gratuitous ARP: 0xC00254 (offset of 596) to 0xC00290 (offset of 656)

Length = 60 bytes

SIP/SDP Invite: 0xC00000 (offset of 0) to 0xC00253 (offset of 595)

Length = 595 bytes

SIP/SDP ACK: 0xC00290 (offset of 656) to 0xC00428 (offset of 1064)

Length = 409 bytes

NOTE: This packet is designed with a built-in "tag" size of 10. The tag, as discussed later, is variable length, meaning this size alters slightly

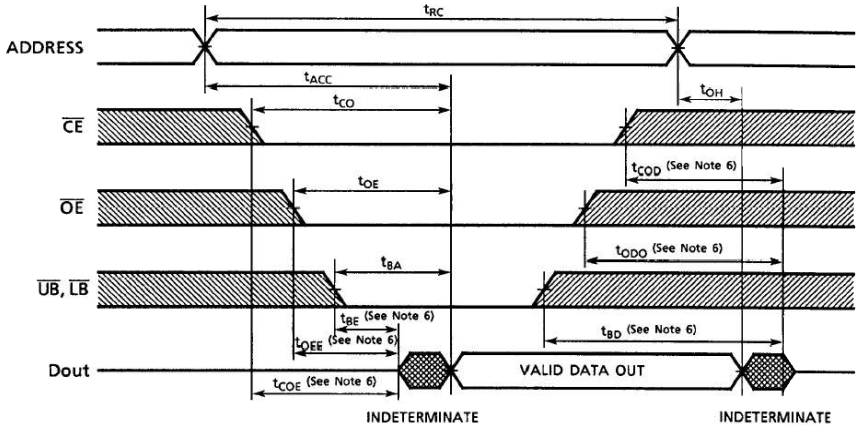
RTP: 0xC00429 (offset of 1065) to 0xC00455 (offset of 1109)

Length = 44 bytes

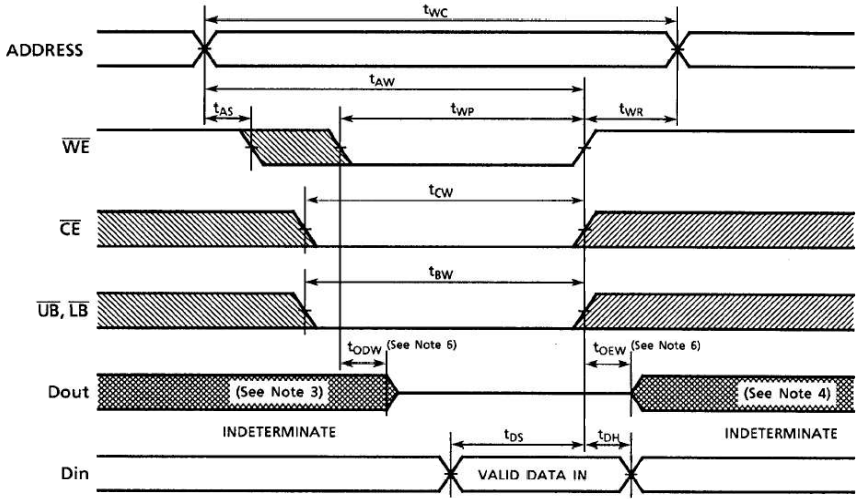
NOTE: This length doesn't meet the minimum packet size, but the packet is designed to be appended with the sequence number, timestamp, and payload audio data

TIMING DIAGRAMS

READ CYCLE (See Note 2)



WRITE CYCLE 1 (WE CONTROLLED) (See Note 5)



Notes from the VoIP Team

What Colin Learned:

Integrating into a large pre-existing system, with strict standards is significantly more difficult than creating one's own standards. Working on the audio, I was forced to abide by the bitrate and codec standards that are demanded by the ITU's G.711 recommendation added significant difficulty to my work on the audio. Using audio hardware that is designed for my purposes would have been beneficial. In terms of teamwork, I learned that it is important to sleep, to audit each other's code when sleep is deprived and also that taking on large responsibilities by one's self for no reason does not help anyone. Sharing responsibility and being familiar with the inner workings of the entire project are both necessary.

If I had to do it all over again, I would have read documentation more thoroughly before I began planning. Then I would have planned on paper. Then I would have slept for awhile, and checked if I was making sense. Then I would have coded. Organization in the early phases makes a huge difference.

What Sam Learned:

Early on, I fell behind. One day I was in the lab as much as the rest of my group, trying to figure out how to use the audio codec. The next day I had no idea what was going on. From then on it was a struggle to keep up. Colin and Raj often went to the lab without telling me in advance, they often stayed in the lab later than I did, and I didn't do much about it except complain and feel upset. If there's a lesson here, it's that group work sessions need to be scheduled further in advance. It was hard for me to get out of club meetings, drop my current work, or otherwise change my plans when all the forewarning I get is “we're going to the lab now”.

Feeling like you aren't pulling your own weight in a project you care about sucks. Hard.

I ended up writing outlines and code for structures that didn't make it into our final design. I became very familiar with the Xilinx primitives, especially block memory. Our project is not really distributed over many entities, however, so instantiating library structures never really became that much of an issue. I also spent time on the state machine. On the subject of state machines, I would say that it is essential that you understand the timing of your system *intimately* before actually writing your states.

In terms of things I would have done differently, aside from spending more time in the lab, I would have done more research on the entities that I was trying to build before I started writing code. There's more than one way to do just about anything on these boards, and the first solution you develop is rarely the best. In this sense, it would have been prudent to consider alternative designs before writing anything. For example, memory. We initially wanted to store encoded audio in BRAM before shifting it into packets. I spent probably a dozen hours trying to get this setup into code before we realized that writing directly into SRAM was better (and easier).

What Raj Learned:

Technically, I learned an incredible amount about networking, in particular, UDP, SIP, RTP, ARP, SDP, etc.. OPB Arbitration is not easy or fun, and should be

avoided when possible. If I were to ever create a protocol, never include variable length identifiers; they were the bane of my existence for a good long time. We gained a lot by using other's code, but it would have been more beneficial to initially spend the time to fully understand how that project/code functions. We used a lot of the ethernet code from JAYCam, and because their code worked so well, it didn't initially seem necessary to delve into how it functioned. To augment their code to receive and to send at a fast enough rate, a great deal of room for improvement was found only after understanding why they did certain things they did (and that we didn't need).

Planning is the most critical phase, and we should've spent more time on it. We ended up having a very functional project, but not without increased work later on because of design decisions that may have been questionable. In terms of Ethernet and OPB arbitration, this all worked out OK, but in terms of audio, we ran into some problems. If we had planned better and had more time, the ethernet could've run off interrupts, and RTP would probably then run at a higher rate, making audio quality better. The NIC dropped an incredible number of packets, and that caused a lot of problems for us; when we can't detect that the phone call has been answered, we can't acknowledge the phone call, and then the call is automatically hung up.

In terms of teamwork, we should've worked more together, rather than on separate sections with little interaction until integration. Lack of sleep and the like caused us to make a number of stupid mistakes (e.g. we spent hours trying to figure out why the phone wasn't ringing one night, and it turned out to be because we were missing the “l” in Columbia). In the end, we had to end up learning more about each other's code anyways.

Acknowledgements

The VoIP team would like to thank:

Prof. Edwards, Marcio, and all the previous TA's
Schulzrinne or however you spell his name
Josh Weinberg and team JAYCam
The Nortsam project team

CODE CODE CODE!

(in a tar file some where...)

Files of Interest:

opb_ak4565_v1_00_a:
opb_ak4565.vhd
opb_ak4565_v2_1_0.mpd
opb_ak4565_v2_1_0.pao

opb_cntrl_v1_00_a:
opb_cntrl.vhd
pad_io.vhd
memoryctrl.vhd
opb_cntrl_v2_1_0.mpd
opb_cntrl_v2_1_0.pao

system files:
system.mhs
system.mss
system.ucf
Makefile

c_source_files:
main.c
test.c
phone.h

system.mhs

```
# Parameters
PARAMETER VERSION = 2.1.0

# Global Ports

PORT PB_A = PB_A, DIR = OUT, VEC = [19:0]
PORT PB_D = PB_D, DIR = INOUT, VEC = [15:0]
PORT PB_LB_N = PB_LB_N, DIR = OUT
PORT PB_UB_N = PB_UB_N, DIR = OUT
PORT PB_WE_N = PB_WE_N, DIR = OUT
PORT PB_OE_N = PB_OE_N, DIR = OUT
PORT RAM_CE_N = RAM_CE_N, DIR = OUT

#audio iface:
PORT AU_CSN_N =  AU_CSN_N, DIR=OUT
PORT AU_BCLK =  AU_BCLK, DIR=OUT
PORT AU_MCLK = AU_MCLK, DIR=OUT
PORT AU_LRCK = AU_LRCK, DIR=OUT
PORT AU_SDTI = AU_SDTI, DIR=OUT
PORT AU_SDTO0 = AU_SDTO0, DIR=IN

#ethernet:
PORT ETHERNET_CS_N = ETHERNET_CS_N, DIR = OUT
PORT ETHERNET_RDY = ETHERNET_RDY, DIR = IN
PORT ETHERNET_IREQ = ETHERNET_IREQ, DIR = IN
PORT ETHERNET_IOCS16_N = ETHERNET_IOCS16_N, DIR = IN



PORT FPGA_CLK1 = FPGA_CLK1, DIR = IN
PORT RS232_TD = RS232_TD, DIR=OUT
PORT RS232_RD = RS232_RD, DIR=IN


# Hint: Put your peripheral first in this file so it will be
analyzed
# first and will generate errors faster.

# AK4565 Audio codec peripheral

BEGIN opb_ak4565
 PARAMETER INSTANCE = ak4565
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0xFEFE0000
 PARAMETER C_HIGHADDR = 0xFEFEFFFF
 PORT OPB_Clk = sys_clk
 #PORT PB_D = PB_D
 PORT AU_CSN_N =  AU_CSN_N
 PORT AU_BCLK =  AU_BCLK
 PORT AU_MCLK = AU_MCLK
 PORT AU_LRCK = AU_LRCK
 PORT AU_SDTI = AU_SDTI
 PORT AU_SDTO0 = AU_SDTO0
 BUS_INTERFACE SOPB = myopb_bus
 #Interrupt:
 PORT Interrupt = interrupt
END

# OPB ctrler for ethernet

BEGIN opb_cntrl
```

```
 PARAMETER INSTANCE = arbiter
 PARAMETER HW_VER = 1.00.a

 PARAMETER C_BASEADDR = 0x00800000
 PARAMETER C_HIGHADDR = 0x00FFFFFF
 PORT PB_A = PB_A
 PORT PB_D = PB_D
 PORT PB_LB_N = PB_LB_N
 PORT PB_UB_N = PB_UB_N
 PORT PB_WE_N = PB_WE_N
 PORT PB_OE_N = PB_OE_N
 PORT RAM_CE_N = RAM_CE_N
 PORT ETHERNET_CS_N = ETHERNET_CS_N
 PORT ETHERNET_RDY = ETHERNET_RDY
 PORT ETHERNET_IREQ = ETHERNET_IREQ
 PORT ETHERNET_IOCS16_N = ETHERNET_IOCS16_N
 PORT OPB_Clk = sys_clk
 PORT io_clock = sys_clk

 BUS_INTERFACE SOPB = myopb_bus
END


# The main processor core

BEGIN microblaze
 PARAMETER INSTANCE = mymicroblaze
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_USE_BARREL = 1
 PARAMETER C_USE_ICACHE = 0
 PORT Clk = sys_clk
 PORT Reset = fpga_reset
 BUS_INTERFACE DLMB = d_lmb
 BUS_INTERFACE ILMB = i_lmb
 BUS_INTERFACE DOPB = myopb_bus
 BUS_INTERFACE IOPB = myopb_bus
 #interrupt:
 port INTERRUPT = interrupt
END

# Block RAM for code and data is connected through two LMB busses
# to the Microblaze, which has two ports on it for just this reason.

# Data LMB bus

BEGIN lmb_v10
 PARAMETER INSTANCE = d_lmb
 PARAMETER HW_VER = 1.00.a
 PORT LMB_Clk = sys_clk
 PORT SYS_Rst = fpga_reset
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = lmb_data_controller
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00001FFF
 BUS_INTERFACE SLMB = d_lmb
 BUS_INTERFACE BRAM_PORT = conn_0
END

# Instruction LMB bus
```

```
BEGIN lmb_v10
 PARAMETER INSTANCE = i_lmb
 PARAMETER HW_VER = 1.00.a
 PORT LMB_Clk = sys_clk
 PORT SYS_Rst = fpga_reset
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = lmb_instruction_controller
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00001FFF
 BUS_INTERFACE SLMB = i_lmb
 BUS_INTERFACE BRAM_PORT = conn_1
END

# The actual block memory

BEGIN bram_block
 PARAMETER INSTANCE = bram
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = conn_0
 BUS_INTERFACE PORTB = conn_1
END

# Clock divider to make the whole thing run

BEGIN clkgen
 PARAMETER INSTANCE = clkgen_0
 PARAMETER HW_VER = 1.00.a
 PORT FPGA_CLK1 = FPGA_CLK1
 PORT sys_clk = sys_clk
# PORT io_clock = sys_clk
 PORT fpga_reset = fpga_reset
END

# The OPB bus controller connected to the Microblaze
# All peripherals are connected to this

BEGIN opb_v20
 PARAMETER INSTANCE = myopb_bus
 PARAMETER HW_VER = 1.10.a
 PARAMETER C_DYNAM_PRIORITY = 0
 PARAMETER C_REG_GRANTS = 0
 PARAMETER C_PARK = 0
 PARAMETER C_PROC_INTRFCE = 0
 PARAMETER C_DEV_BLK_ID = 0
 PARAMETER C_DEV_MIR_ENABLE = 0
 PARAMETER C_BASEADDR = 0x0fff1000
 PARAMETER C_HIGHADDR = 0x0fff10ff
 PORT SYS_Rst = fpga_reset
 PORT OPB_Clk = sys_clk
END

# UART: Serial port hardware

BEGIN opb_uartlite
 PARAMETER INSTANCE = myuart
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_CLK_FREQ = 50_000_000
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_BASEADDR = 0xFEFF0100
 PARAMETER C_HIGHADDR = 0xFEFF01FF
```

```
 PORT OPB_Clk = sys_clk
 BUS_INTERFACE SOPB = myopb_bus
 PORT RX=RS232_RD
 PORT TX=RS232_TD
END
```

System.mss

```
PARAMETER VERSION = 2.2.0
PARAMETER HW_SPEC_FILE = system.mhs

BEGIN PROCESSOR
 PARAMETER HW_INSTANCE = mymicroblaze
 PARAMETER DRIVER_NAME = cpu
 PARAMETER DRIVER_VER = 1.00.a
END

BEGIN OS
  PARAMETER PROC_INSTANCE = mymicroblaze
  PARAMETER OS_NAME = standalone
  PARAMETER OS_VER = 1.00.a
  PARAMETER STDIN = myuart
  PARAMETER STDOUT = myuart
END

BEGIN DRIVER
 PARAMETER HW_INSTANCE = myuart
 PARAMETER DRIVER_NAME = uartlite
 PARAMETER DRIVER_VER = 1.00.b
END

# Use null drivers for peripherals that don't need them
# This supresses warnings

BEGIN DRIVER
 PARAMETER HW_INSTANCE = ak4565
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
# PARAMETER INT_HANDLER = audio_int_handler, INT_PORT = Interrupt
END

BEGIN DRIVER
 PARAMETER HW_INSTANCE = bram_peripheral
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
END

BEGIN DRIVER
 PARAMETER HW_INSTANCE = lmb_data_controller
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
END

BEGIN DRIVER
 PARAMETER HW_INSTANCE = lmb_instruction_controller
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
END
```

---

system.ucf

```
net sys_clk period = 18.000;
#net io_clock period = 9.000;

net FPGA_CLK1 loc="p77";

net AU_CSN_N loc="p165";
net AU_BCLK loc="p166";
net AU_MCLK loc="p167";
net AU_LRCK loc="p168";
net AU_SDTI loc="p169";
net AU_SDTO0 loc="p173";

net PB_A<0> loc="p83";
net PB_A<1> loc="p84";
net PB_A<2> loc="p86";
net PB_A<3> loc="p87";
net PB_A<4> loc="p88";
net PB_A<5> loc="p89";
net PB_A<6> loc="p93";
net PB_A<7> loc="p94";
net PB_A<8> loc="p100";
net PB_A<9> loc="p101";
net PB_A<10> loc="p102";
net PB_A<11> loc="p109";
net PB_A<12> loc="p110";
net PB_A<13> loc="p111";
net PB_A<14> loc="p112";
net PB_A<15> loc="p113";
net PB_A<16> loc="p114";
net PB_A<17> loc="p115";
net PB_A<18> loc="p121";
net PB_A<19> loc="p122";

net PB_D<0> loc="p153";
net PB_D<1> loc="p145";
net PB_D<2> loc="p141";
net PB_D<3> loc="p135";
net PB_D<4> loc="p126";
net PB_D<5> loc="p120";
net PB_D<6> loc="p116";
net PB_D<7> loc="p108";
net PB_D<8> loc="p127";
net PB_D<9> loc="p129";
net PB_D<10> loc="p132";
net PB_D<11> loc="p133";
net PB_D<12> loc="p134";
net PB_D<13> loc="p136";
net PB_D<14> loc="p138";
net PB_D<15> loc="p139";


net PB_LB_N loc="p140";
net PB_UB_N loc="p146";
net PB_WE_N loc="p123";
net PB_OE_N loc="p125";


net RAM_CE_N loc="p147";

#Ethernet pins
```

```
net ETHERNET_CS_N loc="p82";
net ETHERNET_RDY loc="p81";
net ETHERNET_IREQ loc="p75";
net ETHERNET_IOCS16_N loc="p74";

net RS232_TD loc="p71";
net RS232_RD loc="p73";
```

---

phone.h

```
#include "xio.h"
#include "xbasic_types.h"

#ifndef BYTE
#define BYTE unsigned char
#endif
#ifndef WORD
#define WORD unsigned short
#endif


// define the size of a packet
#define PACKET_SIZE 81

// NE2000 definitions
#define NIC_BASE (0x00A00400) // Base I/O address of the NIC card
#define DATAPORT (0x10*2)
#define NE_RESET (0x1f*2)


// NIC page0 register offsets
#define CMDR    (0x00*2) // command register for read & write
#define CR      (0x00*2) // command register for read & write

#define PSTART  (0x01*2) // page start register for write
#define PSTOP   (0x02*2) // page stop register for write
#define BNRY    (0x03*2) // boundary reg for rd and wr
#define TPSR    (0x04*2) // tx start page start reg for wr
#define TBCR0   (0x05*2) // tx byte count 0 reg for wr
#define TBCR1   (0x06*2) // tx byte count 1 reg for wr
#define ISR     (0x07*2) // interrupt status reg for rd and wr
#define RSAR0   (0x08*2) // low byte of remote start addr
#define RSAR1   (0x09*2) // hi byte of remote start addr
#define RBCR0   (0x0A*2) // remote byte count reg 0 for wr
#define RBCR1   (0x0B*2) // remote byte count reg 1 for wr
#define RCR     (0x0C*2) // rx configuration reg for wr
#define TCR     (0x0D*2) // tx configuration reg for wr
#define DCR     (0x0E*2) // data configuration reg for wr
#define IMR     (0x0F*2) // interrupt mask reg for wr

// NIC page 1 register offsets
#define PAR0    (0x01*2) // physical addr reg 0 for rd and wr
#define CURR    (0x07*2) // current page reg for rd and wr
#define MAR0    (0x08*2) // multicast addr reg 0 for rd and WR

// Buffer Length and Field Definition Info
#define TXSTART  0x41              // Tx buffer start page
#define TXPAGES  8                 // Pages for Tx buffer
#define RXSTART  (TXSTART+TXPAGES) // Rx buffer start page
#define RXSTOP   0x80              // Rx buffer end page for word
mode
```

```
#define DCRVAL   0x01              // DCR values for word mode

// macros for reading and writting registers
#define outnic(addr, data) XIo_Out16(NIC_BASE+addr, data)
#define innic(addr) XIo_In16(NIC_BASE+addr)



#define PS1 0x80
#define PS0 0x40
#define RD2 0x20
#define RD1 0x10
#define RD0 0x08
#define TXP 0x04
#define START 0x02
#define STOP 0x01


#define  enetpacketstatus     0x00
#define  nextblock_ptr        0x01
#define       enetpacketLenL          0x02
#define       enetpacketLenH          0x03




// Private prototypes
void diag();
int init_etherne();
void resetnic();
void delay(int);
```

---

main.c

```
#include "xparameters.h"
#include "xbasic_types.h"
#include "xio.h"
//#include "types.h"
//#include "pcmu_l.h"
#include "phone.h"

static BYTE incr = 0;

BYTE packet[260];
BYTE audio_out[160];
BYTE tag[15];

BYTE enableSend = 1;
BYTE rtpport1;
BYTE rtpport2;
short isRTP = 0;
int taglen;
unsigned int sequence=0;
unsigned long timestamp=0;
unsigned int initPacketReceive(void);

#define      SIGN_BIT   (0x80)           /* Sign bit for a A-law
byte. */
#define      QUANT_MASK (0xf)       /* Quantization field mask. */
```

```
#define      NSEGS            (8)         /* Number of A-law
segments. */
#define      SEG_SHIFT  (4)         /* Left shift for segment number.
*/
#define      SEG_MASK   (0x70)            /* Segment field mask. */

static short seg_end[8] = {0xFF, 0x1FF, 0x3FF, 0x7FF,
                           0xFFF, 0x1FFF, 0x3FFF, 0x7FFF};
#define BIAS 0x84   /* define the add-in bias for 16 bit samples */
#define CLIP 32635
short clear_au_in = 0;
short pkt_cntr = 0;
short audio_cntr = 0;
unsigned char zed = 0;
unsigned char enable = 1;

void audio_int_handler(void * X){
  short linear_data;
  unsigned char ulaw;
  unsigned char packet_data;
  short sample_to_write;

  if(!(enable||enableSend)){
    return;
  }else{
    microblaze_disable_interrupts();

    //if(enableSend == 1){
             linear_data = XIo_In16(0xFEFE0000);
        ulaw = linear2ulaw((linear_data));

        if(audio_cntr < 159){
               if(ulaw == 0x0){
                  ulaw = 0x2;
               }
               audio_out[audio_cntr++] = ulaw;
        }else{
                 //print("done packet");
                  audio_out[audio_cntr] = 0x7f;
             }
     //XIo_Out32(0xFEFE2222,linear_data);
    //}

    //if(enable == 1){
       if(pkt_cntr < 159){
         packet_data = packet[54 + pkt_cntr++];
       }else{
         enable = 0;
         packet_data = 0x7f;
       }

//       sample_to_write = ulaw2linear(packet_data);
       XIo_Out32(0xFEFE2222,ulaw2linear(packet_data));

    //}
    microblaze_enable_interrupts();
    return;
    }
}


void loadRTP() {
  unsigned int i,j,k,memaddress, packlen, addr, word;
```

```
  outnic(RSAR0, 0); // set DMA starting address
  outnic(RSAR1, TXSTART);

  outnic(ISR, 0xFF); // clear ISR

  outnic(RBCR0, 44); // set Remote DMA Byte Count
  outnic(RBCR1, 0);
  outnic(TBCR0, (44&0xff)); // set Transmit Byte Count
  outnic(TBCR1, (44>>8));

  outnic(CMDR, 0x12); // start the DMA write

  for(memaddress=1065; memaddress<1109; memaddress++) {
    if (memaddress == 1065+36) {
        word = rtpport1 | (rtpport2<<8);
             memaddress++;
    } else {

      addr = 0xC00000 + (memaddress<<1);
      j = XIo_In8(addr);
      memaddress++;
      addr = 0xC00000 + ((memaddress)<<1);
      k = XIo_In8(addr);
      word = (j) | (k<<8);

    }


    outnic(DATAPORT, word);

  }


  if(innic(ISR)&0x40){
    print("RTP Loaded\r\n");
  }
  else{
    print("RTP Load Failed!\r\n"); // . Restart the board\r\n\r\n");
    //return;
  }


}


void sendRTP() {
  unsigned int i,j,k,memaddress, packlen, addr, word;

//  print("sending");

  outnic(RSAR0, 44); // set DMA starting address
  outnic(RSAR1, TXSTART);

  outnic(ISR, 0xFF); // clear ISR



  outnic(RBCR0, 214*2); // set Remote DMA Byte Count
  outnic(RBCR1, 0);
  outnic(TBCR0, (214&0xff)); // set Transmit Byte Count
  outnic(TBCR1, (214>>8));
```

```
  timestamp +=400;
  sequence+=1;


    outnic(DATAPORT, ((sequence>>8) | (sequence<<8)));
    i = timestamp >> 16;
    j = timestamp & 0xffff;

    outnic(DATAPORT, ((i >> 8) | (i << 8)));
    outnic(DATAPORT, ((j >> 8) | (j << 8)));

    //synch source id - random
    outnic(DATAPORT, 0x5823);
    outnic(DATAPORT, 0x4264);

    for (i=0; i<160; i+=2) {
      outnic(DATAPORT, (audio_out[i] | (audio_out[i+1] << 8)));

    }


  if(innic(ISR)&0x40){
  }
  else{
//    print("RTP Failed!\r\n"); // . Restart the board\r\n\r\n");
    //return;
  }

    outnic(TPSR, TXSTART); // set Transmit Page Start Register
    outnic(CMDR, 0x24); // start transmission

    //    if (rtp==1)
    //  audio_cntr=0;

  j=1000;
  while(j-->0 && !(innic(ISR)&0x02));
  if(!j){
    return 1;
  }

  outnic(ISR,0xFF);

    //print("OK\r\n");


  outnic(CR,0x22);

}


void sendSRAMPacket(int start, int end, BYTE puttag, BYTE rtp) {
  int tagpos=-1;
  int length = end-start;
  unsigned int i,j,k,memaddress, packlen, addr, word;
  outnic(CR,0x22);

  if (puttag == 1) {
    length = length + (taglen);
    rtp = 0;
  }

  if (rtp == 1) {
```

```
  length = 214;
  puttag=0;
}



outnic(RSAR0, 0); // set DMA starting address
outnic(RSAR1, TXSTART);

outnic(ISR, 0xFF); // clear ISR



outnic(RBCR0, length*2); // set Remote DMA Byte Count
outnic(RBCR1, 0);
outnic(TBCR0, (length&0xff)); // set Transmit Byte Count
outnic(TBCR1, (length>>8));


outnic(CMDR, 0x12); // start the DMA write
memaddress = start;

if (rtp==1) {
  timestamp +=1000;
  sequence+=1;

  //putnum(sequence);
}


for(i=start; i<start+length; i+=2){


 //if RTP - change the seq # and timestamp
  if (rtp==1 && i >= end) break;

 //if ACK - add To tag
  if (puttag==1 && i>=start+266 && i <= start+266+(taglen)) {
    j = tag[++tagpos];
    k = tag[++tagpos];

    if (tagpos > taglen) {
           //j = k;
           k = 0x0d;
           memaddress++;
    }

    word = j | (k << 8);

  } else if (rtp==1 && (memaddress == start+36)) {
      word = rtpport1 | (rtpport2<<8);
           memaddress +=2;
  } else {

    addr = 0xC00000 + (memaddress<<1);
    j = XIo_In8(addr);
    memaddress++;
    addr = 0xC00000 + ((memaddress)<<1);
    k = XIo_In8(addr);
    word = (j) | (k<<8);
    memaddress++;

    if (puttag==1 && (memaddress == start+18 || memaddress ==
```

```
start+40))
            word=word+((taglen-10)<<8);

    }



    if (puttag==1 && (memaddress == start+26)) {
      print("checksum\r\n");

      if (taglen == 12 || taglen==13) word = 0xb577;
   }

    outnic(DATAPORT, word);

  }


  if (rtp == 1) {
    outnic(DATAPORT, ((sequence>>8) | (sequence<<8)));
    i = timestamp >> 16;
    j = timestamp & 0xffff;

    outnic(DATAPORT, ((i >> 8) | (i << 8)));
    outnic(DATAPORT, ((j >> 8) | (j << 8)));

    //synch source id - random
    outnic(DATAPORT, 0x5823);
    outnic(DATAPORT, 0x4264);

    for (i=0; i<160; i+=2) {
      outnic(DATAPORT, (audio_out[i] | (audio_out[i+1] << 8)));

      //          word = audioout[i] | (audioout[i+1] << 8);
    }

  }

  if(innic(ISR)&0x40){
   // print("OK\r\n");
  }
  else{
    print("Error! bad DMA"); // . Restart the board\r\n\r\n");
    //return;
  }




    outnic(TPSR, TXSTART); // set Transmit Page Start Register
    outnic(CMDR, 0x24); // start transmission

    //    if (rtp==1)
    //  audio_cntr=0;

  j=1000;
  while(j-->0 && !(innic(ISR)&0x02));
  if(!j){
    return 1;
  }

  outnic(ISR,0xFF);
```

```
  outnic(CR,0x22);
}



#define  nextblock_ptr        0x01
static unsigned char nextPage;
static unsigned int currentRetrieveAddress;


int main()
{
  unsigned int i,j;
  unsigned char recvstatus, length;
  short audio_to_swap;

//  BYTE packet[15];

  WORD word;
  int k,addr;

  //-----REGISTER INTERRUPT:
  microblaze_register_handler(audio_int_handler, (void *)0);


  //microblaze_enable_icache();

//  print("\r\n\r\n***** Ethernet Diagnostic *****\r\n\r\n");
//  diag();
//  print("\r\n\r\n***** Ethernet Configuration *****\r\n\r\n");

  // reset the NIC card
  if(!init_etherne()){
    print("Ethernet NIC not detected");
//        return;
  }
//  print("Done!\r\n");



  print("Writing headers to memory...");


//for (i=0; i<3; i++) {


// ------GRATUITOUS ARP REQUEST ------
//sendSRAMPacket(596, 596+60, 0, 0);

//}



// --------------SIP INVITE -----------------
sendSRAMPacket(0, 596, 0, 0);



// RECEIVE
```

```
  microblaze_enable_interrupts();
  while (1) {

    recvstatus=0;

      if (enableSend == 1 && audio_cntr >= 159 ) {
             audio_cntr=0;
        sendRTP();
      }


    while((recvstatus&0x01)!=1 && (recvstatus&0x04)!=0x04) {

      recvstatus =  innic(ISR);

      if (enableSend == 1 && audio_cntr >= 159) {
             audio_cntr=0;
        sendRTP();
      }
    }


    length = initPacketReceive();
    if (isRTP == 1) {
      pkt_cntr = 0;
      enable = 1;
      isRTP=0;
    }

//    if (length == 5) break;

    outnic(ISR, 0x40);
    outnic(CR, (RD2|START));
    recvstatus=0;

  }

  return 0;

}




// diagnostic that tests nic functionality
void diag(){
  int i;
/*
  print("         Command Register Page Switching Test\r\n");
  print("Switching to page 1\r\n");
  outnic(CMDR, 0x61);

  print("Writing to and reading from 0x0D (value should be 0x4e):
");
  outnic(0x0D*2,0x4e);
  putnum(innic(0x0D*2));
  print("\r\n");

  print("Switching to page 0\r\n");
  outnic(CMDR, 0x21);

  print("Reading from reg offset 0x0D: ");
```

```
  putnum(innic(0x0D*2));
  print("\r\n");

  print("Switching to page 1\r\n");
  outnic(CMDR, 0x61);
  print("Reading from reg offset 0x0D (should be 0x4e): ");
  putnum(innic(0x0D*2));
  print("\r\n\r\n");

  print("         Default Value Test\r\n");
  print("Switching to page 0\r\n");
  outnic(CMDR,0x21);
  print("Reading from 0x16 (value should be 0x15): ");
  putnum(innic(0x16*2));
  print("\r\n");
  print("Reading from 0x12 (value should be 0x0c): ");
  putnum(innic(0x12*2));
  print("\r\n");
  print("Reading from 0x13 (value should be 0x12): ");
  putnum(innic(0x13*2));
  print("\r\n");
*/
}

int init_etherne()
{
  outnic(NE_RESET, innic(NE_RESET));  // Do reset
  print("Reset sent...\n\r");
  //delay(2);
  if ((innic(ISR) & 0x80) == 0)       // Report if failed
  {
//    print("Ethernet card failed to reset! Read (should be 0x80)
\r\n");
//   putnum(innic(ISR));
    print("failed\r\n");
    return 0;
  }
  else
  {

//    print("Ethernet card reset successful!\r\n");
    resetnic(); // Reset Ethernet card,
//    print("Ethernet card intialization complete!\r\n");
//    diag();
  }

  return 1;
}







void resetnic(){
  int i, count;

  outnic(CMDR, RD2|STOP);  // Abort and DMA and stop the NIC
  delay(10);

  outnic(DCR, DCRVAL);   // Set word-wide access
```

```
  outnic(RBCR0, 0x00); // Clear the count regs
  outnic(RBCR1, 0x00);

  outnic(IMR, 0x00);   // Mask completion irq
  outnic(ISR, 0xFF);   // clear interrupt status register

  outnic(RCR, 0x20);   // 0x20  Set to monitor mode
//  outnic(RCR, 0x6f);

  //outnic(TCR, 0x02);  // put nic in normal operation

  // Set Rx start, Rx stop, Boundary and TX start regs
  outnic(BNRY, RXSTART);
  outnic(PSTART, RXSTART);
  outnic(PSTOP, RXSTOP);
  nextPage = RXSTART+1;

  outnic(CMDR, (PS0|RD2|STOP));
  //write mac
  outnic(PAR0+0, 0x00);
  outnic(PAR0+1, 0x08);
  outnic(PAR0+2, 0x02);
  outnic(PAR0+3, 0x62);
  outnic(PAR0+4, 0x30);
  outnic(PAR0+5, 0x5d); //laptop = 5d

  outnic(CURR, RXSTART+1);


  outnic(CMDR, (RD2|START));
  outnic(RCR, 0x40+0x10); //+8+4+2+1);

  //full duplex
  outnic(TCR, 0x80);

  outnic(TPSR, TXSTART);


  outnic(CMDR, (RD2|STOP));

  outnic(DCR, DCRVAL);   // Set word-wide access

  outnic(CMDR, (RD2|START));


  outnic(ISR, 0xFF);   // clear interrupt status register
  outnic(IMR, 0x01);   // Mask completion irq

  outnic(TCR, 0x80);


  // put nic in start mode
  outnic(CMDR, 0x22); // start nic
  outnic(TCR, 0x80);  // put nic in normal operation


}



unsigned int initPacketReceive(void)
{
```

```
//           unsigned char writePagePtr;
//           unsigned char readPagePtr;
//           unsigned char bnryPagePtr;
             unsigned char i, j, tagstart, tagend;
             unsigned int k;
             int length;

             unsigned char pageheader[4];
             //   BYTE packet[260]; //was 214 for testing

             unsigned int rxlen, word;


             // clear the packet received interrupt flag
             outnic(ISR, 0x01);

             // if the boundary pointer is invalid,
             // reset the contents of the buffer and exit


             if( (nextPage < RXSTART) || (nextPage >= RXSTOP) )
             {
                  //print("nextpage");
                  outnic(BNRY, RXSTART);
                  outnic(CR, (PS0|RD2|START));
                  outnic(CURR, RXSTART+1);
                  outnic(CR, (RD2|START));
//                nextPage = RXSTART+1;
//                return 0;
             }



             // initiate DMA to transfer the RTL8019 packet header
             outnic(RBCR0, 4);
             outnic(RBCR1, 0);
             outnic(RSAR0, 0);
             outnic(RSAR1, nextPage);   //readPagePtr);
             outnic(CR, (RD0|START));

             //print("\r\nPacket header:\r\n");

             for(i=0;i<4;i++) {
                  j = innic(DATAPORT);
//                pageheader[i+1] = j;
                  pageheader[i] = (unsigned char)j;

             }


             // end the DMA operation

             outnic(CR, (RD2|START));
             for(i = 0; i <= 20; i++)
             if(innic(ISR) & 0x40)
                  break;

             outnic(ISR, 0x40);


             //using wrong pageheader reader, length is in
pageheader[1]
             //length = pageheader[1]-4;
```

```
            //trying w/ correct - 2?
            length = pageheader[1]-4;


            if (length > 255) length = 255;
            currentRetrieveAddress = (nextPage<<8) + 2;


            outnic(ISR, 0xFF); // clear ISR


            outnic(RBCR0, (unsigned char)length);
            outnic(RBCR1, (unsigned char)(length>>8));
            outnic(RSAR0, (unsigned char)currentRetrieveAddress);
            outnic(RSAR1, (unsigned char)(currentRetrieveAddress >>
8));
            outnic(CR, (RD0|START));

            for (i=0; i<length; i+=2) {
                 k = innic(DATAPORT);
                 packet[i] = (k); //lower 8
                 packet[i+1] = (k>>8);//upper 8
                 //switch i, i+1 for easy retransmission
            }

            outnic(CR, (RD2|START));
            for (i=0; i<=20; i++) {
                 if (innic(ISR) & 0x40)
                       break;
            }
            outnic(ISR, 0x40);




            length = pageheader[1]-4;



            //nextPage++ works, but not for longer/linked packets,
me thinks
            //packets that extend beyond a single buffer page need
            //to obtain nextPage off the pageheader[]

            nextPage++;

            if(nextPage-1 < RXSTART) nextPage = RXSTOP;


            outnic(BNRY, nextPage-1);


//          nextPage++;
//          i = nextPage;  //store temporarily b/c we try find
linked buffers for a SIP 183
//          nextPage = pageheader[3];
//          nextPage++;
//          outnic(BNRY, nextPage-1);
```

```
            //CHECK IF SIP OR RTP PACKET

            //RANDOM SIP NOTES:
            /* we don't need to get the RTP port out of the 183, we
can just receive an
               RTP packet, which is small, and find the source
port; otherwise we'll have
               to read extra linked buffer pages on the 183 ...
screw that! */

            //DEBUGGING:
            /*
            print("page: ");
            putnum(nextPage);
            print("\r\n");
            for (i=42; i<48; i++) {
            putnum(packet[i]); //1 = 56
            print("\r\n");
            }
            return 0;
            */



            //SIP

            if (packet[44] == 0x53) {

                  j = (packet[52]-0x30)*100 + (packet[53]-0x30)*10 +
(packet[54]-0x30);



                  if (j==100) {
                        print("SIP 100\r\n");
//                      nextPage++;
                        return 4;
     /*           } else if (j==200) {
                        print("SIP 200\r\n");



                        //if we missed the 183 packet, let's get the
tag now


                        return 4;
     */
                  } else if (j==200) { //|| j==183) { // || j==183)
{
                        print("SIP 183/200\r\n");

                        //return 4;


                        //the 183 packet is really long (includes
SDP)...
                        //going to have to find linked packets
                        //the "TO" "TAG" should be in the 2nd buffer
                        //and we need this tag from here b/c we need
to send an ACK
                        //response including the tag
```

```
                        currentRetrieveAddress+=length;

//                      print("\r\nlength: ");
//                      putnum(length);

                        outnic(ISR, 0xFF); // clear ISR

                        outnic(RBCR0, (unsigned char)250);
                        outnic(RBCR1, (unsigned char)(250 >> 8));
                        outnic(RSAR0, (unsigned char)
currentRetrieveAddress);
                        outnic(RSAR1, (unsigned char)
(currentRetrieveAddress >> 8));
                        outnic(CR, (RD0|START));

                        //overwrite old packet to preserve space
                        for (i=0; i<250; i+=2) {
                              k = innic(DATAPORT);
//                            putnum(k);
//                            print("\r\n");
                              packet[i] = (k); //lower 8
                              packet[i+1] = (k>>8);//upper 8
                        }

                        outnic(CR, (RD2|START));

                        for (i=0; i<=20; i++) {
                              if (innic(ISR) & 0x40)
                                    break;
                        }

                        outnic(ISR, 0x40);


                        print("searching...\r\n");


                        i=0;
                        for (j=0; j<250; j++) {
                              //putnum(j);
                              //print(" ");
                              //putnum(packet[j]);
                              //print("\r\n");
                              //traverse looking for "To:"
                              if (i!=1 && packet[j] == 0x0a &&
packet[j+1] == 0x54 && (packet[j+2] == 0x6e || packet[j+2]==0x6f)) {
                                    //"to" found at position a in
packet
                                    i = 1;
                                    print("TO\r\n");

                              } else if (i==1 && packet[j]==0x74 &&
packet[j+1]==0x61 && (packet[j+2] == 0x66 || packet[j+2]==0x67)) {
                                    print("TAG\r\n");

                                    j+=4 ;
                                    tagstart = j;

                                    i=0;
                                    while (packet[j] != 0x0d && i<15)
{  //dangerous?
                                          putnum(packet[j]);
                                          print("\r\n");
```

```
                                        tag[i] = packet[j];
                                        j++;
                                        i++;
                                    }
                                    i=-1;


                                    j-1;
                                    taglen = j-1-tagstart;

                                    putnum(tagend - tagstart);

                                    // -----------SIP ACK
------------
                                    sendSRAMPacket(596+60,
409+596+60, 1, 0);

                                    loadRTP();


                                    enableSend = 1;


                                    //break;
                               }

                          }



                          return 4;


                         }

                         return 4;

                    //RTP:
                    } else if (rtpport1 == 0 && packet[44] == 0x80) {
                      //print("RTP!\r\n");
                      isRTP = 1;

                        //first RTP, parse out port # in bytes 36, 37
                        //packet is offset by 2
                        rtpport1 = packet[36];
                        rtpport2 = packet[37];
                        enableSend = 1;

                        print("Port: ");
                        putnum(rtpport1 | (rtpport2 << 8));
                        print("\r\n");


                        loadRTP();


                        return 4;
                    } else if (rtpport1 != 0 && packet[36] == rtpport1 &&
packet[37]==rtpport2) {
                      isRTP = 1;
                      return 4;
                    }
```

```
            //END CHECK


            return 0;

}




void delay(int mult){
  int i;
  int delay = 1000000*mult;
  for(i=0; i<delay; i++);
}


//#define    BIAS       (0x84)           /* Bias for linear code. */

static int
search(
            int        val,
            short            *table,
            int        size)
{
            int        i;

            for (i = 0; i < size; i++) {
                 if (val <= *table++)
                       return (i);
            }
            return (size);
}

/*
 * linear2ulaw() - Convert a linear PCM value to u-law
 *
 * In order to simplify the encoding process, the original linear
magnitude
 * is biased by adding 33 which shifts the encoding range from (0 -
8158) to
 * (33 - 8191). The result can be seen in the following encoding
table:
 *
 *           Biased Linear Input Code     Compressed Code
 *           ------------------------     ---------------
 *           00000001wxyza                000wxyz
 *           0000001wxyzab                001wxyz
 *           000001wxyzabc                010wxyz
 *           00001wxyzabcd                011wxyz
 *           0001wxyzabcde                100wxyz
 *           001wxyzabcdef                101wxyz
 *           01wxyzabcdefg                110wxyz
 *           1wxyzabcdefgh                111wxyz
 *
 * Each biased linear code has a leading 1 which identifies the
segment
```

```
 * number. The value of the segment number is equal to 7 minus the
number
 * of leading 0's. The quantization interval is directly available
as the
 * four bits wxyz.  * The trailing bits (a - h) are ignored.
 *
 * Ordinarily the complement of the resulting code word is used for
 * transmission, and so the code word is complemented before it is
returned.
 *
 * For further information see John C. Bellamy's Digital Telephony,
1982,
 * John Wiley & Sons, pps 98-111 and 472-476.
 */
unsigned char
linear2ulaw(
            int        pcm_val)    /* 2's complement (16-bit range)
*/
{
            int        mask;
            int        seg;
            unsigned char    uval;

            /* Get the sign and the magnitude of the value. */
            if (pcm_val < 0) {
                 pcm_val = BIAS - pcm_val;
                 mask = 0x7F;
            } else {
                 pcm_val += BIAS;
                 mask = 0xFF;
            }

            /* Convert the scaled magnitude to segment number. */
            seg = search(pcm_val, seg_end, 8);

            /*
             * Combine the sign, segment, quantization bits;
             * and complement the code word.
             */
            if (seg >= 8)          /* out of range, return maximum
value. */
                 return (0x7F ^ mask);
            else {
                 uval = (seg << 4) | ((pcm_val >> (seg + 3)) &
0xF);
                 return (uval ^ mask);
            }

}

/*
 * ulaw2linear() - Convert a u-law value to 16-bit linear PCM
 *
 * First, a biased linear code is derived from the code word. An
unbiased
 * output can then be obtained by subtracting 33 from the biased
code.
 *
 * Note that this function expects to be passed the complement of
the
 * original code word. This is in keeping with ISDN conventions.
 */
int
```

```
ulaw2linear(
            unsigned char    u_val)
{
            int        t;

            /* Complement to obtain normal u-law value. */
            u_val = ~u_val;

            /*
             * Extract and bias the quantization bits. Then
             * shift up by the segment number and subtract out the
bias.
             */
            t = ((u_val & QUANT_MASK) << 3) + BIAS;
            t <<= ((unsigned)u_val & SEG_MASK) >> SEG_SHIFT;

            return ((u_val & SIGN_BIT) ? (BIAS - t) : (t - BIAS));
}
```

---

opb_ak4565_v2_1_0.pao

```
lib opb_ak4565_v1_00_a opb_ak4565
```

---

opb_ak4565_v2_1_0.mpd

```
##############################################################################
##
## opb_ak4565_v2_1_0.mpd
##
## Microprocessor Peripheral Definition
##
##############################################################################

BEGIN opb_ak4565, IPTYPE = PERIPHERAL
OPTION IMP_NETLIST = TRUE
OPTION HDL = VHDL

# Define bus interface
BUS_INTERFACE BUS=SOPB, BUS_STD=OPB, BUS_TYPE=SLAVE

# Generics for vhdl or parameters for verilog
PARAMETER C_OPB_AWIDTH = 32, DT=integer
PARAMETER C_OPB_DWIDTH = 32, DT=integer
PARAMETER C_BASEADDR = 0xFFFFFFFF, DT=std_logic_vector,
MIN_SIZE=0x100, BUS=SOPB
PARAMETER C_HIGHADDR = 0x00000000, DT=std_logic_vector, BUS=SOPB


# Signals
PORT OPB_Clk = "", DIR=IN, BUS=SOPB, SIGIS=CLK
PORT OPB_Rst = OPB_Rst, DIR=IN, BUS=SOPB

# OPB slave signals
PORT OPB_ABus = OPB_ABus, DIR=IN, VEC=[0:C_OPB_AWIDTH-1], BUS=SOPB
PORT OPB_BE = OPB_BE, DIR=IN, VEC=[0:C_OPB_DWIDTH/8-1], BUS=SOPB
PORT OPB_DBus = OPB_DBus, DIR=IN, VEC=[0:C_OPB_DWIDTH-1], BUS=SOPB
PORT OPB_RNW = OPB_RNW, DIR=IN, BUS=SOPB
PORT OPB_select = OPB_select, DIR=IN, BUS=SOPB
PORT OPB_seqAddr = OPB_seqAddr, DIR=IN, BUS=SOPB
```

```
PORT Sln_DBus = Sl_DBus, DIR=OUT, VEC=[0:C_OPB_DWIDTH-1], BUS=SOPB
PORT Sln_errAck = Sl_errAck, DIR=OUT, BUS=SOPB
PORT Sln_retry = Sl_retry, DIR=OUT, BUS=SOPB
PORT Sln_toutSup = Sl_toutSup, DIR=OUT, BUS=SOPB
PORT Sln_xferAck = Sl_xferAck, DIR=OUT, BUS=SOPB

#PORT PB_D = "", DIR=INOUT, VEC=[15:0], 3STATE=FALSE, IOB_STATE=BUF
PORT AU_CSN_N = "", DIR=OUT
PORT AU_BCLK = "", DIR=OUT
PORT AU_MCLK = "", DIR=OUT
PORT AU_LRCK = "", DIR=OUT
PORT AU_SDTI = "", DIR=OUT
PORT AU_SDTO0 = "", DIR=IN
PORT Interrupt = "", SIGIS=INTERRUPT

END
```

---

opb_ak4565.vhd

```
------------------------------------------------------------------------
-- ak4565 audio codec, written by colin gilboy. (clock generation
section taken
-- from previous year).
--
------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity opb_ak4565 is

  generic(
    C_OPB_AWIDTH    : integer := 32;
    C_OPB_DWIDTH    : integer := 32;
    C_BASEADDR      : std_logic_vector(31 downto 0) := X"FEFE_0000";
    C_HIGHADDR      : std_logic_vector(31 downto 0) := X"FEFE_FFFF"
    );

  port(
            --ins
    OPB_Clk : in std_logic;
    OPB_Rst : in std_logic;
    OPB_ABus : in std_logic_vector (C_OPB_AWIDTH-1 downto 0);
    OPB_BE : in std_logic_vector (3 downto 0);
    OPB_DBus : in std_logic_vector (C_OPB_DWIDTH-1 downto 0);
    OPB_RNW : in std_logic;
    OPB_select : in std_logic;
    OPB_seqAddr : in std_logic;

    AU_SDTO0 : in std_logic;

    --outs
    AU_CSN_N : out std_logic;
    AU_BCLK : out std_logic;
    AU_MCLK : out std_logic;
    AU_LRCK : out std_logic;
    AU_SDTI : out std_logic;


    Sln_DBus : out std_logic_vector (C_OPB_DWIDTH-1 downto 0);
```

```
    Sln_errAck : out std_logic;
    Sln_retry : out std_logic;
    Sln_toutSup : out std_logic;
    Sln_xferAck : out std_logic;

    Interrupt : out std_logic

    );
end opb_ak4565;

architecture Behavioral of opb_ak4565 is

  signal RNW : std_logic;
  signal ABus : std_logic_vector(C_OPB_AWIDTH-1 downto 0);
  signal DBus : std_logic_vector(C_OPB_DWIDTH-1 downto 0);



  signal data_out,data_in : std_logic_vector(15 downto 0);  --
offchip data signals
  signal cs_data, cs_config, cs, readctrlregs : std_logic:= '0';  --
chip selects
  signal config_clk, config_in : std_logic;  -- config signals
  signal audio_clk,lrck : std_logic;         -- internal clock sigs
  signal clkcnt : std_logic_vector(4 downto 0);  -- divider counter
  signal bit_cnt : std_logic_vector(4 downto 0);-- counter for which
bit
                                                --(0-32) that we're
writing now
  signal adc_reg: std_logic_vector(31 downto 0);
  signal adc_reg_lowerhalf : std_logic_vector(15 downto 0);
  signal adc_reg_lowerhalf_rev : std_logic_vector(15 downto 0);
  signal config_reg: std_logic_vector(15 downto 0);
  signal config_cnt : std_logic_vector(4 downto 0):= "00000";  --
config counter
  signal parallel_out : std_logic_vector(15 downto 0); -- to be sent
to microblaze
  signal rdata : std_logic_vector(31 downto 0):=X"0000_0000";  --
data to send to slndbus
  signal tristate : std_logic_vector(15 downto 0):=X"0000";  --
tristate driver ctrl vector
  signal cfgdatabit : std_logic;
  signal poll : std_logic;
  signal state : std_logic_vector(3 downto 0);      -- state
variable
  signal xferAck : std_logic;
  signal sampleToWrite : std_logic_vector(15 downto 0);  -- for
audio output
  signal sampleToWrite_rev : std_logic_vector(15 downto 0);  -- for
audio output
  signal serialToWrite : std_logic;     -- serial stream for audio
output
  signal sample_cnt : std_logic_vector(2 downto 0);
  signal itrpt : std_logic := '0';
begin  -- Behavioral

  Interrupt <= itrpt;
  Sln_errAck <='0';
  Sln_retry <= '0';
  Sln_toutSup <= '0';

  cs <= OPB_select when OPB_ABus(31 downto 16) = X"FEFE" else '0';
```

```
  AU_CSN_N <= '1';

--clock dividers to create proper clocks for the codec:
process(OPB_Clk, OPB_Rst)
begin
            if OPB_Rst = '1' then
                 clkcnt <= "00000";
            elsif OPB_clk'event and OPB_clk='1' then
                 clkcnt <= clkcnt + 1;
            end if;
end process;
AU_MCLK <= clkcnt(1); -- master clock, 12.5 MHz
audio_clk <= clkcnt(4); -- this is the serial clock, 1.5625 Mhz
--AU_BCLK <= audio_clk;
AU_BCLK <= not audio_clk; -- BCLK is antiphase (as to sample in the
middle
                         -- of data)

-- process(OPB_Clk, OPB_Rst, clkcnt, sample_cnt)
--   begin
--     if OPB_Rst = '1' then
--       itrpt <= '0';
--     elsif(OPB_Clk'event and OPB_Clk = '1') then
--       if(clkcnt = "0100010000" and sample_cnt = "101") then
--         itrpt <= '1';
--         sample_cnt <= "000";
--       elsif (clkcnt = "0100000000") then
--         sample_cnt <= sample_cnt + 1;
--       else
--         itrpt <= '0';
--       end if;
--     end if;
--   end process;

process(audio_clk, OPB_Rst)
begin

  if OPB_rst = '1' then
    bit_cnt <= "00000";
    lrck <= '0';
  elsif audio_clk'event and audio_clk='1' then
    bit_cnt <= bit_cnt + 1;
    lrck <= not bit_cnt(4);
    if(bit_cnt = "10001") then
      itrpt <= '1';
      sample_cnt <= "000";
    elsif(bit_cnt = "10000") then
      sample_cnt <= sample_cnt + 1;
    else
      itrpt <= '0';
    end if;
  end if;
end process;

AU_LRCK <= lrck; --channel selection clock, 48.828 kHz


--fill register with the serial bits of data, send the left channel
(bottom 16
--bits to the microblaze via parallel_out signal
process(audio_clk,bit_cnt,lrck)
  begin
  if audio_clk'event and audio_clk = '0' then
```

```
      adc_reg(conv_integer(bit_cnt - 1)) <= AU_SDTO0;
  end if;
  if audio_clk'event and audio_clk = '1' then
    if( (bit_cnt) < "10000" ) then
      serialToWrite <= sampleToWrite_rev(conv_integer(bit_cnt));
    else
      serialToWrite <= '0';
    end if;
  end if;
end process;
AU_SDTI <= serialToWrite;

process(audio_clk)
  begin
    if audio_clk'event and audio_clk = '0' and bit_cnt = "10000"
then
      adc_reg_lowerhalf <= adc_reg(15 downto 0);
    end if;

end process;

process (adc_reg_lowerhalf,sampleToWrite) begin
        for i in 0 to 15 loop
                  adc_reg_lowerhalf_rev(i)<=adc_reg_lowerhalf(15-i);
             end loop;
        for j in 0 to 15 loop
                  sampleToWrite_rev(j)<=sampleToWrite(15-j);
             end loop;
end process;

--rdata(31 downto 16) <= X"0000";
Sln_DBus <= rdata when cs = '1' and OPB_RNW = '1' else X"0000_0000";

--FSMish thing:
--states: selected  (0001)
--        ready poll(0010)
--        ready data(0100)
--        xfer      (1000)
--        write     (1111)
--        idle      (0000)
poll_and_read : process(OPB_Clk,OPB_Rst, state, poll, OPB_RNW,
cs,OPB_ABus, bit_cnt, clkcnt)
  begin
    if(OPB_Rst = '1') then
      xferAck <= '0';
      rdata(15 downto 0) <= X"0000";
      state <= "0000";
    elsif(OPB_Clk'event and OPB_Clk = '1') then
      if (cs = '1' ) then
        case state is

          when "0000" =>
            xferAck <= '0';
            rdata <= X"0000_0000";
            if(OPB_RNW = '1') then
              state <= "0001";   --selected + reading
            elsif OPB_ABus(15 downto 0) = X"2222" then
              state <= "1111";   --writing
            else
              state <= "0000";
            end if;

          when  "0001" =>
```

```
            if(OPB_ABus(15 downto 0) = X"0000") then
              state <= "0100";
            else
              state <= "0000";
            end if;
          when "0100" =>
            xferAck <= '0';
            --rdata(25 downto 16) <= clkcnt(9 downto 0);--
adc_reg_lowerhalf_rev;
            rdata(15 downto 0) <= adc_reg_lowerhalf_rev;
            state <= "1000"; --go to xfer

          when "1000" =>
            xferAck <= '1';
            state <= "0000";

          when "1111" =>
            sampleToWrite <= OPB_DBus(15 downto 0);
            state <= "1000";

          when others =>
           null;
        end case;
      else
        xferAck <='0';
        state <= "0000";

      end if; --end if cs=1
    end if; --end if OPB_RST
end process poll_and_read;
Sln_xferAck <= xferAck;

end Behavioral;
```

---

opb_cntrl_v2_1_0.pao

```
##############################################################################
#
# opb controller pao file
#
##############################################################################

lib opb_cntrl_v1_00_a opb_cntrl
#lib opb_cntrl_v1_00_a opb_eth
#lib opb_cntrl_v1_00_a opb_sram
lib opb_cntrl_v1_00_a pad_io
lib opb_cntrl_v1_00_a memoryctrl
```

---

opb_cntrl_v2_1_0.mpd

```
##############################################################################
##
## Microprocessor Peripheral Definition
##
##############################################################################

BEGIN opb_cntrl, IPTYPE = PERIPHERAL, EDIF=TRUE

BUS_INTERFACE BUS = SOPB, BUS_STD = OPB, BUS_TYPE = SLAVE
```

```
## Generics for VHDL
PARAMETER c_baseaddr     = 0x00100000, DT = std_logic_vector,
MIN_SIZE = 0xFF
PARAMETER c_highaddr     = 0x00000000, DT = std_logic_vector
PARAMETER c_opb_awidth   = 32,         DT = integer
PARAMETER c_opb_dwidth   = 32,         DT = integer

# Signals
PORT OPB_Clk = "", DIR=IN, BUS=SOPB, SIGIS=CLK
PORT OPB_Rst = OPB_Rst, DIR=IN, BUS=SOPB

# OPB slave signals
PORT OPB_ABus = OPB_ABus, DIR=IN, VEC=[0:C_OPB_AWIDTH-1], BUS=SOPB
PORT OPB_BE = OPB_BE, DIR=IN, VEC=[0:C_OPB_DWIDTH/8-1], BUS=SOPB
PORT OPB_DBus = OPB_DBus, DIR=IN, VEC=[0:C_OPB_DWIDTH-1], BUS=SOPB
PORT OPB_RNW = OPB_RNW, DIR=IN, BUS=SOPB
PORT OPB_select = OPB_select, DIR=IN, BUS=SOPB
PORT OPB_seqAddr = OPB_seqAddr, DIR=IN, BUS=SOPB
PORT UIO_DBus = Sl_DBus, DIR=OUT, VEC=[0:C_OPB_DWIDTH-1], BUS=SOPB
PORT UIO_errAck = Sl_errAck, DIR=OUT, BUS=SOPB
PORT UIO_retry = Sl_retry, DIR=OUT, BUS=SOPB
PORT UIO_toutSup = Sl_toutSup, DIR=OUT, BUS=SOPB
PORT UIO_xferAck = Sl_xferAck, DIR=OUT, BUS=SOPB


PORT PB_A = "", DIR=OUT, VEC=[19:0], IOB_STATE=BUF
PORT PB_LB_N = "", DIR=OUT, IOB_STATE=BUF
PORT PB_UB_N = "", DIR=OUT, IOB_STATE=BUF
PORT PB_D = "", DIR=INOUT, VEC=[15:0], 3STATE=FALSE, IOB_STATE=BUF
PORT PB_WE_N = "", DIR = OUT, IOB_STATE=BUF
PORT PB_OE_N = "", DIR = OUT, IOB_STATE=BUF
PORT RAM_CE_N = "", RAM_CE_N, DIR = OUT, IOB_STATE=BUF

PORT ETHERNET_CS_N = "", ETHERNET_CS_N, DIR = OUT, IOB_STATE=BUF
PORT ETHERNET_RDY = "", DIR = IN, IOB_STATE=BUF
PORT ETHERNET_IREQ = "", DIR = IN, IOB_STATE=BUF
PORT ETHERNET_IOCS16_N = "", DIR = IN, IOB_STATE=BUF

PORT io_clock = "", DIR=IN

END
```

---

pad_io.vhd

```
----------------------------------------------------------------------
-----------
-- CSEE 4840 Embedded System Design
-- Jaycam
-- Yaniv Schiller - yhs2001@columbia.edu
-- Avrum Tilman - amt77@columbia.edu
-- Josh Weinberg - jmw211@columbia.edu
----------------------------------------------------------------------
-----------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
```

```
entity pad_io is
    Port ( sys_clk : in std_logic;
           io_clock : in std_logic;
           read_early : in std_logic;
           write_early : in std_logic;

           rst : in std_logic;
           PB_A : out std_logic_vector(19 downto 0);
           PB_UB_N : out std_logic;
           PB_LB_N : out std_logic;
           PB_WE_N : out std_logic;
           PB_OE_N : out std_logic;
           RAM_CE_N : out std_logic;
           ETHERNET_CS_N : out std_logic;
           ETHERNET_RDY : in std_logic;
           ETHERNET_IREQ : in std_logic;
           ETHERNET_IOCS16_N : in std_logic;
           PB_D : inout std_logic_vector(15 downto 0);

           pb_addr : in std_logic_vector(19 downto 0);
           pb_ub : in std_logic;
           pb_lb : in std_logic;
           pb_wr : in std_logic;
           pb_rd : in std_logic;
           ram_ce : in std_logic;
           ethernet_ce : in std_logic;
           pb_dread : out std_logic_vector(15 downto 0);
           pb_dwrite : in std_logic_vector(15 downto 0));
end pad_io;

architecture Behavioral of pad_io is

component FDCE
port (C : in std_logic;
                 CLR : in std_logic;
                 CE : in std_logic;
                 D : in std_logic;
                 Q : out std_logic);
end component;

component FDPE
port (C : in std_logic;
                 PRE : in std_logic;
                 CE : in std_logic;
                 D : in std_logic;
                 Q : out std_logic);
end component;

attribute iob : string;
attribute iob of FDCE : component is "true";
attribute iob of FDPE : component is "true";

component OBUF_F_24
port (O : out STD_ULOGIC;
                 I : in STD_ULOGIC);
end component;

component IOBUF_F_24
port (O : out STD_ULOGIC;
                 IO : inout STD_ULOGIC;
                 I : in STD_ULOGIC;
                 T : in STD_ULOGIC);
end component;
```

```
signal io_half : std_logic;
signal pb_addr_1: std_logic_vector(19 downto 0);
signal pb_dwrite_1: std_logic_vector(15 downto 0);
signal pb_tristate: std_logic_vector(15 downto 0);
signal pb_tristate_1: std_logic_vector(15 downto 0);
signal pb_dread_a: std_logic_vector(15 downto 0);
signal we_n, pb_we_n1: std_logic;
signal oe_n, pb_oe_n1: std_logic;
signal lb_n, pb_lb_n1: std_logic;
signal ub_n, pb_ub_n1: std_logic;
signal ethce_n, eth_ce_n1 : std_logic;
signal ramce_n, ram_ce_n1: std_logic;
signal dataz : std_logic;

signal rd_ce, wr_ce, din_ce, rd_early, wr_early : std_logic;

--attribute equivalent_register_removal: string;
--attribute equivalent_register_removal of pb_tristate_1 : signal is
"no";
--attribute equivalent_register_removal of pb_dwrite_1 : signal is
"no";


begin

------------------------ !!!!!!!!!!!!!!!!!!!!
-----------------------------
--process (io_clock)
--begin
--  if io_clock'event and io_clock = '1' then
--    io_half <= sys_clk;
--  end if;
--end process;
io_half <= not sys_clk;
-------------------------------------------------------------------------
-----------

process(rst, sys_clk)
begin
            if rst='1' then
          rd_early <= '0';
          wr_early <= '0';
        elsif sys_clk'event and sys_clk='1' then
          rd_early <= read_early;
          wr_early <= write_early;
      end if;
end process;

dataz <= (not pb_wr) or pb_rd;
pb_tristate_1 <= "1111111111111111" when dataz ='1' else
"0000000000000000";



-- control
pb_we_n1 <= not pb_wr; -- or (not io_half and wr_early);

pb_oe_n1 <= not pb_rd; --or (not io_half and rd_early);

pb_lb_n1 <= '0' when (ram_ce='1') else (not pb_lb);

pb_ub_n1 <= '0' when (ram_ce='1') else (not pb_ub);
```

```
ram_ce_n1 <= not ram_ce;

eth_ce_n1 <= not ethernet_ce;


-- I/O BUFFERS

webuf : OBUF_F_24
port map (O => PB_WE_N,
                         I => pb_we_n1);

oebuf : OBUF_F_24
port map (O => PB_OE_N,
                         I => pb_oe_n1);

ramcebuf : OBUF_F_24
port map (O => RAM_CE_N,
                         I => ram_ce_n1);

ethcebuf : OBUF_F_24
port map (O => ETHERNET_CS_N,
                         I => eth_ce_n1);

--  ETHERNET_RDY : in std_logic;
--  ETHERNET_IREQ : in std_logic;
--  ETHERNET_IOCS16_N : in std_logic;




ubbuf : OBUF_F_24
port map (O => PB_UB_N,
                         I => pb_ub_n1);

lbbuf : OBUF_F_24
port map (O => PB_LB_N,
                         I => pb_lb_n1);

abuf : for i in 0 to 19 generate
             abuf : OBUF_F_24 port map (
                  O => PB_A(i),
                  I => pb_addr(i));
end generate;

dbuf : for i in 0 to 15 generate
             dbuf : IOBUF_F_24 port map (
                  O => pb_dread(i),
                  IO => PB_D(i),
                  I => pb_dwrite(i),
                  T => pb_tristate_1(i));
end generate;



end Behavioral;
```

---

memoryctrl.vhd

```
----------------------------------------------------------------------
-----------
-- CSEE 4840 Embedded System Design
```

```
-- Jaycam
-- Yaniv Schiller - yhs2001@columbia.edu
-- Avrum Tilman - amt77@columbia.edu
-- Josh Weinberg - jmw211@columbia.edu
----------------------------------------------------------------------
-----------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity memoryctrl is
    Port ( rst : in std_logic;
           clk : in std_logic;
           cs : in std_logic; -- any of my devices selected
           opb_select : in std_logic; -- original select

           rnw : in std_logic;

           eth_io : in std_logic;
           fullword : in std_logic;
                read_early : out std_logic;
                write_early : out std_logic;
           bus_req : out std_logic;

           videocycle : out std_logic;
           hihalf : out std_logic;
           wr_req : out std_logic;
           rd_req : out std_logic;
           xfer : out std_logic;
           ce0 : out std_logic;
           ce1 : out std_logic;
           rres : out std_logic;

                ram_output_enable : out std_logic;
                ram_WE : out std_logic;
                sram_ce : in std_logic
                          );
end memoryctrl;

architecture Behavioral of memoryctrl is

signal r_idle, r_common, r_w32, r_ra, r_rb, r_rc, r_rd, r_xfer :
std_logic;
signal r_weth1, r_weth2, r_weth3 : std_logic;
signal r_v1, r_v0, r_v2 : std_logic;
signal wr_req_i, rd_req_i, videocycle_i : std_logic;
signal sram_rst : std_logic;

  signal WE : std_logic;
  signal output_enable : std_logic;

  constant STATE_BITS : integer := 3;
  constant Idle     : std_logic_vector(0 to 2) := "000";
  constant Selected : std_logic_vector(0 to 2) := "001";
  constant Read     : std_logic_vector(0 to 2) := "011";
  constant Xfer1     : std_logic_vector(0 to 2) := "111";
```

```
  constant Xfer2     : std_logic_vector(0 to 2) := "110";

  signal present_state, next_state : std_logic_vector(0 to 2);


begin

process(rst, clk)
begin
            if rst = '1' or cs='0' then
          r_idle <= '1';
          r_common <= '0';
          r_w32 <= '0';
          r_ra <= '0'; r_rb <= '0'; r_rc <= '0'; r_xfer <= '0';
r_weth1 <= '0';
          r_weth2 <= '0'; r_weth3 <= '0';

             elsif clk'event and clk='1' then

             r_idle <= (r_idle and not cs) or (r_xfer) or (not
opb_select);
             r_common <= opb_select and (r_idle and cs);

        r_weth1 <= opb_select and (r_common and not rnw and eth_io);
        r_weth2 <= opb_select and (r_weth1);
        r_weth3 <= opb_select and (r_weth2);
             r_w32 <= opb_select and (r_common and not rnw and
fullword);

             r_ra <= opb_select and (r_common and rnw);
             r_rb  <= opb_select and (r_ra);
        r_rc <= opb_select and r_rb and (fullword or eth_io);

             r_xfer <= opb_select and ( (r_common and not rnw and
not fullword and not eth_io)
                   or (r_w32)
                   or (r_rb and not fullword and not eth_io)
                   or (r_rc)
                or (r_weth2));

             end if;
end process;




  fsm_seq : process(clk, sram_rst)
  begin
   if (sram_ce='1') then

    if sram_rst = '1' then
      present_state <= Idle;
    elsif clk'event and clk = '1' then
      present_state <= next_state;
    end if;

             xfer <= present_state(0);
             rres <= present_state(0);

             read_early <= '1'; -- hmmm   -- r_ra and eth_io;
             write_early <= '1';
             hihalf <= '0';
```

```
            wr_req_i <= not WE;
            rd_req_i <= WE;
            wr_req <= wr_req_i;
            rd_req <= rd_req_i;
            bus_req <= rd_req_i or wr_req_i or r_common;

            ce0 <= '1';
            ce1 <= '1';


   else
            xfer <= r_xfer;
            rres <= r_xfer;

            read_early <= r_ra and eth_io;
            write_early <= not ((r_common and not rnw and eth_io)
or (r_weth1) or r_weth2);
            hihalf <= r_w32 or (r_ra and fullword);

            wr_req_i <= (r_common and not rnw and not eth_io) or
(r_w32) or (r_weth1) or (r_weth2) or (r_weth3);
            rd_req_i <= (r_common and rnw) or (r_ra and (fullword
or eth_io));
            wr_req <= wr_req_i;
            rd_req <= rd_req_i;
            bus_req <= rd_req_i or wr_req_i or r_common;

            ce0 <= r_rb or (r_rc and eth_io);
            ce1 <= r_rb or r_rc;
   end if;
--   end if;

  end process fsm_seq;


  fsm_comb : process(sram_rst, present_state, sram_ce, OPB_Select,
rnw)
  begin
--  if (sram_ce = '1') then
    sram_rst <= '1';
    WE <= '1';
    output_enable <= '1';
    if rst = '1' then
      next_state <= Idle;
    else
      case present_state is
        when Idle =>
          if sram_ce = '1' then
            next_state <= Selected;
          else
            next_state <= Idle;
            WE <= '1';
                output_enable <= '1';
          end if;

        when Selected =>
          if OPB_Select = '1' then
            if RNW = '1' then
              sram_rst <= '0';
              next_state <= Read;
            else
              WE <= '0';
              next_state <= Xfer1;
```

```
            end if;
          else
            next_state <= Idle;
          end if;

        when Read =>
--          if OPB_Select = '1' then
            output_enable <= '0';
            next_state <= Xfer1;
--          else
--            next_state <= Idle;
--          end if;

        -- State encoding is critical here: xfer must only be true
here
        when Xfer1 =>
          next_state <= Idle;


        when others =>
          next_state <= Idle;
      end case;
--    end if;

ram_output_enable <= output_enable;
ram_we <= we;


    end if;
  end process fsm_comb;



end Behavioral;
```

---

opb_cntrl.vhd

```
------------------------------------------------------------------------
-- CSEE 4840 Embedded System Design
-- Jaycam
-- Yaniv Schiller - yhs2001@columbia.edu
-- Avrum Tilman - amt77@columbia.edu
-- Josh Weinberg - jmw211@columbia.edu
------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity opb_cntrl is
  generic (
    C_OPB_AWIDTH    : integer := 32;
    C_OPB_DWIDTH    : integer := 32;
    C_BASEADDR      : std_logic_vector := X"2000_0000";
    C_HIGHADDR      : std_logic_vector := X"2000_00FF"
```

```
    );

    Port (   OPB_Clk : in std_logic;
                               OPB_Rst : in std_logic;
                               OPB_ABus : in std_logic_vector (31
downto 0);
                               OPB_BE : in std_logic_vector (3 downto
0);
                               OPB_DBus : in std_logic_vector (31
downto 0);
                               OPB_RNW : in std_logic;
                               OPB_select : in std_logic;
                               OPB_seqAddr : in std_logic;
                               pixel_clock : in std_logic;
                          io_clock: in std_logic;
                               UIO_DBus : out std_logic_vector (31
downto 0);
                               UIO_errAck : out std_logic;
                               UIO_retry : out std_logic;
                               UIO_toutSup : out std_logic;
                               UIO_xferAck : out std_logic;
                               PB_A : out std_logic_vector (19 downto
0);
                               PB_UB_N : out std_logic;
                               PB_LB_N : out std_logic;
                               PB_WE_N : out std_logic;
                               PB_OE_N : out std_logic;
                               RAM_CE_N : out std_logic;
                               ETHERNET_CS_N : out std_logic;
                               ETHERNET_RDY : in std_logic;
                               ETHERNET_IREQ : in std_logic;
                               ETHERNET_IOCS16_N : in std_logic;

                               PB_D : inout std_logic_vector (15
downto 0)
                             );
end opb_cntrl;

architecture Behavioral of opb_cntrl is

signal addr_mux : std_logic_vector(19 downto 0);
signal video_addr : std_logic_vector (19 downto 0);
signal video_data : std_logic_vector (15 downto 0);
signal video_req : std_logic;
signal video_ce : std_logic;
signal i : integer;
signal cs : std_logic;

signal fullword, eth_io, read_early, write_early : std_logic ;
signal videocycle, amuxsel, hihalf : std_logic;
signal rce0, rce1, rreset : std_logic;
signal xfer : std_logic;
signal wr_req, rd_req, bus_req : std_logic;
signal pb_rd, pb_wr : std_logic;

signal sram_ce : std_logic;
signal ethernet_ce : std_logic;


signal rnw, ram_we, ram_oe, oe : std_logic;

signal addr : std_logic_vector (23 downto 0);
```

```
signal be : std_logic_vector (3 downto 0);
signal pb_bytesel : std_logic_vector (1 downto 0);

signal wdata : std_logic_vector (31 downto 0);
signal wdata_mux : std_logic_vector (15 downto 0);

signal rdata : std_logic_vector (15 downto 0);  -- register data read
- FDRE



component memoryctrl
Port (         rst : in std_logic;
           clk : in std_logic;
           cs : in std_logic; -- any of my devices selected
           opb_select : in std_logic; -- original select
           rnw : in std_logic;
           eth_io : in std_logic;
           fullword : in std_logic;
           read_early : out std_logic;
           write_early : out std_logic;
           videocycle : out std_logic;
           hihalf : out std_logic;
           wr_req : out std_logic;
           rd_req : out std_logic;
           bus_req : out std_logic;
           xfer : out std_logic;
           ce0 : out std_logic;
           ce1 : out std_logic;
           rres : out std_logic;
                ram_output_enable : out std_logic;
                ram_WE : out std_logic;
                sram_ce : in std_logic);
end component;

component pad_io
    Port ( sys_clk : in std_logic;
           io_clock : in std_logic;
           read_early : in std_logic;
           write_early : in std_logic;
           rst : in std_logic;
           PB_A : out std_logic_vector(19 downto 0);
           PB_UB_N : out std_logic;
           PB_LB_N : out std_logic;
           PB_WE_N : out std_logic;
           PB_OE_N : out std_logic;
           RAM_CE_N : out std_logic;
           ETHERNET_CS_N : out std_logic;
           ETHERNET_RDY : in std_logic;
           ETHERNET_IREQ : in std_logic;
           ETHERNET_IOCS16_N : in std_logic;
           PB_D : inout std_logic_vector(15 downto 0);

           pb_addr : in std_logic_vector(19 downto 0);
           pb_ub : in std_logic;
           pb_lb : in std_logic;
           pb_wr : in std_logic;
           pb_rd : in std_logic;
           ram_ce : in std_logic;
                ethernet_ce : in std_logic;
           pb_dread : out std_logic_vector(15 downto 0);
           pb_dwrite : in std_logic_vector(15 downto 0));
end component;
```

```
begin



-- the controller state machine
memoryctrl1 : memoryctrl
port map    (    rst => OPB_Rst,
                      clk => OPB_Clk,
                      cs => cs,
                      opb_select => OPB_select,
                      rnw => rnw,

                      fullword => fullword,
                      eth_io =>eth_io,
                      read_early => read_early,
                      write_early => write_early,
                      videocycle => videocycle,
                      hihalf => hihalf,
                      wr_req => wr_req,
                      rd_req => rd_req,
                      bus_req => bus_req,

                      xfer => xfer,
                      ce0 => rce0,
                      ce1 => rce1,
                      rres => rreset,


                      ram_output_enable => ram_oe,
                        ram_WE => ram_we,
                      sram_ce => sram_ce
                      );

-- PADS

pad_io1 : pad_io
port map    (    sys_clk => OPB_Clk,
                      io_clock => io_clock,
                      read_early => read_early,
                      write_early => write_early,
                      rst => OPB_Rst,
                      PB_A => PB_A,
                      PB_UB_N => PB_UB_N,
                      PB_LB_N => PB_LB_N,
                      PB_WE_N => PB_WE_N,
                      PB_OE_N => PB_OE_N,
                      RAM_CE_N => RAM_CE_N,
                      ETHERNET_CS_N => ETHERNET_CS_N,
                      ETHERNET_RDY => ETHERNET_RDY,
                      ETHERNET_IREQ => ETHERNET_IREQ,
                      ETHERNET_IOCS16_N => ETHERNET_IOCS16_N,
                      PB_D => PB_D,

                      pb_addr => addr_mux,
                      pb_wr => pb_wr,
                      pb_rd => oe,
                      pb_ub => pb_bytesel(1),
                      pb_lb => pb_bytesel(0),
                      ram_ce => sram_ce,
                      ethernet_ce => ethernet_ce,
```

```
                        pb_dread  => rdata,
                        pb_dwrite => wdata_mux);


--           amuxsel <= videocycle;
             video_ce <= '0';
             video_req <= '0';

             addr_mux <= (addr(20 downto 2)) & (addr(1) or hihalf);

             fullword <= be(2) and be(0);

             wdata_mux <= wdata(15 downto 0) when ((addr(1) or
hihalf) = '1') else wdata(31 downto 16);




pb_rd <= rd_req;
pb_wr <= wr_req;
oe <= ram_oe when (sram_ce='1') else pb_rd;
--ram_oe active low

cs <= OPB_select when OPB_ABus(31 downto 23) = "000000001" else '0';
sram_ce <= '1' when addr(22 downto 21)="10" else '0';
ethernet_ce <= '1' when addr(22 downto 21) ="01" else '0';
eth_io <= '1' when addr(22 downto 21) /= "10" else '0';



process (OPB_Clk, OPB_Rst)
begin

-- register rw
             if OPB_Clk'event and OPB_Clk ='1' then

                  if OPB_Rst = '1' then
                        rnw <= '0';
                  else
                        rnw <= OPB_RNW;
                  end if;

             end if;

-- register adresses A23 .. A0
             if OPB_Clk'event and OPB_Clk ='1' then
                for i in 0 to 23 loop
                        if OPB_Rst = '1' then
                              addr(i) <= '0';
                        else
                              addr(i) <= OPB_ABus(i);
                        end if;
             end loop;
             end if;

             -- register BE
             if OPB_Clk'event and OPB_Clk ='1' then
                        if OPB_Rst = '1' then
                              be <= "0000";
                        else
                              be <= OPB_BE;
                        end if;
             end if;
```

```
-- register data write
             if OPB_Clk'event and OPB_Clk ='1' then
                for i in 0 to 31 loop
                        if OPB_Rst = '1' then
                              wdata(i) <= '0';
                        else
                              wdata(i) <= OPB_DBus(i);
                        end if;
             end loop;
             end if;

-- the fun begins
-- ce0/ce1 enables writing MSB (low) / LSB (high) halves

--always @(posedge OPB_Rst or posedge OPB_Clk) begin (synchrounous
or asynchronous reset??)

                  for i in 0 to 15 loop
                        if OPB_Rst = '1' then
                              UIO_DBus(i) <= '0';

                        elsif OPB_Clk'event and OPB_Clk ='1' then
                              if rreset = '1' then
                                    UIO_DBus(i) <= '0';
                              elsif (rce1 or rce0) = '1' or
ram_oe='0' then
                                    UIO_DBus(i) <= rdata(i);
                              end if;
                        end if;
                  end loop;

                  for i in 16 to 31 loop
                        if OPB_Rst = '1' then
                              UIO_DBus(i) <= '0';
                        elsif OPB_Clk'event and OPB_Clk ='1' then

                              if rreset = '1' then
                                    UIO_DBus(i) <= '0';
                              elsif rce0 = '1' then
                                    UIO_DBus(i) <= rdata(i-16);
                              end if;
                        end if;
                  end loop;



end process;

-- tie unused to ground
UIO_errAck <= '0';
UIO_retry <= '0';
UIO_toutSup <= '0';

UIO_xferAck <= xfer;


end Behavioral;
```

packets.c

```
#include <stdio.h>
#include <stdlib.h>


  int numberToCall[10];

  struct packets {
    int value;
  } packet[2000];


//numerical form - 128.59...
void addIPNumerical(struct packets *packet, int *ipoffset) {
  packet[(*ipoffset)++].value=0x31;
  packet[(*ipoffset)++].value=0x32;
  packet[(*ipoffset)++].value=0x38;
  packet[(*ipoffset)++].value=0x2e;
  packet[(*ipoffset)++].value=0x35;
  packet[(*ipoffset)++].value=0x39;
  packet[(*ipoffset)++].value=0x2e;
  packet[(*ipoffset)++].value=0x31;
  packet[(*ipoffset)++].value=0x34;
  packet[(*ipoffset)++].value=0x39;
  packet[(*ipoffset)++].value=0x2e;
  packet[(*ipoffset)++].value=0x31;
  packet[(*ipoffset)++].value=0x30;
  packet[(*ipoffset)++].value=0x36;

}


//form of 80, 3b, 85, a2
void addIPHex(struct packets *packet, int *ipoffset) {
  packet[(*ipoffset)++].value=0x80; //source ip
  packet[(*ipoffset)++].value=0x3b;
  packet[(*ipoffset)++].value=0x95;
  packet[(*ipoffset)++].value=0x6a;
}

void addUDPIPHeader(struct packets *packet, int *headeroffset, int
packetlengthbyte1, int packetlengthbyte2, int checksum1, int
checksum2) {
  unsigned int packetlength = (packetlengthbyte1<<8) +
packetlengthbyte2 - 20;


  packet[(*headeroffset)++].value=0x00;
  packet[(*headeroffset)++].value=0xd0;
  packet[(*headeroffset)++].value=0x06;
  packet[(*headeroffset)++].value=0x26; //26 vs 24
  packet[(*headeroffset)++].value=0x9c; //9c vs c8
  packet[(*headeroffset)++].value=0x0a;
  packet[(*headeroffset)++].value=0x00;
  packet[(*headeroffset)++].value=0x08;
  packet[(*headeroffset)++].value=0x02;

  packet[(*headeroffset)++].value=0x62;
  packet[(*headeroffset)++].value=0x30;
  packet[(*headeroffset)++].value=0x5d;
```

```
  packet[(*headeroffset)++].value=0x08;
  packet[(*headeroffset)++].value=0x00;
  packet[(*headeroffset)++].value=0x45;
  packet[(*headeroffset)++].value=0x00; //expedited = b8.. on for
rtp, not for sip



  packet[(*headeroffset)++].value=packetlengthbyte1; //total packet
length
  packet[(*headeroffset)++].value=packetlengthbyte2;
  packet[(*headeroffset)++].value=0x04;
  packet[(*headeroffset)++].value=0x41;
  packet[(*headeroffset)++].value=0x00;
  packet[(*headeroffset)++].value=0x00;
  packet[(*headeroffset)++].value=0x80;
  packet[(*headeroffset)++].value=0x11;
  packet[(*headeroffset)++].value=checksum1; //checksum
  packet[(*headeroffset)++].value=checksum2;

  addIPHex(packet, headeroffset);

  packet[(*headeroffset)++].value=0x80;
  packet[(*headeroffset)++].value=0x3b;
  packet[(*headeroffset)++].value=0x27;
  packet[(*headeroffset)++].value=0x7f;
  packet[(*headeroffset)++].value=0x13;
  packet[(*headeroffset)++].value=0xc4;
  packet[(*headeroffset)++].value=0x13;
  packet[(*headeroffset)++].value=0xc4;
  packet[(*headeroffset)++].value=(packetlength >> 8); //udp length
  packet[(*headeroffset)++].value=(packetlength);
  packet[(*headeroffset)++].value=0x00; //50, 24 //disable udp
checksum
  packet[(*headeroffset)++].value=0x00;
}


void addSIPCallID(struct packets *packet, int *callidoffset) {
  packet[(*callidoffset)++].value=0x43;
  packet[(*callidoffset)++].value=0x61;
  packet[(*callidoffset)++].value=0x6c;
  packet[(*callidoffset)++].value=0x6c;
  packet[(*callidoffset)++].value=0x2d;
  packet[(*callidoffset)++].value=0x49;
  packet[(*callidoffset)++].value=0x44;
  packet[(*callidoffset)++].value=0x3a;
  packet[(*callidoffset)++].value=0x20;

  //random thingy
  packet[(*callidoffset)++].value=0x45;
  packet[(*callidoffset)++].value=0x45;
  packet[(*callidoffset)++].value=0x45;
  packet[(*callidoffset)++].value=0x33;
  packet[(*callidoffset)++].value=0x35;
  packet[(*callidoffset)++].value=0x35;
  packet[(*callidoffset)++].value=0x42;
  packet[(*callidoffset)++].value=0x31;
  packet[(*callidoffset)++].value=0x2d;
  packet[(*callidoffset)++].value=0x34;
  packet[(*callidoffset)++].value=0x32;
  packet[(*callidoffset)++].value=0x30;
  packet[(*callidoffset)++].value=0x33; //0
```

```
  packet[(*callidoffset)++].value=0x2d;
  packet[(*callidoffset)++].value=0x34;
  packet[(*callidoffset)++].value=0x45;
  packet[(*callidoffset)++].value=0x36;
  packet[(*callidoffset)++].value=0x33;
  packet[(*callidoffset)++].value=0x2d;
  packet[(*callidoffset)++].value=0x38;
  packet[(*callidoffset)++].value=0x41;
  packet[(*callidoffset)++].value=0x39;
  packet[(*callidoffset)++].value=0x35;
  packet[(*callidoffset)++].value=0x2d;
  packet[(*callidoffset)++].value=0x43;
  packet[(*callidoffset)++].value=0x43;
  packet[(*callidoffset)++].value=0x34;
  packet[(*callidoffset)++].value=0x31;
  packet[(*callidoffset)++].value=0x42;
  packet[(*callidoffset)++].value=0x37;
  packet[(*callidoffset)++].value=0x33;
  packet[(*callidoffset)++].value=0x31;
  packet[(*callidoffset)++].value=0x39;
  packet[(*callidoffset)++].value=0x30;
  packet[(*callidoffset)++].value=0x41;
  packet[(*callidoffset)++].value=0x43;

  packet[(*callidoffset)++].value=0x40;
  //ip
  addIPNumerical(packet, callidoffset);


  packet[(*callidoffset)++].value=0x0d;
  packet[(*callidoffset)++].value=0x0a;
}

void addSIPToLocal(struct packets *packet, int *tooffset) {
  packet[(*tooffset)++].value=0x54;
  packet[(*tooffset)++].value=0x6f;
  packet[(*tooffset)++].value=0x3a;
  packet[(*tooffset)++].value=0x20;
  packet[(*tooffset)++].value=0x3c;
  packet[(*tooffset)++].value=0x73;
  packet[(*tooffset)++].value=0x69;
  packet[(*tooffset)++].value=0x70;
  packet[(*tooffset)++].value=0x3a;
  packet[(*tooffset)++].value=0x37;  //start phone #
  packet[(*tooffset)++].value=0x31;
  packet[(*tooffset)++].value=0x38;
  packet[(*tooffset)++].value=0x35;
  packet[(*tooffset)++].value=0x30;
  packet[(*tooffset)++].value=0x34;
  packet[(*tooffset)++].value=0x32;
  packet[(*tooffset)++].value=0x31;
  packet[(*tooffset)++].value=0x35;
  packet[(*tooffset)++].value=0x34;
  packet[(*tooffset)++].value=0x40;
  packet[(*tooffset)++].value=0x73;
  packet[(*tooffset)++].value=0x69;
  packet[(*tooffset)++].value=0x70;
  packet[(*tooffset)++].value=0x2e;
  packet[(*tooffset)++].value=0x62;
  packet[(*tooffset)++].value=0x72;
  packet[(*tooffset)++].value=0x6f;
  packet[(*tooffset)++].value=0x61;
  packet[(*tooffset)++].value=0x64;
```

```
  packet[(*tooffset)++].value=0x76;
  packet[(*tooffset)++].value=0x6f;
  packet[(*tooffset)++].value=0x69;
  packet[(*tooffset)++].value=0x63;
  packet[(*tooffset)++].value=0x65;
  packet[(*tooffset)++].value=0x2e;
  packet[(*tooffset)++].value=0x63;
  packet[(*tooffset)++].value=0x6f;
  packet[(*tooffset)++].value=0x6d;
  packet[(*tooffset)++].value=0x3e;
  packet[(*tooffset)++].value=0x0d;
  packet[(*tooffset)++].value=0x0a;
}



void addSIPVia(struct packets *packet, int *offset) {
  //VIA
  packet[(*offset)++].value=0x56;
  packet[(*offset)++].value=0x69;
  packet[(*offset)++].value=0x61;
  packet[(*offset)++].value=0x3a;
  packet[(*offset)++].value=0x20;
  packet[(*offset)++].value=0x53;
  packet[(*offset)++].value=0x49;
  packet[(*offset)++].value=0x50;
  packet[(*offset)++].value=0x2f;
  packet[(*offset)++].value=0x32;
  packet[(*offset)++].value=0x2e;
  packet[(*offset)++].value=0x30;
  packet[(*offset)++].value=0x2f;
  packet[(*offset)++].value=0x55;
  packet[(*offset)++].value=0x44;
  packet[(*offset)++].value=0x50;
  packet[(*offset)++].value=0x20;

  addIPNumerical(packet, offset);

  packet[(*offset)++].value=0x3a;
  packet[(*offset)++].value=0x35;
  packet[(*offset)++].value=0x30;
  packet[(*offset)++].value=0x36;
  packet[(*offset)++].value=0x30;
  packet[(*offset)++].value=0x3b;

  packet[(*offset)++].value=0x72;
  packet[(*offset)++].value=0x70;
  packet[(*offset)++].value=0x6f;
  packet[(*offset)++].value=0x72;
  packet[(*offset)++].value=0x74;

  packet[(*offset)++].value=0x3b;

  packet[(*offset)++].value=0x62;
  packet[(*offset)++].value=0x72;
  packet[(*offset)++].value=0x61;
  packet[(*offset)++].value=0x6e;
  packet[(*offset)++].value=0x63;
  packet[(*offset)++].value=0x68;
  packet[(*offset)++].value=0x3d;

  packet[(*offset)++].value=0x7a; //branch
  packet[(*offset)++].value=0x39;
```

```
  packet[(*offset)++].value=0x31;
  packet[(*offset)++].value=0x68;
  packet[(*offset)++].value=0x47;
  packet[(*offset)++].value=0x34;
  packet[(*offset)++].value=0x62;
  packet[(*offset)++].value=0x4b;
  packet[(*offset)++].value=0x38;
  packet[(*offset)++].value=0x30;
  packet[(*offset)++].value=0x33;
  packet[(*offset)++].value=0x62;
  packet[(*offset)++].value=0x39;
  packet[(*offset)++].value=0x35;
  packet[(*offset)++].value=0x61;
  packet[(*offset)++].value=0x32;
  packet[(*offset)++].value=0x30;
  packet[(*offset)++].value=0x31;
  packet[(*offset)++].value=0x33;
  packet[(*offset)++].value=0x31;
  packet[(*offset)++].value=0x63;
  packet[(*offset)++].value=0x39;
  packet[(*offset)++].value=0x62;
  packet[(*offset)++].value=0x31;
  packet[(*offset)++].value=0x34;
  packet[(*offset)++].value=0x32;
  packet[(*offset)++].value=0x35;
  packet[(*offset)++].value=0x64;
  packet[(*offset)++].value=0x63;
  packet[(*offset)++].value=0x66;
  packet[(*offset)++].value=0x38;
  packet[(*offset)++].value=0x38;
  packet[(*offset)++].value=0x30;
  packet[(*offset)++].value=0x31;
  packet[(*offset)++].value=0x34;
  packet[(*offset)++].value=0x30;
  packet[(*offset)++].value=0x35;
  packet[(*offset)++].value=0x61;
  packet[(*offset)++].value=0x38;
  packet[(*offset)++].value=0x31;
  packet[(*offset)++].value=0x30;
  packet[(*offset)++].value=0x30;
  packet[(*offset)++].value=0x30;
  packet[(*offset)++].value=0x30;
  packet[(*offset)++].value=0x30;
  packet[(*offset)++].value=0x30;
  packet[(*offset)++].value=0x30;
  packet[(*offset)++].value=0x30;
  packet[(*offset)++].value=0x31;

  packet[(*offset)++].value=0x0d;
  packet[(*offset)++].value=0x0a;
}


void addSIPContact(struct packets *packet, int *offset) {
  packet[(*offset)++].value=0x43;
  packet[(*offset)++].value=0x6f;
  packet[(*offset)++].value=0x6e;
  packet[(*offset)++].value=0x74;
  packet[(*offset)++].value=0x61;
  packet[(*offset)++].value=0x63;
  packet[(*offset)++].value=0x74;
  packet[(*offset)++].value=0x3a;
  packet[(*offset)++].value=0x20;
  packet[(*offset)++].value=0x3c;
```

```
  packet[(*offset)++].value=0x73;
  packet[(*offset)++].value=0x69;
  packet[(*offset)++].value=0x70;
  packet[(*offset)++].value=0x3a;
  packet[(*offset)++].value=0x33;
  packet[(*offset)++].value=0x35;
  packet[(*offset)++].value=0x33;
  packet[(*offset)++].value=0x36;
  packet[(*offset)++].value=0x30;
  packet[(*offset)++].value=0x40;

  addIPNumerical(packet, offset);

  packet[(*offset)++].value=0x3a;
  packet[(*offset)++].value=0x35;
  packet[(*offset)++].value=0x30;
  packet[(*offset)++].value=0x36;
  packet[(*offset)++].value=0x30; /*

  packet[(*offset)++].value=0x3e;
  packet[(*offset)++].value=0x2e;
  packet[(*offset)++].value=0x31;
  packet[(*offset)++].value=0x36;
  packet[(*offset)++].value=0x32;
  packet[(*offset)++].value=0x3a;
  packet[(*offset)++].value=0x35;
  packet[(*offset)++].value=0x30;
  packet[(*offset)++].value=0x36;
  packet[(*offset)++].value=0x30;*/
  packet[(*offset)++].value=0x3e;
  packet[(*offset)++].value=0x0d;
  packet[(*offset)++].value=0x0a;

}

void addSIPToRemote(struct packets *packet, int *tooffset) {
  packet[(*tooffset)++].value=0x54;
  packet[(*tooffset)++].value=0x6f;
  packet[(*tooffset)++].value=0x3a;
  packet[(*tooffset)++].value=0x20;
  packet[(*tooffset)++].value=0x3c;
  packet[(*tooffset)++].value=0x73;
  packet[(*tooffset)++].value=0x69;
  packet[(*tooffset)++].value=0x70;
  packet[(*tooffset)++].value=0x3a;
  packet[(*tooffset)++].value=0x30 + numberToCall[0];  //start phone
#
  packet[(*tooffset)++].value=0x30 + numberToCall[1];
  packet[(*tooffset)++].value=0x30 + numberToCall[2];
  packet[(*tooffset)++].value=0x30 + numberToCall[3];
  packet[(*tooffset)++].value=0x30 + numberToCall[4];
/*  packet[(*tooffset)++].value=0x30 + numberToCall[5];
  packet[(*tooffset)++].value=0x30 + numberToCall[6];
  packet[(*tooffset)++].value=0x30 + numberToCall[7];
  packet[(*tooffset)++].value=0x30 + numberToCall[8];
  packet[(*tooffset)++].value=0x30 + numberToCall[9]; //end phone #
*/
  packet[(*tooffset)++].value=0x40;
  packet[(*tooffset)++].value=0x63;
  packet[(*tooffset)++].value=0x6f;
  packet[(*tooffset)++].value=0x6c;
  packet[(*tooffset)++].value=0x75;
  packet[(*tooffset)++].value=0x6d;
```

```
  packet[(*tooffset)++].value=0x62;
  packet[(*tooffset)++].value=0x69;
  packet[(*tooffset)++].value=0x61;
  packet[(*tooffset)++].value=0x2e;
  packet[(*tooffset)++].value=0x65;
  packet[(*tooffset)++].value=0x64;
  packet[(*tooffset)++].value=0x75;
  packet[(*tooffset)++].value=0x3e;
  /*
  packet[(*tooffset)++].value=0x65;
  packet[(*tooffset)++].value=0x2e;
  packet[(*tooffset)++].value=0x63;
  packet[(*tooffset)++].value=0x6f;
  packet[(*tooffset)++].value=0x6d;
  packet[(*tooffset)++].value=0x3e; */
  packet[(*tooffset)++].value=0x0d;
  packet[(*tooffset)++].value=0x0a;
}



void addSIPToRemoteWithTag(struct packets *packet, int *tooffset) {
  packet[(*tooffset)++].value=0x54;
  packet[(*tooffset)++].value=0x6f;
  packet[(*tooffset)++].value=0x3a;
  packet[(*tooffset)++].value=0x20;
  packet[(*tooffset)++].value=0x3c;
  packet[(*tooffset)++].value=0x73;
  packet[(*tooffset)++].value=0x69;
  packet[(*tooffset)++].value=0x70;
  packet[(*tooffset)++].value=0x3a;
  packet[(*tooffset)++].value=0x30 + numberToCall[0];  //start phone
#
  packet[(*tooffset)++].value=0x30 + numberToCall[1];
  packet[(*tooffset)++].value=0x30 + numberToCall[2];
  packet[(*tooffset)++].value=0x30 + numberToCall[3];
  packet[(*tooffset)++].value=0x30 + numberToCall[4];
  packet[(*tooffset)++].value=0x40;
  packet[(*tooffset)++].value=0x63;
  packet[(*tooffset)++].value=0x6f;
  packet[(*tooffset)++].value=0x6c;
  packet[(*tooffset)++].value=0x75;
  packet[(*tooffset)++].value=0x6d;
  packet[(*tooffset)++].value=0x62;
  packet[(*tooffset)++].value=0x69;
  packet[(*tooffset)++].value=0x61;
  packet[(*tooffset)++].value=0x2e;
  packet[(*tooffset)++].value=0x65;
  packet[(*tooffset)++].value=0x64;
  packet[(*tooffset)++].value=0x75;
  packet[(*tooffset)++].value=0x3e;

  packet[(*tooffset)++].value=0x3b;
  packet[(*tooffset)++].value=0x74;
  packet[(*tooffset)++].value=0x61;
  packet[(*tooffset)++].value=0x67;
  packet[(*tooffset)++].value=0x3d;

  //space to fill in tag
  /*
  packet[(*tooffset)++].value=0x00;
  packet[(*tooffset)++].value=0x00;
```

```
  packet[(*tooffset)++].value=0x00;
  packet[(*tooffset)++].value=0x00;
  packet[(*tooffset)++].value=0x00;
  packet[(*tooffset)++].value=0x00;
  packet[(*tooffset)++].value=0x00;
  packet[(*tooffset)++].value=0x00;
  packet[(*tooffset)++].value=0x00;
  packet[(*tooffset)++].value=0x00;
  */


  /*
  packet[(*tooffset)++].value=0x65;
  packet[(*tooffset)++].value=0x2e;
  packet[(*tooffset)++].value=0x63;
  packet[(*tooffset)++].value=0x6f;
  packet[(*tooffset)++].value=0x6d;
  packet[(*tooffset)++].value=0x3e; */
  packet[(*tooffset)++].value=0x0d;
  packet[(*tooffset)++].value=0x0a;
}




void addSIPFrom(struct packets *packet, int *fromoffset) {
  packet[(*fromoffset)++].value=0x46;
  packet[(*fromoffset)++].value=0x72;
  packet[(*fromoffset)++].value=0x6f;
  packet[(*fromoffset)++].value=0x6d;
  packet[(*fromoffset)++].value=0x3a;
  packet[(*fromoffset)++].value=0x20;
  packet[(*fromoffset)++].value=0x52;
  packet[(*fromoffset)++].value=0x61;
  packet[(*fromoffset)++].value=0x6a;
  packet[(*fromoffset)++].value=0x20;
  packet[(*fromoffset)++].value=0x42;
  packet[(*fromoffset)++].value=0x61;
  packet[(*fromoffset)++].value=0x6b;
  packet[(*fromoffset)++].value=0x68;
  packet[(*fromoffset)++].value=0x72;
  packet[(*fromoffset)++].value=0x75;
  packet[(*fromoffset)++].value=0x20;
  packet[(*fromoffset)++].value=0x3c;
  packet[(*fromoffset)++].value=0x73;
  packet[(*fromoffset)++].value=0x69;
  packet[(*fromoffset)++].value=0x70;
  packet[(*fromoffset)++].value=0x3a;
  packet[(*fromoffset)++].value=0x33;
  packet[(*fromoffset)++].value=0x35;
  packet[(*fromoffset)++].value=0x33;
  packet[(*fromoffset)++].value=0x36;
  packet[(*fromoffset)++].value=0x30;
  packet[(*fromoffset)++].value=0x40;
  packet[(*fromoffset)++].value=0x63;
  packet[(*fromoffset)++].value=0x6f;
  packet[(*fromoffset)++].value=0x6c;
  packet[(*fromoffset)++].value=0x75;
  packet[(*fromoffset)++].value=0x6d;
  packet[(*fromoffset)++].value=0x62;
  packet[(*fromoffset)++].value=0x69;
```

```
  packet[(*fromoffset)++].value=0x61;
  packet[(*fromoffset)++].value=0x2e;
  packet[(*fromoffset)++].value=0x65;
  packet[(*fromoffset)++].value=0x64;
  packet[(*fromoffset)++].value=0x75;/*
  packet[(*fromoffset)++].value=0x3a;
  packet[(*fromoffset)++].value=0x35;
  packet[(*fromoffset)++].value=0x30;
  packet[(*fromoffset)++].value=0x36;
  packet[(*fromoffset)++].value=0x30;*/
  packet[(*fromoffset)++].value=0x3e;
  packet[(*fromoffset)++].value=0x3b;

  packet[(*fromoffset)++].value=0x74;
  packet[(*fromoffset)++].value=0x61;
  packet[(*fromoffset)++].value=0x67;
  packet[(*fromoffset)++].value=0x3d;
  //tag
  packet[(*fromoffset)++].value=0x38;
  packet[(*fromoffset)++].value=0x37;
  packet[(*fromoffset)++].value=0x35;
  packet[(*fromoffset)++].value=0x35;
  packet[(*fromoffset)++].value=0x32;
  packet[(*fromoffset)++].value=0x33;
  packet[(*fromoffset)++].value=0x39;


  packet[(*fromoffset)++].value=0x0d;
  packet[(*fromoffset)++].value=0x0a;
}

void addSIPAuth(struct packets *packet, int *authoffset) {
  packet[(*authoffset)++].value = 0x41;
  packet[(*authoffset)++].value = 0x75;
  packet[(*authoffset)++].value = 0x74;
  packet[(*authoffset)++].value = 0x68;
  packet[(*authoffset)++].value = 0x6f;
  packet[(*authoffset)++].value = 0x72;
  packet[(*authoffset)++].value = 0x69;
  packet[(*authoffset)++].value = 0x7a;
  packet[(*authoffset)++].value = 0x61;
  packet[(*authoffset)++].value = 0x74;
  packet[(*authoffset)++].value = 0x69;
  packet[(*authoffset)++].value = 0x6f;
  packet[(*authoffset)++].value = 0x6e;
  packet[(*authoffset)++].value = 0x3a;
  packet[(*authoffset)++].value = 0x20;
  packet[(*authoffset)++].value = 0x44;
  packet[(*authoffset)++].value = 0x69;
  packet[(*authoffset)++].value = 0x67;
  packet[(*authoffset)++].value = 0x65;
  packet[(*authoffset)++].value = 0x73;
  packet[(*authoffset)++].value = 0x74;
  packet[(*authoffset)++].value = 0x20;
  packet[(*authoffset)++].value = 0x75;
  packet[(*authoffset)++].value = 0x73;
  packet[(*authoffset)++].value = 0x65;
  packet[(*authoffset)++].value = 0x72;
  packet[(*authoffset)++].value = 0x6e;
  packet[(*authoffset)++].value = 0x61;
  packet[(*authoffset)++].value = 0x6d;
  packet[(*authoffset)++].value = 0x65;
  packet[(*authoffset)++].value = 0x3d;
```

```
packet[(*authoffset)++].value = 0x22;
packet[(*authoffset)++].value = 0x37;
packet[(*authoffset)++].value = 0x31;
packet[(*authoffset)++].value = 0x38;
packet[(*authoffset)++].value = 0x35;
packet[(*authoffset)++].value = 0x30;
packet[(*authoffset)++].value = 0x34;
packet[(*authoffset)++].value = 0x32;
packet[(*authoffset)++].value = 0x31;
packet[(*authoffset)++].value = 0x35;
packet[(*authoffset)++].value = 0x34;
packet[(*authoffset)++].value = 0x22;
packet[(*authoffset)++].value = 0x2c;
packet[(*authoffset)++].value = 0x72;
packet[(*authoffset)++].value = 0x65;
packet[(*authoffset)++].value = 0x61;
packet[(*authoffset)++].value = 0x6c;
packet[(*authoffset)++].value = 0x6d;
packet[(*authoffset)++].value = 0x3d;
packet[(*authoffset)++].value = 0x22;
packet[(*authoffset)++].value = 0x42;
packet[(*authoffset)++].value = 0x72;
packet[(*authoffset)++].value = 0x6f;
packet[(*authoffset)++].value = 0x61;
packet[(*authoffset)++].value = 0x64;
packet[(*authoffset)++].value = 0x57;
packet[(*authoffset)++].value = 0x6f;
packet[(*authoffset)++].value = 0x72;
packet[(*authoffset)++].value = 0x6b;
packet[(*authoffset)++].value = 0x73;
packet[(*authoffset)++].value = 0x22;
packet[(*authoffset)++].value = 0x2c;
packet[(*authoffset)++].value = 0x6e;
packet[(*authoffset)++].value = 0x6f;
packet[(*authoffset)++].value = 0x6e;
packet[(*authoffset)++].value = 0x63;
packet[(*authoffset)++].value = 0x65;
packet[(*authoffset)++].value = 0x3d;
packet[(*authoffset)++].value = 0x22;
packet[(*authoffset)++].value = 0x31;
packet[(*authoffset)++].value = 0x31;
packet[(*authoffset)++].value = 0x31;
packet[(*authoffset)++].value = 0x33;
packet[(*authoffset)++].value = 0x34;
packet[(*authoffset)++].value = 0x32;
packet[(*authoffset)++].value = 0x31;
packet[(*authoffset)++].value = 0x34;
packet[(*authoffset)++].value = 0x32;
packet[(*authoffset)++].value = 0x38;
packet[(*authoffset)++].value = 0x30;
packet[(*authoffset)++].value = 0x31;
packet[(*authoffset)++].value = 0x33;
packet[(*authoffset)++].value = 0x22;
packet[(*authoffset)++].value = 0x2c;
packet[(*authoffset)++].value = 0x75;
packet[(*authoffset)++].value = 0x72;
packet[(*authoffset)++].value = 0x69;
packet[(*authoffset)++].value = 0x3d;
packet[(*authoffset)++].value = 0x22;
packet[(*authoffset)++].value = 0x73;
packet[(*authoffset)++].value = 0x69;
packet[(*authoffset)++].value = 0x70;
packet[(*authoffset)++].value = 0x3a;
```

```
packet[(*authoffset)++].value = 0x73;
packet[(*authoffset)++].value = 0x69;
packet[(*authoffset)++].value = 0x70;
packet[(*authoffset)++].value = 0x2e;
packet[(*authoffset)++].value = 0x72;
packet[(*authoffset)++].value = 0x65;
packet[(*authoffset)++].value = 0x73;
packet[(*authoffset)++].value = 0x70;
packet[(*authoffset)++].value = 0x6f;
packet[(*authoffset)++].value = 0x6e;
packet[(*authoffset)++].value = 0x73;
packet[(*authoffset)++].value = 0x65;
packet[(*authoffset)++].value = 0x3d;
packet[(*authoffset)++].value = 0x22;
packet[(*authoffset)++].value = 0x33;
packet[(*authoffset)++].value = 0x65;
packet[(*authoffset)++].value = 0x39;
packet[(*authoffset)++].value = 0x61;
packet[(*authoffset)++].value = 0x63;
packet[(*authoffset)++].value = 0x66;
packet[(*authoffset)++].value = 0x30;
packet[(*authoffset)++].value = 0x34;
packet[(*authoffset)++].value = 0x37;
packet[(*authoffset)++].value = 0x61;
packet[(*authoffset)++].value = 0x31;
packet[(*authoffset)++].value = 0x37;
packet[(*authoffset)++].value = 0x34;
packet[(*authoffset)++].value = 0x39;
packet[(*authoffset)++].value = 0x35;
packet[(*authoffset)++].value = 0x37;
packet[(*authoffset)++].value = 0x66;
packet[(*authoffset)++].value = 0x64;
packet[(*authoffset)++].value = 0x34;
packet[(*authoffset)++].value = 0x30;
packet[(*authoffset)++].value = 0x66;
packet[(*authoffset)++].value = 0x61;
packet[(*authoffset)++].value = 0x30;
packet[(*authoffset)++].value = 0x32;
packet[(*authoffset)++].value = 0x63;
packet[(*authoffset)++].value = 0x36;
packet[(*authoffset)++].value = 0x30;
packet[(*authoffset)++].value = 0x30;
packet[(*authoffset)++].value = 0x65;
packet[(*authoffset)++].value = 0x31;
packet[(*authoffset)++].value = 0x64;
packet[(*authoffset)++].value = 0x65;
packet[(*authoffset)++].value = 0x22;
packet[(*authoffset)++].value = 0x2c;
packet[(*authoffset)++].value = 0x61;
packet[(*authoffset)++].value = 0x6c;
packet[(*authoffset)++].value = 0x67;
packet[(*authoffset)++].value = 0x6f;
packet[(*authoffset)++].value = 0x72;
packet[(*authoffset)++].value = 0x69;
packet[(*authoffset)++].value = 0x74;
packet[(*authoffset)++].value = 0x68;
packet[(*authoffset)++].value = 0x6d;
packet[(*authoffset)++].value = 0x3d;
packet[(*authoffset)++].value = 0x22;
packet[(*authoffset)++].value = 0x4d;
packet[(*authoffset)++].value = 0x44;
packet[(*authoffset)++].value = 0x35;
packet[(*authoffset)++].value = 0x22;
```

```
  packet[(*authoffset)++].value = 0x0d;
  packet[(*authoffset)++].value = 0x0a;
}


int main()
{

  FILE * fp;
  int i;
  int packetoffset=0;

  fp = fopen("sram.bin", "w+b");



  printf("Please enter the Columbia extension to call: ");
  for (i =0; i<5; i++) numberToCall[i] = getchar() - 0x30;

  printf("\r\nPlease wait while I configure the system to call x");

  for (i=0; i<5; i++) putchar(numberToCall[i] + 0x30);


  if (fp == NULL) {
    printf("Error opening\n");
  }

  addUDPIPHeader(packet, &packetoffset, 0x02, 0x21, 0x76, 0xdf);


  //SIP REGISTER header
  packet[packetoffset++].value=0x52;
  packet[packetoffset++].value=0x45;
  packet[packetoffset++].value=0x47;
  packet[packetoffset++].value=0x49;
  packet[packetoffset++].value=0x53;
  packet[packetoffset++].value=0x54;
  packet[packetoffset++].value=0x45;
  packet[packetoffset++].value=0x52;
  packet[packetoffset++].value=0x20;
  packet[packetoffset++].value=0x73;
  packet[packetoffset++].value=0x69;
  packet[packetoffset++].value=0x70;
  packet[packetoffset++].value=0x3a;
  packet[packetoffset++].value=0x73;
  packet[packetoffset++].value=0x69;
  packet[packetoffset++].value=0x70;
  packet[packetoffset++].value=0x2E;
  packet[packetoffset++].value=0x62;
  packet[packetoffset++].value=0x72;
  packet[packetoffset++].value=0x6F;
  packet[packetoffset++].value=0x61;
  packet[packetoffset++].value=0x64;
  packet[packetoffset++].value=0x76;
  packet[packetoffset++].value=0x6F;
  packet[packetoffset++].value=0x69;
  packet[packetoffset++].value=0x63;
  packet[packetoffset++].value=0x65;
  packet[packetoffset++].value=0x2e;
  packet[packetoffset++].value=0x63;
  packet[packetoffset++].value=0x6F;
  packet[packetoffset++].value=0x6D;
```

```
 packet[packetoffset++].value=0x20;
 packet[packetoffset++].value=0x53;
 packet[packetoffset++].value=0x49;
 packet[packetoffset++].value=0x50;
 packet[packetoffset++].value=0x2F;
 packet[packetoffset++].value=0x32;
 packet[packetoffset++].value=0x2E;
 packet[packetoffset++].value=0x30;
 packet[packetoffset++].value=0x0d;
 packet[packetoffset++].value=0x0a;



  //CONTENT LENGTH 0
 packet[packetoffset++].value=0x43;
 packet[packetoffset++].value=0x6f;
 packet[packetoffset++].value=0x6e;
 packet[packetoffset++].value=0x74;
 packet[packetoffset++].value=0x65;
 packet[packetoffset++].value=0x6e;
 packet[packetoffset++].value=0x74;
 packet[packetoffset++].value=0x2d;
 packet[packetoffset++].value=0x4c;
 packet[packetoffset++].value=0x65;
 packet[packetoffset++].value=0x6e;
 packet[packetoffset++].value=0x67;
 packet[packetoffset++].value=0x74;
 packet[packetoffset++].value=0x68;
 packet[packetoffset++].value=0x3a;
 packet[packetoffset++].value=0x20;
 packet[packetoffset++].value=0x30;
 packet[packetoffset++].value=0x0d;
 packet[packetoffset++].value=0x0a;


//CONTACT
 packet[packetoffset++].value=0x43;
 packet[packetoffset++].value=0x6f;
 packet[packetoffset++].value=0x6e;
 packet[packetoffset++].value=0x74;
 packet[packetoffset++].value=0x61;
 packet[packetoffset++].value=0x63;
 packet[packetoffset++].value=0x74;
 packet[packetoffset++].value=0x3a;
 packet[packetoffset++].value=0x20;
 packet[packetoffset++].value=0x3c;
 packet[packetoffset++].value=0x73;
 packet[packetoffset++].value=0x69;
 packet[packetoffset++].value=0x70;
 packet[packetoffset++].value=0x3a;
 packet[packetoffset++].value=0x37;
 packet[packetoffset++].value=0x31;
 packet[packetoffset++].value=0x38;
 packet[packetoffset++].value=0x35;
 packet[packetoffset++].value=0x30;
 packet[packetoffset++].value=0x34;
 packet[packetoffset++].value=0x32;
 packet[packetoffset++].value=0x31;
 packet[packetoffset++].value=0x35;
 packet[packetoffset++].value=0x34;
 packet[packetoffset++].value=0x40;

 addIPNumerical(packet, &packetoffset);
```

```
packet[packetoffset++].value=0x3a;
packet[packetoffset++].value=0x35;
packet[packetoffset++].value=0x30;
packet[packetoffset++].value=0x36;
packet[packetoffset++].value=0x30;
packet[packetoffset++].value=0x3e;
packet[packetoffset++].value=0x0d;
packet[packetoffset++].value=0x0a;


 //SIP call-id
addSIPCallID(packet, &packetoffset);


//SIP FROM
addSIPFrom(packet, &packetoffset);


//SIP CSEQ - 1 Register
packet[packetoffset++].value=0x43;
packet[packetoffset++].value=0x53;
packet[packetoffset++].value=0x65;
packet[packetoffset++].value=0x71;
packet[packetoffset++].value=0x3a;
packet[packetoffset++].value=0x20;
packet[packetoffset++].value=0x31;
packet[packetoffset++].value=0x20;
packet[packetoffset++].value=0x52;
packet[packetoffset++].value=0x45;
packet[packetoffset++].value=0x47;
packet[packetoffset++].value=0x49;
packet[packetoffset++].value=0x53;
packet[packetoffset++].value=0x54;
packet[packetoffset++].value=0x45;
packet[packetoffset++].value=0x52;
packet[packetoffset++].value=0x0d;
packet[packetoffset++].value=0x0a;


//SIP TO
addSIPToLocal(packet, &packetoffset);









//VIA
packet[packetoffset++].value=0x56;
packet[packetoffset++].value=0x69;
packet[packetoffset++].value=0x61;
packet[packetoffset++].value=0x3a;
packet[packetoffset++].value=0x20;
packet[packetoffset++].value=0x53;
packet[packetoffset++].value=0x49;
packet[packetoffset++].value=0x50;
packet[packetoffset++].value=0x2f;
packet[packetoffset++].value=0x32;
packet[packetoffset++].value=0x2e;
```

```
packet[packetoffset++].value=0x30;
packet[packetoffset++].value=0x2f;
packet[packetoffset++].value=0x55;
packet[packetoffset++].value=0x44;
packet[packetoffset++].value=0x50;
packet[packetoffset++].value=0x20;

addIPNumerical(packet, &packetoffset);

packet[packetoffset++].value=0x3a;
packet[packetoffset++].value=0x35;
packet[packetoffset++].value=0x30;
packet[packetoffset++].value=0x36;
packet[packetoffset++].value=0x30;
packet[packetoffset++].value=0x3b;
packet[packetoffset++].value=0x62;
packet[packetoffset++].value=0x72;
packet[packetoffset++].value=0x61;
packet[packetoffset++].value=0x6e;
packet[packetoffset++].value=0x63;
packet[packetoffset++].value=0x68;
packet[packetoffset++].value=0x3d;

packet[packetoffset++].value=0x7a; //branch
packet[packetoffset++].value=0x39;
packet[packetoffset++].value=0x31;
packet[packetoffset++].value=0x68;
packet[packetoffset++].value=0x47;
packet[packetoffset++].value=0x34;
packet[packetoffset++].value=0x62;
packet[packetoffset++].value=0x4b;
packet[packetoffset++].value=0x38;
packet[packetoffset++].value=0x30;
packet[packetoffset++].value=0x33;
packet[packetoffset++].value=0x62;
packet[packetoffset++].value=0x39;
packet[packetoffset++].value=0x35;
packet[packetoffset++].value=0x61;
packet[packetoffset++].value=0x32;
packet[packetoffset++].value=0x30;
packet[packetoffset++].value=0x31;
packet[packetoffset++].value=0x33;
packet[packetoffset++].value=0x31;
packet[packetoffset++].value=0x63;
packet[packetoffset++].value=0x39;
packet[packetoffset++].value=0x62;
packet[packetoffset++].value=0x31;
packet[packetoffset++].value=0x34;
packet[packetoffset++].value=0x32;
packet[packetoffset++].value=0x35;
packet[packetoffset++].value=0x64;
packet[packetoffset++].value=0x63;
packet[packetoffset++].value=0x66;
packet[packetoffset++].value=0x38;
packet[packetoffset++].value=0x38;
packet[packetoffset++].value=0x30;
packet[packetoffset++].value=0x30;
packet[packetoffset++].value=0x30;
packet[packetoffset++].value=0x30;
packet[packetoffset++].value=0x35;
packet[packetoffset++].value=0x61;
packet[packetoffset++].value=0x38;
packet[packetoffset++].value=0x31;
```

```
  packet[packetoffset++].value=0x30;
  packet[packetoffset++].value=0x30;
  packet[packetoffset++].value=0x30;
  packet[packetoffset++].value=0x30;
  packet[packetoffset++].value=0x30;
  packet[packetoffset++].value=0x30;
  packet[packetoffset++].value=0x30;
  packet[packetoffset++].value=0x31;

  packet[packetoffset++].value=0x0d;
  packet[packetoffset++].value=0x0a;



  //authorization
  addSIPAuth(packet, &packetoffset);
  packet[packetoffset++].value=0x0d;
  packet[packetoffset++].value=0x0a;









  //END REGISTER PACKET!
  //START ACK PACKET!









    packetoffset=0;
//remove register (we don't use it...)


  //START SIP INVITE!
```

```
  addUDPIPHeader(packet, &packetoffset, 0x02, 0x46, 0x77, 0x06);

    //SIP INVITE header
  packet[packetoffset++].value=0x49;
  packet[packetoffset++].value=0x4e;
  packet[packetoffset++].value=0x56;
  packet[packetoffset++].value=0x49;
  packet[packetoffset++].value=0x54;
  packet[packetoffset++].value=0x45;
  packet[packetoffset++].value=0x20;
  packet[packetoffset++].value=0x73;
  packet[packetoffset++].value=0x69;
  packet[packetoffset++].value=0x70;
  packet[packetoffset++].value=0x3a;
  packet[packetoffset++].value=0x30 + numberToCall[0]; //start phone
#
  packet[packetoffset++].value=0x30 + numberToCall[1];
  packet[packetoffset++].value=0x30 + numberToCall[2];
  packet[packetoffset++].value=0x30 + numberToCall[3];
  packet[packetoffset++].value=0x30 + numberToCall[4];/*
  packet[packetoffset++].value=0x30 + numberToCall[5];
  packet[packetoffset++].value=0x30 + numberToCall[6];
  packet[packetoffset++].value=0x30 + numberToCall[7];
  packet[packetoffset++].value=0x30 + numberToCall[8];
  packet[packetoffset++].value=0x30 + numberToCall[9];*/
  packet[packetoffset++].value=0x40;
  packet[packetoffset++].value=0x63;
  packet[packetoffset++].value=0x6f;
  packet[packetoffset++].value=0x6c;
  packet[packetoffset++].value=0x75;
  packet[packetoffset++].value=0x6d;
  packet[packetoffset++].value=0x62;
  packet[packetoffset++].value=0x69;
  packet[packetoffset++].value=0x61;
  packet[packetoffset++].value=0x2e;
  packet[packetoffset++].value=0x65;
  packet[packetoffset++].value=0x64;
  packet[packetoffset++].value=0x75;
/*  packet[packetoffset++].value=0x20;
  packet[packetoffset++].value=0x63;
  packet[packetoffset++].value=0x65;
```

```
packet[packetoffset++].value=0x2e;
packet[packetoffset++].value=0x63;
packet[packetoffset++].value=0x6f;
packet[packetoffset++].value=0x6d;*/

packet[packetoffset++].value=0x20;
packet[packetoffset++].value=0x53;
packet[packetoffset++].value=0x49;
packet[packetoffset++].value=0x50;
packet[packetoffset++].value=0x2f;
packet[packetoffset++].value=0x32;
packet[packetoffset++].value=0x2e;
packet[packetoffset++].value=0x30;
packet[packetoffset++].value=0x0d;
packet[packetoffset++].value=0x0a;


//VIA
addSIPVia(packet, &packetoffset); /*
packet[packetoffset++].value=0x56;
packet[packetoffset++].value=0x69;
packet[packetoffset++].value=0x61;
packet[packetoffset++].value=0x3a;
packet[packetoffset++].value=0x20;
packet[packetoffset++].value=0x53;
packet[packetoffset++].value=0x49;
packet[packetoffset++].value=0x50;
packet[packetoffset++].value=0x2f;
packet[packetoffset++].value=0x32;
packet[packetoffset++].value=0x2e;
packet[packetoffset++].value=0x30;
packet[packetoffset++].value=0x2f;
packet[packetoffset++].value=0x55;
packet[packetoffset++].value=0x44;
packet[packetoffset++].value=0x50;
packet[packetoffset++].value=0x20;

addIPNumerical(packet, &packetoffset);

packet[packetoffset++].value=0x3a;
packet[packetoffset++].value=0x35;
packet[packetoffset++].value=0x30;
packet[packetoffset++].value=0x36;
packet[packetoffset++].value=0x30;
packet[packetoffset++].value=0x3b;

packet[packetoffset++].value=0x72;
packet[packetoffset++].value=0x70;
packet[packetoffset++].value=0x6f;
packet[packetoffset++].value=0x72;
packet[packetoffset++].value=0x74;/*
packet[packetoffset++].value=0x33;
packet[packetoffset++].value=0x35;
packet[packetoffset++].value=0x30;
packet[packetoffset++].value=0x36;
packet[packetoffset++].value=0x30;*
packet[packetoffset++].value=0x3b;

packet[packetoffset++].value=0x62;
packet[packetoffset++].value=0x72;
packet[packetoffset++].value=0x61;
packet[packetoffset++].value=0x6e;
packet[packetoffset++].value=0x63;
```

```
 packet[packetoffset++].value=0x68;
 packet[packetoffset++].value=0x3d;

 packet[packetoffset++].value=0x7a; //branch
 packet[packetoffset++].value=0x39;
 packet[packetoffset++].value=0x31;
 packet[packetoffset++].value=0x68;
 packet[packetoffset++].value=0x47;
 packet[packetoffset++].value=0x34;
 packet[packetoffset++].value=0x62;
 packet[packetoffset++].value=0x4b;
 packet[packetoffset++].value=0x38;
 packet[packetoffset++].value=0x30;
 packet[packetoffset++].value=0x33;
 packet[packetoffset++].value=0x62;
 packet[packetoffset++].value=0x39;
 packet[packetoffset++].value=0x35;
 packet[packetoffset++].value=0x61;
 packet[packetoffset++].value=0x32;
 packet[packetoffset++].value=0x30;
 packet[packetoffset++].value=0x31;
 packet[packetoffset++].value=0x33;
 packet[packetoffset++].value=0x31;
 packet[packetoffset++].value=0x63;
 packet[packetoffset++].value=0x39;
 packet[packetoffset++].value=0x62;
 packet[packetoffset++].value=0x31;
 packet[packetoffset++].value=0x34;
 packet[packetoffset++].value=0x32;
 packet[packetoffset++].value=0x35;
 packet[packetoffset++].value=0x64;
 packet[packetoffset++].value=0x63;
 packet[packetoffset++].value=0x66;
 packet[packetoffset++].value=0x38;
 packet[packetoffset++].value=0x38;
 packet[packetoffset++].value=0x30;
 packet[packetoffset++].value=0x31;
 packet[packetoffset++].value=0x30;
 packet[packetoffset++].value=0x30;
 packet[packetoffset++].value=0x35;
 packet[packetoffset++].value=0x61;
 packet[packetoffset++].value=0x38;
 packet[packetoffset++].value=0x31;
 packet[packetoffset++].value=0x30;
 packet[packetoffset++].value=0x30;
 packet[packetoffset++].value=0x30;
 packet[packetoffset++].value=0x30;
 packet[packetoffset++].value=0x30;
 packet[packetoffset++].value=0x30;
 packet[packetoffset++].value=0x30;
 packet[packetoffset++].value=0x31;

 packet[packetoffset++].value=0x0d;
 packet[packetoffset++].value=0x0a; */

 //SIP FROM
 addSIPFrom(packet, &packetoffset);

 //SIP TO
 addSIPToRemote(packet, &packetoffset);


//CONTACT
```

```
/*
  packet[packetoffset++].value=0x43;
  packet[packetoffset++].value=0x6f;
  packet[packetoffset++].value=0x6e;
  packet[packetoffset++].value=0x74;
  packet[packetoffset++].value=0x61;
  packet[packetoffset++].value=0x63;
  packet[packetoffset++].value=0x74;
  packet[packetoffset++].value=0x3a;
  packet[packetoffset++].value=0x20;
  packet[packetoffset++].value=0x3c;
  packet[packetoffset++].value=0x73;
  packet[packetoffset++].value=0x69;
  packet[packetoffset++].value=0x70;
  packet[packetoffset++].value=0x3a;
  packet[packetoffset++].value=0x33;
  packet[packetoffset++].value=0x35;
  packet[packetoffset++].value=0x33;
  packet[packetoffset++].value=0x36;
  packet[packetoffset++].value=0x30;
  packet[packetoffset++].value=0x40;

  addIPNumerical(packet, &packetoffset);

  packet[packetoffset++].value=0x3a;
  packet[packetoffset++].value=0x35;
  packet[packetoffset++].value=0x30;
  packet[packetoffset++].value=0x36;
  packet[packetoffset++].value=0x30; /*

  packet[packetoffset++].value=0x3e;
  packet[packetoffset++].value=0x2e;
  packet[packetoffset++].value=0x31;
  packet[packetoffset++].value=0x36;
  packet[packetoffset++].value=0x32;
  packet[packetoffset++].value=0x3a;
  packet[packetoffset++].value=0x35;
  packet[packetoffset++].value=0x30;
  packet[packetoffset++].value=0x36;
  packet[packetoffset++].value=0x30;*
  packet[packetoffset++].value=0x3e;
  packet[packetoffset++].value=0x0d;
  packet[packetoffset++].value=0x0a;*/

  addSIPContact(packet, &packetoffset);


   //SIP call-id
  addSIPCallID(packet, &packetoffset);

  //CSEQ : 7472 INVITE
  packet[packetoffset++].value=0x43;
  packet[packetoffset++].value=0x53;
  packet[packetoffset++].value=0x65;
  packet[packetoffset++].value=0x71;
  packet[packetoffset++].value=0x3a;
  packet[packetoffset++].value=0x20;
  packet[packetoffset++].value=0x37;
  packet[packetoffset++].value=0x34;
  packet[packetoffset++].value=0x37;
  packet[packetoffset++].value=0x32;
  packet[packetoffset++].value=0x20;
  packet[packetoffset++].value=0x49;
```

```
  packet[packetoffset++].value=0x4e;
  packet[packetoffset++].value=0x56;
  packet[packetoffset++].value=0x49;
  packet[packetoffset++].value=0x54;
  packet[packetoffset++].value=0x45;
  packet[packetoffset++].value=0x0d;
  packet[packetoffset++].value=0x0a;



  //CONTENT TYPE = SDP
  packet[packetoffset++].value=0x43;
  packet[packetoffset++].value=0x6f;
  packet[packetoffset++].value=0x6e;
  packet[packetoffset++].value=0x74;
  packet[packetoffset++].value=0x65;
  packet[packetoffset++].value=0x6e;
  packet[packetoffset++].value=0x74;
  packet[packetoffset++].value=0x2d;
  packet[packetoffset++].value=0x54;
  packet[packetoffset++].value=0x79;
  packet[packetoffset++].value=0x70;
  packet[packetoffset++].value=0x65;
  packet[packetoffset++].value=0x3a;
  packet[packetoffset++].value=0x20;
  packet[packetoffset++].value=0x61;
  packet[packetoffset++].value=0x70;
  packet[packetoffset++].value=0x70;
  packet[packetoffset++].value=0x6c;
  packet[packetoffset++].value=0x69;
  packet[packetoffset++].value=0x63;
  packet[packetoffset++].value=0x61;
  packet[packetoffset++].value=0x74;
  packet[packetoffset++].value=0x69;
  packet[packetoffset++].value=0x6f;
  packet[packetoffset++].value=0x6e;
  packet[packetoffset++].value=0x2f;
  packet[packetoffset++].value=0x73;
  packet[packetoffset++].value=0x64;
  packet[packetoffset++].value=0x70;
  packet[packetoffset++].value=0x0d;
  packet[packetoffset++].value=0x0a;

/*
  //User Agent
  packet[packetoffset++].value=0x55;
  packet[packetoffset++].value=0x73;
  packet[packetoffset++].value=0x65;
  packet[packetoffset++].value=0x72;
  packet[packetoffset++].value=0x2d;
  packet[packetoffset++].value=0x41;
  packet[packetoffset++].value=0x67;
  packet[packetoffset++].value=0x65;
  packet[packetoffset++].value=0x6e;
  packet[packetoffset++].value=0x74;
  packet[packetoffset++].value=0x3a;
  packet[packetoffset++].value=0x20;
  packet[packetoffset++].value=0x58;
  packet[packetoffset++].value=0x2d;
  packet[packetoffset++].value=0x4c;
  packet[packetoffset++].value=0x69;
  packet[packetoffset++].value=0x74;
```

```
  packet[packetoffset++].value=0x65;
  packet[packetoffset++].value=0x20;
  packet[packetoffset++].value=0x72;
  packet[packetoffset++].value=0x65;
  packet[packetoffset++].value=0x6c;
  packet[packetoffset++].value=0x65;
  packet[packetoffset++].value=0x61;
  packet[packetoffset++].value=0x73;
  packet[packetoffset++].value=0x65;
  packet[packetoffset++].value=0x20;
  packet[packetoffset++].value=0x31;
  packet[packetoffset++].value=0x31;
  packet[packetoffset++].value=0x30;
  packet[packetoffset++].value=0x33;
  packet[packetoffset++].value=0x6d;
  packet[packetoffset++].value=0x0d;
  packet[packetoffset++].value=0x0a;

*/

  //CONTENT LENGTH 153
  packet[packetoffset++].value=0x43;
  packet[packetoffset++].value=0x6f;
  packet[packetoffset++].value=0x6e;
  packet[packetoffset++].value=0x74;
  packet[packetoffset++].value=0x65;
  packet[packetoffset++].value=0x6e;
  packet[packetoffset++].value=0x74;
  packet[packetoffset++].value=0x2d;
  packet[packetoffset++].value=0x4c;
  packet[packetoffset++].value=0x65;
  packet[packetoffset++].value=0x6e;
  packet[packetoffset++].value=0x67;
  packet[packetoffset++].value=0x74;
  packet[packetoffset++].value=0x68;
  packet[packetoffset++].value=0x3a;
  packet[packetoffset++].value=0x20;
  packet[packetoffset++].value=0x31;
  packet[packetoffset++].value=0x35;
  packet[packetoffset++].value=0x33;
  packet[packetoffset++].value=0x0d;
  packet[packetoffset++].value=0x0a;


  //authorization
//  addSIPAuth(packet, &packetoffset);




  packet[packetoffset++].value=0x0d;
  packet[packetoffset++].value=0x0a;


  //SESSION DESCRIPTION PROTOCOL AS PART OF INVITE!
  packet[packetoffset++].value=0x76; //sdp version = 0
  packet[packetoffset++].value=0x3d;
  packet[packetoffset++].value=0x30;
  packet[packetoffset++].value=0x0d;
  packet[packetoffset++].value=0x0a;
```

```
packet[packetoffset++].value=0x6f; //session #1
packet[packetoffset++].value=0x3d;
packet[packetoffset++].value=0x2d;
packet[packetoffset++].value=0x20; // to
packet[packetoffset++].value=0x31;
packet[packetoffset++].value=0x20;
packet[packetoffset++].value=0x31; // #1
packet[packetoffset++].value=0x20;

packet[packetoffset++].value=0x49; //in
packet[packetoffset++].value=0x4e;
packet[packetoffset++].value=0x20;

packet[packetoffset++].value=0x49; //IP4
packet[packetoffset++].value=0x50;
packet[packetoffset++].value=0x34;
packet[packetoffset++].value=0x20;

addIPNumerical(packet, &packetoffset);


packet[packetoffset++].value=0x0d;
packet[packetoffset++].value=0x0a;

packet[packetoffset++].value=0x73; //no session name
packet[packetoffset++].value=0x3d;
packet[packetoffset++].value=0x2d;
packet[packetoffset++].value=0x0d;
packet[packetoffset++].value=0x0a;


packet[packetoffset++].value=0x63; //connection info
packet[packetoffset++].value=0x3d;
packet[packetoffset++].value=0x49; //in
packet[packetoffset++].value=0x4e;
packet[packetoffset++].value=0x20;

packet[packetoffset++].value=0x49; //IP4
packet[packetoffset++].value=0x50;
packet[packetoffset++].value=0x34;
packet[packetoffset++].value=0x20;

addIPNumerical(packet, &packetoffset);

packet[packetoffset++].value=0x0d;
packet[packetoffset++].value=0x0a;

packet[packetoffset++].value=0x74; //time
packet[packetoffset++].value=0x3d;
packet[packetoffset++].value=0x30;
packet[packetoffset++].value=0x20;
packet[packetoffset++].value=0x30;
packet[packetoffset++].value=0x0d;
packet[packetoffset++].value=0x0a;


packet[packetoffset++].value=0x6d; //media type=audio
packet[packetoffset++].value=0x3d;
packet[packetoffset++].value=0x61;
packet[packetoffset++].value=0x75;
packet[packetoffset++].value=0x64;
packet[packetoffset++].value=0x69;
packet[packetoffset++].value=0x6f;
```

```
  packet[packetoffset++].value=0x20;

  packet[packetoffset++].value=0x31; //port = 16384
  packet[packetoffset++].value=0x36;
  packet[packetoffset++].value=0x33;
  packet[packetoffset++].value=0x38;
  packet[packetoffset++].value=0x34;
  packet[packetoffset++].value=0x20;

  packet[packetoffset++].value=0x52; //RTP/AVP
  packet[packetoffset++].value=0x54;
  packet[packetoffset++].value=0x50;
  packet[packetoffset++].value=0x2f;
  packet[packetoffset++].value=0x41;
  packet[packetoffset++].value=0x56;
  packet[packetoffset++].value=0x50;

  packet[packetoffset++].value=0x20;
  packet[packetoffset++].value=0x30; //g.711u
  packet[packetoffset++].value=0x0d;
  packet[packetoffset++].value=0x0a;



  packet[packetoffset++].value=0x61; //rtpmap
  packet[packetoffset++].value=0x3d;
  packet[packetoffset++].value=0x72;
  packet[packetoffset++].value=0x74;
  packet[packetoffset++].value=0x70;
  packet[packetoffset++].value=0x6d;
  packet[packetoffset++].value=0x61;
  packet[packetoffset++].value=0x70;
  packet[packetoffset++].value=0x3a;
  packet[packetoffset++].value=0x31;
  packet[packetoffset++].value=0x30;
  packet[packetoffset++].value=0x31;
  packet[packetoffset++].value=0x20;
  packet[packetoffset++].value=0x74;
  packet[packetoffset++].value=0x65;
  packet[packetoffset++].value=0x6c;
  packet[packetoffset++].value=0x65;
  packet[packetoffset++].value=0x70;
  packet[packetoffset++].value=0x68;
  packet[packetoffset++].value=0x6f;
  packet[packetoffset++].value=0x6e;
  packet[packetoffset++].value=0x65;
  packet[packetoffset++].value=0x2d;
  packet[packetoffset++].value=0x65;
  packet[packetoffset++].value=0x76;
  packet[packetoffset++].value=0x65;
  packet[packetoffset++].value=0x6e;
  packet[packetoffset++].value=0x74;
  packet[packetoffset++].value=0x2f;
  packet[packetoffset++].value=0x38;
  packet[packetoffset++].value=0x30;
  packet[packetoffset++].value=0x30;
  packet[packetoffset++].value=0x30;
  packet[packetoffset++].value=0x0d;
  packet[packetoffset++].value=0x0a;

/*
  packet[packetoffset++].value=0x61; //rtpmap
  packet[packetoffset++].value=0x3d;
```

```
  packet[packetoffset++].value=0x72;
  packet[packetoffset++].value=0x74;
  packet[packetoffset++].value=0x70;
  packet[packetoffset++].value=0x6d;
  packet[packetoffset++].value=0x61;
  packet[packetoffset++].value=0x70;
  packet[packetoffset++].value=0x3a;
  packet[packetoffset++].value=0x30;
  packet[packetoffset++].value=0x20;
  packet[packetoffset++].value=0x70;
  packet[packetoffset++].value=0x63;
  packet[packetoffset++].value=0x6d;
  packet[packetoffset++].value=0x75;
  packet[packetoffset++].value=0x2f;
  packet[packetoffset++].value=0x38;
  packet[packetoffset++].value=0x30;
  packet[packetoffset++].value=0x30;
  packet[packetoffset++].value=0x30;
  packet[packetoffset++].value=0x0d;
  packet[packetoffset++].value=0x0a; */

  packet[packetoffset++].value=0x61; //fmtp
  packet[packetoffset++].value=0x3d;
  packet[packetoffset++].value=0x66;
  packet[packetoffset++].value=0x6d;
  packet[packetoffset++].value=0x74;
  packet[packetoffset++].value=0x70;
  packet[packetoffset++].value=0x3a;
  packet[packetoffset++].value=0x31;
  packet[packetoffset++].value=0x30;
  packet[packetoffset++].value=0x31;
  packet[packetoffset++].value=0x20;
  packet[packetoffset++].value=0x30;
  packet[packetoffset++].value=0x2d;
  packet[packetoffset++].value=0x31;
  packet[packetoffset++].value=0x31;
  packet[packetoffset++].value=0x2c;
  packet[packetoffset++].value=0x31;
  packet[packetoffset++].value=0x36;
  packet[packetoffset++].value=0x0d;
  packet[packetoffset++].value=0x0a;




/* GRATUITOUS ARP QUERY */
  packet[packetoffset++].value=0xff;
  packet[packetoffset++].value=0xff;
  packet[packetoffset++].value=0xff;
  packet[packetoffset++].value=0xff;
  packet[packetoffset++].value=0xff;
  packet[packetoffset++].value=0xff;

  //MAC
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x08;
  packet[packetoffset++].value=0x02;
  packet[packetoffset++].value=0x62;
  packet[packetoffset++].value=0x30;
  packet[packetoffset++].value=0x5d;
```

```
  packet[packetoffset++].value=0x08;
  packet[packetoffset++].value=0x06;


  //ARP data
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x01;
  packet[packetoffset++].value=0x08;
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x06;
  packet[packetoffset++].value=0x04;
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x01;

  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x08;
  packet[packetoffset++].value=0x02;
  packet[packetoffset++].value=0x62;
  packet[packetoffset++].value=0x30;
  packet[packetoffset++].value=0x5d;

  //ip i want
  addIPHex(packet, &packetoffset);

  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x00;

  addIPHex(packet, &packetoffset);

  //pad - trailer
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x00;

  //printf("start: %d", packetoffset);


/* SIP ACK ------------------------*/


  addUDPIPHeader(packet, &packetoffset, 0x01, 0x95, 0x77, 0xb5);

    //SIP ACK header
```

```
  packet[packetoffset++].value=0x41;
  packet[packetoffset++].value=0x43;
  packet[packetoffset++].value=0x4b;
  packet[packetoffset++].value=0x20;
  packet[packetoffset++].value=0x73;
  packet[packetoffset++].value=0x69;
  packet[packetoffset++].value=0x70;
  packet[packetoffset++].value=0x3a;
  packet[packetoffset++].value=0x30 + numberToCall[0]; //start phone
#
  packet[packetoffset++].value=0x30 + numberToCall[1];
  packet[packetoffset++].value=0x30 + numberToCall[2];
  packet[packetoffset++].value=0x30 + numberToCall[3];
  packet[packetoffset++].value=0x30 + numberToCall[4]; /*
  packet[packetoffset++].value=0x30 + numberToCall[5];
  packet[packetoffset++].value=0x30 + numberToCall[6];
  packet[packetoffset++].value=0x30 + numberToCall[7];
  packet[packetoffset++].value=0x30 + numberToCall[8];
  packet[packetoffset++].value=0x30 + numberToCall[9]; */
  packet[packetoffset++].value=0x40;

  packet[packetoffset++].value=0x63;
  packet[packetoffset++].value=0x6f;
  packet[packetoffset++].value=0x6c;
  packet[packetoffset++].value=0x75;
  packet[packetoffset++].value=0x6d;
  packet[packetoffset++].value=0x62;
  packet[packetoffset++].value=0x69;
  packet[packetoffset++].value=0x61;
  packet[packetoffset++].value=0x2e;
  packet[packetoffset++].value=0x65;
  packet[packetoffset++].value=0x64;
  packet[packetoffset++].value=0x75;

  packet[packetoffset++].value=0x20;
  packet[packetoffset++].value=0x53;
  packet[packetoffset++].value=0x49;
  packet[packetoffset++].value=0x50;
  packet[packetoffset++].value=0x2f;
  packet[packetoffset++].value=0x32;
  packet[packetoffset++].value=0x2e;
  packet[packetoffset++].value=0x30;




  packet[packetoffset++].value=0x0d;
  packet[packetoffset++].value=0x0a;


  addSIPVia(packet, &packetoffset);

    //SIP FROM
  addSIPFrom(packet, &packetoffset);

  //SIP TO
  addSIPToRemoteWithTag(packet, &packetoffset);
  //printf("tag: %d\n", packetoffset);

   addSIPContact(packet, &packetoffset);

   //SIP call-id
  addSIPCallID(packet, &packetoffset);
```

```
  //CSEQ : 7472 ACK
  packet[packetoffset++].value=0x43;
  packet[packetoffset++].value=0x53;
  packet[packetoffset++].value=0x65;
  packet[packetoffset++].value=0x71;
  packet[packetoffset++].value=0x3a;
  packet[packetoffset++].value=0x20;
  packet[packetoffset++].value=0x37;
  packet[packetoffset++].value=0x34;
  packet[packetoffset++].value=0x37;
  packet[packetoffset++].value=0x32;
  packet[packetoffset++].value=0x20;
  packet[packetoffset++].value=0x41;
  packet[packetoffset++].value=0x43;
  packet[packetoffset++].value=0x4b;
  packet[packetoffset++].value=0x0d;
  packet[packetoffset++].value=0x0a;


    //CONTENT LENGTH 0
  packet[packetoffset++].value=0x43;
  packet[packetoffset++].value=0x6f;
  packet[packetoffset++].value=0x6e;
  packet[packetoffset++].value=0x74;
  packet[packetoffset++].value=0x65;
  packet[packetoffset++].value=0x6e;
  packet[packetoffset++].value=0x74;
  packet[packetoffset++].value=0x2d;
  packet[packetoffset++].value=0x4c;
  packet[packetoffset++].value=0x65;
  packet[packetoffset++].value=0x6e;
  packet[packetoffset++].value=0x67;
  packet[packetoffset++].value=0x74;
  packet[packetoffset++].value=0x68;
  packet[packetoffset++].value=0x3a;
  packet[packetoffset++].value=0x20;
  packet[packetoffset++].value=0x30;
  packet[packetoffset++].value=0x0d;
  packet[packetoffset++].value=0x0a;

  //authorization
//  addSIPAuth(packet, &packetoffset);



  packet[packetoffset++].value=0x0d;
  packet[packetoffset++].value=0x0a;

  //END SIP ACK!






/* RTP RTP RTP RTP RTP RTP RTP RTP RTP RTP RTP RTP RTP RTP RTP RTP
*/

  //START RTP Framework
  //printf("Start RTP: %d", packetoffset);

  //  addUDPIPHeader(packet, &packetoffset, 0x00, 0xc8, 0x63, 0xcc);
```

```
  //alter dest ip to columbia's rtp gateway for sip: 128.59.59.242
  //9+10 or 10 + 11 bytes back  hmm



  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0xd0;
  packet[packetoffset++].value=0x06;
  packet[packetoffset++].value=0x26; //26 vs 24
  packet[packetoffset++].value=0x9c; //9c vs c8
  packet[packetoffset++].value=0x0a;
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x08;
  packet[packetoffset++].value=0x02;

  packet[packetoffset++].value=0x62;
  packet[packetoffset++].value=0x30;
  packet[packetoffset++].value=0x5d;
  packet[packetoffset++].value=0x08;
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x45;
  packet[packetoffset++].value=0xb8; //expedited = b8.. on for rtp,
not for sip



  packet[packetoffset++].value=0; //total packet length
  packet[packetoffset++].value=0xc8;
  packet[packetoffset++].value=0x04;
  packet[packetoffset++].value=0x41;
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x80;
  packet[packetoffset++].value=0x11;
  packet[packetoffset++].value=0x63; //checksum
  packet[packetoffset++].value=0x59;

  addIPHex(packet, &packetoffset);

  packet[packetoffset++].value=0x80;
  packet[packetoffset++].value=0x3b;
  packet[packetoffset++].value=0x3b;
  packet[packetoffset++].value=0xf2;

  packet[packetoffset++].value=0x40;
  packet[packetoffset++].value=0x00;
  //dest port
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x00;
  packet[packetoffset++].value=0x00; //udp length
  packet[packetoffset++].value=0xb4;
  packet[packetoffset++].value=0x00; //50, 24 //disable udp checksum
  packet[packetoffset++].value=0x00;


  packet[packetoffset++].value = 0x80;
  packet[packetoffset++].value = 0x00;
```

```
//  fprintf(fp,"%c%c", 0x00, 0x00);
  for (i = 0; i <= packetoffset; i++) {

//    printf("%d\n",i);
    fprintf(fp,"%c%c",packet[i], 0x00);
  }


  //printf("ends: %d", packetoffset);

  fclose(fp);

}
```

Makefile

```
# Makefile for CSEE 4840

SYSTEM = system

MICROBLAZE_OBJS = \
            c_source_files/main.o

REGISTER_OBJS = \
            c_source_files/packet_register.o

LIBRARIES = mymicroblaze/lib/libxil.a

ELF_FILE = $(SYSTEM).elf

NETLIST = implementation/$(SYSTEM).ngc

# Bitstreams for the FPGA

FPGA_BITFILE = implementation/$(SYSTEM).bit
MERGED_BITFILE = implementation/download.bit

# Files to be downloaded to the SRAM

SRAM_BINFILE = implementation/sram.bin
SRAM_HEXFILE = implementation/sram.hex

MHSFILE = $(SYSTEM).mhs
MSSFILE = $(SYSTEM).mss

FPGA_ARCH = spartan2e
DEVICE = xc2s300epq208-6

LANGUAGE = vhdl
PLATGEN_OPTIONS = -p $(FPGA_ARCH) -lang $(LANGUAGE)
LIBGEN_OPTIONS = -p $(FPGA_ARCH) $(MICROBLAZE_LIBG_OPT)

# Paths for programs

XILINX = /usr/cad/xilinx/ise6.2i
ISEBINDIR = $(XILINX)/bin/lin
ISEENVCMDS = LD_LIBRARY_PATH=$(ISEBINDIR) XILINX=$(XILINX) PATH=$
(ISEBINDIR):$(PATH)

XILINX_EDK = /usr/cad/xilinx/edk6.2i
EDKBINDIR = $(XILINX_EDK)/bin/lin
```

```
EDKENVCMDS = LD_LIBRARY_PATH=$(ISEBINDIR):$(EDKBINDIR) XILINX=$
(XILINX) XILINX_EDK=$(XILINX_EDK) PATH=$(ISEBINDIR):$(EDKBINDIR):
$(PATH)

MICROBLAZE = $(XILINX_EDK)/gnu/microblaze/lin
MBBINDIR = $(MICROBLAZE)/bin
XESSBINDIR = /usr/cad/xess/bin

# Executables

PLATGEN = $(EDKENVCMDS) $(EDKBINDIR)/platgen
LIBGEN = $(EDKENVCMDS) $(EDKBINDIR)/libgen

XST = $(ISEENVCMDS) $(ISEBINDIR)/xst
XFLOW = $(ISEENVCMDS) $(ISEBINDIR)/xflow
BITGEN = $(ISEENVCMDS) $(ISEBINDIR)/bitgen
DATA2MEM = $(ISEENVCMDS) $(ISEBINDIR)/data2mem
XSLOAD = $(XESSBINDIR)/xsload
XESS_BOARD = XSB-300E

MICROBLAZE_CC = $(MBBINDIR)/mb-gcc
MICROBLAZE_CC_SIZE = $(MBBINDIR)/mb-size
MICROBLAZE_OBJCOPY = $(MBBINDIR)/mb-objcopy

# External Targets

all :
            @echo "Makefile to build a Microprocessor system :"
            @echo "Run make with any of the following targets"
            @echo "  make libs     : Configures the sw libraries
for this system"
            @echo "  make program  : Compiles the program sources
for all the processor instances"
            @echo "  make netlist  : Generates the netlist for this
system ($(SYSTEM))"
            @echo "  make bits     : Runs Implementation tools to
generate the bitstream"
            @echo "  make init_bram: Initializes bitstream with
BRAM data"
            @echo "  make download : Downloads the bitstream onto
the board"
            @echo "  make netlistclean: Deletes netlist"
            @echo "  make hwclean  : Deletes implementation dir"
            @echo "  make libsclean: Deletes sw libraries"
            @echo "  make programclean: Deletes compiled ELF files"
            @echo "  make clean    : Deletes all generated
files/directories"
            @echo " "
            @echo "  make <target> : (Default)"
            @echo "      Creates a Microprocessor system using
default initializations"
            @echo "      specified for each processor in MSS file"


bits : $(FPGA_BITFILE)

netlist : $(NETLIST)

libs : $(LIBRARIES)

program : $(ELF_FILE)

init_bram : $(MERGED_BITFILE)
```

```
clean : hwclean libsclean programclean
            rm -f bram_init.sh platgen.log platgen.opt libgen.log
            rm -f _impact.cmd xflow.his

hwclean : netlistclean
            rm -rf implementation synthesis xst hdl
            rm -rf xst.srp $(SYSTEM)_xst.srp

netlistclean :
            rm -f $(FPGA_BITFILE) $(MERGED_BITFILE) \
              $(NETLIST) implementation/$(SYSTEM)_bd.bmm

libsclean :
            rm -rf mymicroblaze/lib

programclean :
            rm -f $(ELF_FILE) $(SRAM_BITFILE) $(SRAM_HEXFILE)

#
# Software rules
#

MICROBLAZE_MODE = executable

# Assemble software libraries from the .mss and .mhs files

$(LIBRARIES) : $(MHSFILE) $(MSSFILE)
            $(LIBGEN) $(LIBGEN_OPTIONS) $(MSSFILE)

# Compilation

MICROBLAZE_CC_CFLAGS =
MICROBLAZE_CC_OPT = -O3 #-mxl-gp-opt
MICROBLAZE_CC_DEBUG_FLAG =#  -gstabs
MICROBLAZE_INCLUDES = -I./mymicroblaze/include/ # -I
MICROBLAZE_CFLAGS = \
            $(MICROBLAZE_CC_CFLAGS)\
            -mxl-barrel-shift \
            $(MICROBLAZE_CC_OPT) \
            $(MICROBLAZE_CC_DEBUG_FLAG) \
            $(MICROBLAZE_INCLUDES)

$(MICROBLAZE_OBJS) : %.o : %.c
            PATH=$(MBBINDIR) $(MICROBLAZE_CC) $(MICROBLAZE_CFLAGS)
-c $< -o $@

# Linking

# Uncomment the following to make linker print locations for
everything
# MICROBLAZE_LD_FLAGS = -Wl,-M
#MICROBLAZE_LINKER_SCRIPT = -Wl,-T -Wl,mylinkscript
MICROBLAZE_LIBPATH = -L./mymicroblaze/lib/
MICROBLAZE_CC_START_ADDR_FLAG= -Wl,-defsym -Wl,
_TEXT_START_ADDR=0x00000000
MICROBLAZE_CC_STACK_SIZE_FLAG=  -Wl,-defsym -Wl,_STACK_SIZE=0x200
MICROBLAZE_LFLAGS = \
             -xl-mode-$(MICROBLAZE_MODE) \
            $(MICROBLAZE_LD_FLAGS) \
            $(MICROBLAZE_LINKER_SCRIPT) \
            $(MICROBLAZE_LIBPATH) \
            $(MICROBLAZE_CC_START_ADDR_FLAG) \
```

```
            $(MICROBLAZE_CC_STACK_SIZE_FLAG)

$(ELF_FILE) :  $(LIBRARIES) $(MICROBLAZE_OBJS)
            PATH=$(MBBINDIR) $(MICROBLAZE_CC) $(MICROBLAZE_LFLAGS)
\
                 $(MICROBLAZE_OBJS) -o $(ELF_FILE)
            $(MICROBLAZE_CC_SIZE) $(ELF_FILE)

#
# Hardware rules
#

# Hardware compilation : optimize the netlist, place and route

$(FPGA_BITFILE) : $(NETLIST) \
                  etc/fast_runtime.opt etc/bitgen.ut data/$(SYSTEM).
ucf
            cp -f etc/bitgen.ut implementation/
            cp -f etc/fast_runtime.opt implementation/
            cp -f data/$(SYSTEM).ucf implementation/$(SYSTEM).ucf
            $(XFLOW) -wd implementation -p $(DEVICE) -implement
fast_runtime.opt \
                 $(SYSTEM).ngc
            cd implementation; $(BITGEN) -f bitgen.ut $(SYSTEM)

# Hardware assembly: Create the netlist from the .mhs file

$(NETLIST) : $(MHSFILE)
            $(PLATGEN) $(PLATGEN_OPTIONS) -st xst $(MHSFILE)
            $(XST) -ifn synthesis/$(SYSTEM)_xst.scr
#           perl synth_modules.pl < synthesis/xst.scr > xst.scr
#           $(XST) -ifn xst.scr
#           rm -r xst xst.scr
#           $(XST) -ifn synthesis/$(SYSTEM).scr

#
# Downloading
#

# Add software code to the FPGA bitfile

$(MERGED_BITFILE) : $(FPGA_BITFILE) $(ELF_FILE)
            $(DATA2MEM) -bm implementation/$(SYSTEM)_bd \
              -bt implementation/$(SYSTEM) \
              -bd $(ELF_FILE) tag bram -o b $(MERGED_BITFILE)

# Create a .hex file with data for the SRAM

$(SRAM_HEXFILE) : $(ELF_FILE)
            $(MICROBLAZE_OBJCOPY) \
                 -j .sram_text -j .sdata2 -j .sdata -j .rodata -j .
data \
           -O binary $(ELF_FILE) $(SRAM_BINFILE)
            ./bin2hex  -a 60000 < $(SRAM_BINFILE) > $(SRAM_HEXFILE)

# Download the files to the target board

download-sram : $(MERGED_BITFILE) $(SRAM_HEXFILE)
            $(XSLOAD) -ram -b $(XESS_BOARD) $(SRAM_HEXFILE)
            $(XSLOAD) -fpga -b $(XESS_BOARD) $(MERGED_BITFILE)

packets:
            gcc packets.c
```

```
            ./a.out
            ./bin2hex -a 0 < sram.bin > sram.hex
            /usr/cad/xess/bin/xsload -ram sram.hex -board xsb-300e

download : $(MERGED_BITFILE)
            $(XSLOAD) -fpga -b $(XESS_BOARD) $(MERGED_BITFILE)
```

Packet-Sniffed Data Output

#1 - Gratuitous ARP

Notes: See section on Gratuitous ARP; source and target IP are identical, packet is broadcasted, and packet includes MAC address

No. Time Source Destination Protocol Info

350 5.730819 dyn-128-59-149-106.dyn.columbia.edu Broadcast ARP Who has 128.59.149.106? Gratuitous ARP

Frame 350 (60 bytes on wire, 60 bytes captured)

Arrival Time: May 10, 2005 11:30:13.143614000

Time delta from previous packet: 0.211875000 seconds

Time since reference or first frame: 5.730819000 seconds

Frame Number: 350

Packet Length: 60 bytes

Capture Length: 60 bytes

Ethernet II, Src: 00:08:02:62:30:5d, Dst: ff:ff:ff:ff:ff:ff

Destination: ff:ff:ff:ff:ff:ff (Broadcast)

Source: 00:08:02:62:30:5d (dyn-128-59-149-106.dyn.columbia.edu)

Type: ARP (0x0806)

Trailer: 000000000000000000000000000000000000

Address Resolution Protocol (request/gratuitous ARP)

Hardware type: Ethernet (0x0001)

Protocol type: IP (0x0800)

Hardware size: 6

Protocol size: 4

Opcode: request (0x0001)

Sender MAC address: 00:08:02:62:30:5d (dyn-128-59-149-106.dyn.columbia.edu)

Sender IP address: 128.59.149.106 (128.59.149.106)

Target MAC address: 00:00:00:00:00:00 (00:00:00_00:00:00)

Target IP address: 128.59.149.106 (128.59.149.106)

#2 – SIP/SDP Invite

Notes: Straightforward packet, address (phone to call) modified using C preloader program; SDP makes request for G.711mu law at 8kHz using RTP at desired port

No. Time Source Destination Protocol Info

491 6.466699 128.59.149.106 128.59.39.127 SIP/SDP Request: INVITE sip:44832@columbia.edu, with session description

Frame 491 (596 bytes on wire, 596 bytes captured)

Arrival Time: May 10, 2005 10:50:48.664410000

Time delta from previous packet: 6.466699000 seconds

Time since reference or first frame: 6.466699000 seconds

Frame Number: 491

Packet Length: 596 bytes

Capture Length: 596 bytes

Ethernet II, Src: 00:08:02:62:30:5d, Dst: 00:d0:06:26:9c:0a

Destination: 00:d0:06:26:9c:0a (128.59.144.1)

Source: 00:08:02:62:30:5d (dyn-128-59-149-106.dyn.columbia.edu)

Type: IP (0x0800)

Internet Protocol, Src Addr: 128.59.149.106 (128.59.149.106), Dst Addr: 128.59.39.127 (128.59.39.127)

Version: 4

Header length: 20 bytes

Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)

0000 00.. = Differentiated Services Codepoint: Default (0x00)

.... ..0. = ECN-Capable Transport (ECT): 0

....0 = ECN-CE: 0

Total Length: 582

Identification: 0x0441 (1089)

Flags: 0x00

0... = Reserved bit: Not set

.0.. = Don't fragment: Not set

..0. = More fragments: Not set

Fragment offset: 0

Time to live: 128

Protocol: UDP (0x11)

Header checksum: 0x7706 (correct)

Source: 128.59.149.106 (128.59.149.106)

Destination: 128.59.39.127 (128.59.39.127)

User Datagram Protocol, Src Port: 5060 (5060), Dst Port: 5060 (5060)

Source port: 5060 (5060)

Destination port: 5060 (5060)

Length: 562

Checksum: 0x0000 (none)

Session Initiation Protocol

Request-Line: INVITE sip:44832@columbia.edu SIP/2.0

Method: INVITE

Resent Packet: False

Message Header

Via: SIP/2.0/UDP 128.59.149.106:5060;rport;branch=z91hG4bK803b95a20131c9b1425dcf8801405a8100000001

From: Raj Bakhru <sip:35360@columbia.edu>;tag=8755239

SIP Display info: Raj Bakhru

SIP from address: sip:35360@columbia.edu

SIP tag: 8755239

To: <sip:44832@columbia.edu>

SIP to address: sip:44832@columbia.edu

Contact: <sip:35360@128.59.149.106:5060>

Call-ID: EEE355B1-4203-4E63-8A95-CC41B73190AC@128.59.149.106

CSeq: 7472 INVITE

Content-Type: application/sdp

Content-Length: 153

Message body

Session Description Protocol

Session Description Protocol Version (v): 0

Owner/Creator, Session Id (o): - 1 1 IN IP4 128.59.149.106

Owner Username: -

Session ID: 1

Session Version: 1

Owner Network Type: IN

Owner Address Type: IP4

Owner Address: 128.59.149.106

Session Name (s): -

Connection Information (c): IN IP4 128.59.149.106

Connection Network Type: IN

Connection Address Type: IP4

Connection Address: 128.59.149.106

Time Description, active time (t): 0 0

Session Start Time: 0

Session Stop Time: 0

Media Description, name and address (m): audio 16384 RTP/AVP 0

Media Type: audio

Media Port: 16384

Media Proto: RTP/AVP

Media Format: ITU-T G.711 PCMU

Media Attribute (a): rtpmap:101 telephone-event/8000

Media Attribute Fieldname: rtpmap

Media Attribute Value: 101 telephone-event/8000

Media Attribute (a): fmtp:101 0-11,16

Media Attribute Fieldname: fmtp

Media Attribute Value: 101 0-11,16

#3 – SIP 100 Trying

Notes: Indicates call request received

No. Time Source Destination Protocol Info

492 6.469195 128.59.39.127 128.59.149.106 SIP Status: 100 trying -- your call is important to us

Frame 492 (626 bytes on wire, 626 bytes captured)

Arrival Time: May 10, 2005 10:50:48.666906000

Time delta from previous packet: 0.002496000 seconds

Time since reference or first frame: 6.469195000 seconds

Frame Number: 492

Packet Length: 626 bytes

Capture Length: 626 bytes

Ethernet II, Src: 00:d0:06:26:9c:0a, Dst: 00:08:02:62:30:5d

Destination: 00:08:02:62:30:5d (dyn-128-59-149-106.dyn.columbia.edu)

Source: 00:d0:06:26:9c:0a (128.59.144.1)

Type: IP (0x0800)

Internet Protocol, Src Addr: 128.59.39.127 (128.59.39.127), Dst Addr: 128.59.149.106 (128.59.149.106)

Version: 4

Header length: 20 bytes

Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)

0000 00.. = Differentiated Services Codepoint: Default (0x00)

.... ..0. = ECN-Capable Transport (ECT): 0

....0 = ECN-CE: 0

Total Length: 612

Identification: 0x0000 (0)

Flags: 0x04 (Don't Fragment)

0... = Reserved bit: Not set

.1.. = Don't fragment: Set

..0. = More fragments: Not set

Fragment offset: 0

Time to live: 62

Protocol: UDP (0x11)

Header checksum: 0x7d29 (correct)

Source: 128.59.39.127 (128.59.39.127)

Destination: 128.59.149.106 (128.59.149.106)

User Datagram Protocol, Src Port: 5060 (5060), Dst Port: 5060 (5060)

Source port: 5060 (5060)

Destination port: 5060 (5060)

Length: 592

Checksum: 0xeb7b (correct)

Session Initiation Protocol

Status-Line: SIP/2.0 100 trying -- your call is important to us

Status-Code: 100

Resent Packet: False

Message Header

Via: SIP/2.0/UDP 128.59.149.106:5060;rport=5060;branch=z91hG4bK803b95a20131c9b1425dcf8801405a8100000001

From: Raj Bakhru <sip:35360@columbia.edu>;tag=8755239

SIP Display info: Raj Bakhru

SIP from address: sip:35360@columbia.edu

SIP tag: 8755239

To: <sip:44832@columbia.edu>

SIP to address: sip:44832@columbia.edu

Call-ID: EEE355B1-4203-4E63-8A95-CC41B73190AC@128.59.149.106

CSeq: 7472 INVITE

Server: Sip EXpress router (0.8.12 (i386/linux))

Content-Length: 0

Warning: 392 128.59.39.127:5060 "Noisy feedback tells: pid=13294 req_src_ip=128.59.149.106 req_src_port=5060 in_uri=sip:44832@columbia.edu out_uri=sip:44832@128.59.59.242:5060 via_cnt==1"

#4 – SIP/SDP 183 Session Progress

Notes: Indicates phone is ringing; includes "To" "Tag" field; confirms SDP request

No. Time Source Destination Protocol Info

537 6.912946 128.59.39.127 128.59.149.106 SIP/SDP Status: 183 Session Progress, with session description

Frame 537 (849 bytes on wire, 849 bytes captured)

Arrival Time: May 10, 2005 10:50:49.110657000

Time delta from previous packet: 0.436245000 seconds

Time since reference or first frame: 6.912946000 seconds

Frame Number: 537

Packet Length: 849 bytes

Capture Length: 849 bytes

Ethernet II, Src: 00:d0:06:26:9c:0a, Dst: 00:08:02:62:30:5d

Destination: 00:08:02:62:30:5d (dyn-128-59-149-106.dyn.columbia.edu)

Source: 00:d0:06:26:9c:0a (128.59.144.1)

Type: IP (0x0800)

Internet Protocol, Src Addr: 128.59.39.127 (128.59.39.127), Dst Addr: 128.59.149.106 (128.59.149.106)

Version: 4

Header length: 20 bytes

Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)

0000 00.. = Differentiated Services Codepoint: Default (0x00)

.... ..0. = ECN-Capable Transport (ECT): 0

....0 = ECN-CE: 0

Total Length: 835

Identification: 0x0000 (0)

Flags: 0x04 (Don't Fragment)

0... = Reserved bit: Not set

.1.. = Don't fragment: Set

..0. = More fragments: Not set

Fragment offset: 0

Time to live: 62

Protocol: UDP (0x11)

Header checksum: 0x7c4a (correct)

Source: 128.59.39.127 (128.59.39.127)

Destination: 128.59.149.106 (128.59.149.106)

User Datagram Protocol, Src Port: 5060 (5060), Dst Port: 5060 (5060)

Source port: 5060 (5060)

Destination port: 5060 (5060)

Length: 815

Checksum: 0xc6e4 (correct)

Session Initiation Protocol

Status-Line: SIP/2.0 183 Session Progress

Status-Code: 183

Resent Packet: False

Message Header

Via: SIP/2.0/UDP 128.59.149.106:5060;rport=5060;branch=z91hG4bK803b95a20131c9b1425dcf8801405a8100000001

From: Raj Bakhru <sip:35360@columbia.edu>;tag=8755239

SIP Display info: Raj Bakhru

SIP from address: sip:35360@columbia.edu

SIP tag: 8755239

To: <sip:44832@columbia.edu>;tag=A2E42B84-256A

SIP to address: sip:44832@columbia.edu

SIP tag: A2E42B84-256A

Date: Tue, 10 May 2005 14:50:49 GMT

Call-ID: EEE355B1-4203-4E63-8A95-CC41B73190AC@128.59.149.106

Server: Cisco-SIPGateway/IOS-12.x

CSeq: 7472 INVITE

Allow-Events: telephone-event

Contact: <sip:44832@128.59.59.242:5060>

Record-Route: <sip:44832@128.59.39.127;ftag=8755239;lr=on>

Content-Disposition: session;handling=required

Content-Type: application/sdp

Content-Length: 182

Message body

Session Description Protocol

Session Description Protocol Version (v): 0

Owner/Creator, Session Id (o): CiscoSystemsSIP-GW-UserAgent 3630 8230 IN IP4 128.59.59.242

Owner Username: CiscoSystemsSIP-GW-UserAgent

Session ID: 3630

Session Version: 8230

Owner Network Type: IN

Owner Address Type: IP4

Owner Address: 128.59.59.242

Session Name (s): SIP Call

Connection Information (c): IN IP4 128.59.59.242

Connection Network Type: IN

Connection Address Type: IP4

Connection Address: 128.59.59.242

Time Description, active time (t): 0 0

Session Start Time: 0

Session Stop Time: 0

Media Description, name and address (m): audio 16404 RTP/AVP 0

Media Type: audio

Media Port: 16404

Media Proto: RTP/AVP

Media Format: ITU-T G.711 PCMU

Connection Information (c): IN IP4 128.59.59.242

Connection Network Type: IN

Connection Address Type: IP4

Connection Address: 128.59.59.242

Media Attribute (a): rtpmap:0 PCMU/8000

Media Attribute Fieldname: rtpmap

Media Attribute Value: 0 PCMU/8000

#5 – SIP/SDP 200 OK

Notes: Indicates phone has been answered; is resent a number of times if not acknowledges; includes "To" "Tag" and requires it in acknowledgement

No. Time Source Destination Protocol Info

1017 10.296777 128.59.39.127 128.59.149.106 SIP/SDP Status: 200 OK, with session description

Frame 1017 (875 bytes on wire, 875 bytes captured)

Arrival Time: May 10, 2005 10:50:52.494488000

Time delta from previous packet: 3.383771000 seconds

Time since reference or first frame: 10.296777000 seconds

Frame Number: 1017

Packet Length: 875 bytes

Capture Length: 875 bytes

Ethernet II, Src: 00:d0:06:26:9c:0a, Dst: 00:08:02:62:30:5d

Destination: 00:08:02:62:30:5d (dyn-128-59-149-106.dyn.columbia.edu)

Source: 00:d0:06:26:9c:0a (128.59.144.1)

Type: IP (0x0800)

Internet Protocol, Src Addr: 128.59.39.127 (128.59.39.127), Dst Addr: 128.59.149.106 (128.59.149.106)

Version: 4

Header length: 20 bytes

Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)

0000 00.. = Differentiated Services Codepoint: Default (0x00)

.... ..0. = ECN-Capable Transport (ECT): 0

....0 = ECN-CE: 0

Total Length: 861

Identification: 0x0000 (0)

Flags: 0x04 (Don't Fragment)

0... = Reserved bit: Not set

.1.. = Don't fragment: Set

..0. = More fragments: Not set

Fragment offset: 0

Time to live: 62

Protocol: UDP (0x11)

Header checksum: 0x7c30 (correct)

Source: 128.59.39.127 (128.59.39.127)

Destination: 128.59.149.106 (128.59.149.106)

User Datagram Protocol, Src Port: 5060 (5060), Dst Port: 5060 (5060)

Source port: 5060 (5060)

Destination port: 5060 (5060)

Length: 841

Checksum: 0x8134 (correct)

Session Initiation Protocol

Status-Line: SIP/2.0 200 OK

Status-Code: 200

Resent Packet: False

Message Header

Via: SIP/2.0/UDP 128.59.149.106:5060;rport=5060;branch=z91hG4bK803b95a20131c9b1425dcf8801405a8100000001

From: Raj Bakhru <sip:35360@columbia.edu>;tag=8755239

SIP Display info: Raj Bakhru

SIP from address: sip:35360@columbia.edu

SIP tag: 8755239

To: <sip:44832@columbia.edu>;tag=A2E42B84-256A

SIP to address: sip:44832@columbia.edu

SIP tag: A2E42B84-256A

Date: Tue, 10 May 2005 14:50:49 GMT

Call-ID: EEE355B1-4203-4E63-8A95-CC41B73190AC@128.59.149.106

Server: Cisco-SIPGateway/IOS-12.x

CSeq: 7472 INVITE

Allow: INVITE, OPTIONS, BYE, CANCEL, ACK, PRACK, COMET, REFER, SUBSCRIBE, NOTIFY, INFO

Allow-Events: telephone-event

Contact: <sip:44832@128.59.59.242:5060>

Record-Route: <sip:44832@128.59.39.127;ftag=8755239;lr=on>

Content-Type: application/sdp

Content-Length: 182

Message body

Session Description Protocol

Session Description Protocol Version (v): 0

Owner/Creator, Session Id (o): CiscoSystemsSIP-GW-UserAgent 3630 8230 IN IP4 128.59.59.242

Owner Username: CiscoSystemsSIP-GW-UserAgent

Session ID: 3630

Session Version: 8230

Owner Network Type: IN

Owner Address Type: IP4

Owner Address: 128.59.59.242

Session Name (s): SIP Call

Connection Information (c): IN IP4 128.59.59.242

Connection Network Type: IN

Connection Address Type: IP4

Connection Address: 128.59.59.242

Time Description, active time (t): 0 0

Session Start Time: 0

Session Stop Time: 0

Media Description, name and address (m): audio 16404 RTP/AVP 0

Media Type: audio

Media Port: 16404

Media Proto: RTP/AVP

Media Format: ITU-T G.711 PCMU

Connection Information (c): IN IP4 128.59.59.242

Connection Network Type: IN

Connection Address Type: IP4

Connection Address: 128.59.59.242

Media Attribute (a): rtpmap:0 PCMU/8000

Media Attribute Fieldname: rtpmap

Media Attribute Value: 0 PCMU/8000

#6 – SIP ACK

Notes: Acknowledges phone call is answered; call will be dropped if not received in a timely manner; must include variable length "To" "Tag" found in 200 or 183 packets

No. Time Source Destination Protocol Info

1091 10.753871 128.59.149.106 128.59.39.127 SIP Request: ACK sip:44832@columbia.edu

Frame 1091 (422 bytes on wire, 422 bytes captured)

Arrival Time: May 10, 2005 10:50:52.951582000

Time delta from previous packet: 0.457057000 seconds

Time since reference or first frame: 10.753871000 seconds

Frame Number: 1091

Packet Length: 422 bytes

Capture Length: 422 bytes

Ethernet II, Src: 00:08:02:62:30:5d, Dst: 00:d0:06:26:9c:0a

Destination: 00:d0:06:26:9c:0a (128.59.144.1)

Source: 00:08:02:62:30:5d (dyn-128-59-149-106.dyn.columbia.edu)

Type: IP (0x0800)

Trailer: 0A

Internet Protocol, Src Addr: 128.59.149.106 (128.59.149.106), Dst Addr: 128.59.39.127 (128.59.39.127)

Version: 4

Header length: 20 bytes

Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)

0000 00.. = Differentiated Services Codepoint: Default (0x00)

.... ..0. = ECN-Capable Transport (ECT): 0

....0 = ECN-CE: 0

Total Length: 407

Identification: 0x0441 (1089)

Flags: 0x00

0... = Reserved bit: Not set

.0.. = Don't fragment: Not set

..0. = More fragments: Not set

Fragment offset: 0

Time to live: 128

Protocol: UDP (0x11)

Header checksum: 0x77b5 (correct)

Source: 128.59.149.106 (128.59.149.106)

Destination: 128.59.39.127 (128.59.39.127)

User Datagram Protocol, Src Port: 5060 (5060), Dst Port: 5060 (5060)

Source port: 5060 (5060)

Destination port: 5060 (5060)

Length: 387

Checksum: 0x0000 (none)

Session Initiation Protocol

Request-Line: ACK sip:44832@columbia.edu SIP/2.0

Method: ACK

Resent Packet: False

Message Header

Via: SIP/2.0/UDP 128.59.149.106:5060;rport;branch=z91hG4bK803b95a20131c9b1425dcf8801405a8100000001

From: Raj Bakhru <sip:35360@columbia.edu>;tag=8755239

SIP Display info: Raj Bakhru

SIP from address: sip:35360@columbia.edu

SIP tag: 8755239

To: <sip:44832@columbia.edu>;tag=A2E42B84-256A

SIP to address: sip:44832@columbia.edu

SIP tag: A2E42B84-256A

Contact: <sip:35360@128.59.149.106:5060>

Call-ID: EEE355B1-4203-4E63-8A95-CC41B73190AC@128.59.149.106

CSeq: 7472 ACK

Content-Length: 0

#7 – SIP BYE

Notes: Ends call

No. Time Source Destination Protocol Info

1908 17.325200 128.59.39.127 128.59.149.106 SIP Request: BYE sip:35360@128.59.149.106:5060

Frame 1908 (568 bytes on wire, 568 bytes captured)

Arrival Time: May 10, 2005 10:50:59.522911000

Time delta from previous packet: 6.571329000 seconds

Time since reference or first frame: 17.325200000 seconds

Frame Number: 1908

Packet Length: 568 bytes

Capture Length: 568 bytes

Ethernet II, Src: 00:d0:06:26:9c:0a, Dst: 00:08:02:62:30:5d

Destination: 00:08:02:62:30:5d (dyn-128-59-149-106.dyn.columbia.edu)

Source: 00:d0:06:26:9c:0a (128.59.144.1)

Type: IP (0x0800)

Internet Protocol, Src Addr: 128.59.39.127 (128.59.39.127), Dst Addr: 128.59.149.106 (128.59.149.106)

Version: 4

Header length: 20 bytes

Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)

0000 00.. = Differentiated Services Codepoint: Default (0x00)

.... ..0. = ECN-Capable Transport (ECT): 0

....0 = ECN-CE: 0

Total Length: 554

Identification: 0x0000 (0)

Flags: 0x04 (Don't Fragment)

0... = Reserved bit: Not set

.1.. = Don't fragment: Set

..0. = More fragments: Not set

Fragment offset: 0

Time to live: 62

Protocol: UDP (0x11)
Header checksum: 0x7d63 (correct)
Source: 128.59.39.127 (128.59.39.127)
Destination: 128.59.149.106 (128.59.149.106)
User Datagram Protocol, Src Port: 5060 (5060), Dst Port: 5060 (5060)
Source port: 5060 (5060)
Destination port: 5060 (5060)
Length: 534
Checksum: 0x0313 (correct)
Session Initiation Protocol
Request-Line: BYE sip:35360@128.59.149.106:5060 SIP/2.0
Method: BYE
Resent Packet: False
Message Header
Record-Route: <sip:44832@128.59.39.127;ftag=A2E42B84-256A;lr=on>
Via: SIP/2.0/UDP 128.59.39.127;branch=z9hG4bK90c6.41215125.0
Via: SIP/2.0/UDP 128.59.59.242:5060
From: <sip:44832@columbia.edu>;tag=A2E42B84-256A
SIP from address: sip:44832@columbia.edu
SIP tag: A2E42B84-256A
To: Raj Bakhru <sip:35360@columbia.edu>;tag=8755239
SIP Display info: Raj Bakhru
SIP to address: sip:35360@columbia.edu
SIP tag: 8755239
Date: Tue, 10 May 2005 14:50:54 GMT
Call-ID: EEE355B1-4203-4E63-8A95-CC41B73190AC@128.59.149.106
User-Agent: Cisco-SIPGateway/IOS-12.x
Max-Forwards: 5
Timestamp: 1115736660
CSeq: 101 BYE
Content-Length: 0

#8 – RTP

Notes: Payload encoded in G.711mu-law; Sequence time and offset dynamically appended w/ payload to preloaded RTP framework

No. Time Source Destination Protocol Info

1907 17.300486 128.59.149.106 128.59.59.242 RTP Payload type=ITU-T G.711 PCMU, SSRC=592995394, Seq=400, Time=160000

Frame 1907 (214 bytes on wire, 214 bytes captured)

Arrival Time: May 10, 2005 10:50:59.498197000

Time delta from previous packet: 0.028409000 seconds

Time since reference or first frame: 17.300486000 seconds

Frame Number: 1907

Packet Length: 214 bytes

Capture Length: 214 bytes

Ethernet II, Src: 00:08:02:62:30:5d, Dst: 00:d0:06:26:9c:0a

Destination: 00:d0:06:26:9c:0a (128.59.144.1)

Source: 00:08:02:62:30:5d (dyn-128-59-149-106.dyn.columbia.edu)

Type: IP (0x0800)

Internet Protocol, Src Addr: 128.59.149.106 (128.59.149.106), Dst Addr: 128.59.59.242 (128.59.59.242)

Version: 4

Header length: 20 bytes

Differentiated Services Field: 0xb8 (DSCP 0x2e: Expedited Forwarding; ECN: 0x00)

1011 10.. = Differentiated Services Codepoint: Expedited Forwarding (0x2e)

.... ..0. = ECN-Capable Transport (ECT): 0

....0 = ECN-CE: 0

Total Length: 200

Identification: 0x0441 (1089)

Flags: 0x00

0... = Reserved bit: Not set

.0.. = Don't fragment: Not set

..0. = More fragments: Not set

Fragment offset: 0

Time to live: 128

Protocol: UDP (0x11)

Header checksum: 0x6359 (correct)

Source: 128.59.149.106 (128.59.149.106)

Destination: 128.59.59.242 (128.59.59.242)

User Datagram Protocol, Src Port: 16384 (16384), Dst Port: 16404 (16404)

Source port: 16384 (16384)

Destination port: 16404 (16404)

Length: 180

Checksum: 0x0000 (none)

Real-Time Transport Protocol

Stream setup by SDP (frame 1017)

Setup frame: 1017

Setup Method: SDP

10.. .... = Version: RFC 1889 Version (2)

..0. .... = Padding: False

...0 .... = Extension: False

.... 0000 = Contributing source identifiers count: 0

0... .... = Marker: False

.000 0000 = Payload type: ITU-T G.711 PCMU (0)

Sequence number: 400

Timestamp: 160000

Synchronization Source identifier: 592995394

Payload: 54EEE8E6E5D1697E575A666CE5D4DBE46D4DE26CEBF5654C...