Technical Report
Initiated Protocol Telephony Feasibility
For the US Navy,
Embedded Proof-of-Concept

ISRN L3COM/MARITIMESYSTEMS/TR -- 2011/002

Contract Number: N00014-09-C-0618

Prepared For:

Prepared by:

Maritime Systems
9 Malcolm Hoyt Drive
Newburyport, MA 01950 U.S.A.

This page intentionally left blank.

Maritime Systems

Intelligent Advanced Communications IP Telephony
Feasibility for the US Navy – Phase 3

REPORT DOCUMENTATION PAGE		*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**		
1. REPORT DATE *(DD-MM-YYYY)* 31-03-2011	**2. REPORT TYPE** Research	**3. DATES COVERED** *(From - To)*
4. **TITLE AND SUBTITLE** Intelligent Advanced Communications Assured Services Session Initiated Protocol Telephony Feasibility for the US Navy, Embedded Proof-of-Concept		**5a. CONTRACT NUMBER** N00014-09-C-0618 **5b. GRANT NUMBER** **5c. PROGRAM ELEMENT NUMBER**
6**. AUTHOR(S)** Binns, Todd D; Principle Investigator Oram, Jerald; Investigator Keohane, Stephen; Investigator		**5d. PROJECT NUMBER** N.A. **5e. TASK NUMBER** **5f. WORK UNIT NUMBER**
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) L-3 Maritime Systems 9 Malcolm Hoyt Drive Newburyport, MA 01950-4017		**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Office of Naval Research 875 N Randolph St., Suite 1425 Arlington, VA 22203-1995		**10. SPONSOR/MONITOR'S ACRONYM(S)** ONR **11. SPONSOR/MONITOR'S REPORT NUMBER(S)**
12. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release, distribution is unlimited		
13. SUPPLEMENTARY NOTES		

14. ABSTRACT

The objective of this research paper is to document our experiences building an assured services SIP (AS-SIP) end instrument from a commercial, embedded SIP stack as defined by the DOD Unified Capabilities Requirements specification. The SIP stack chosen came from a company called Unicoi Systems. We chose an existing reference design based on the Analog Devices *Blackfin 537* DSP (Digital Signal Processor) (ADSP-BF537) running Unicoi's Fusion RTOS (real time operating system), SIP and related networking stacks. The reference design also included a complete SIP phone application embedded into a Fanstel ST-3116 phone. It should be noted that while this was an embedded solution, the Unicoi source code could also be cross compiled for nearly any host based platform such as Windows.
This report is based on the following research and validation testing:

- L-3 Maritime Systems internal research,
- L-3 Maritime Systems VoIP lab research,
- L-3 Maritime Systems prototyping of a VoIP Communications Terminal,-
- Unified Capabilities Requirements 2008, Change 1 Final
- L-3 Maritime Systems Technical Report *Intelligent Advanced Communications IP Telephony Feasibility for the US Navy*, Volumes 1 and 2 (ISRN L3COM/Maritime Systems/TR-2007/001).
- Unicoi Fusion RTOS, SIP and networking stacks, call manager and voice engine
- REDCOM Slice2100 soft switch with Transip, HDX
- GL Communications VoIP quality test tool.
- Unicoi Licensing requirements
- Wireshark

The major findings are:
- SIP stack behavior and characteristics vary substantially among vendors, be it commercial or open source and the Unicoi stack was no exception.
- Unlike traditional TDM based phones, SIP stacks and consequently SIP phones are somewhat switch specific in terms of compatibility and vary substantially in terms of what they chose to support. Where you can easily plug a TDM based phone into just about any TDM network, the same is not true for IP phones integrated with the SIP stack or at all.
- The inherent nature of the embedded environment makes the embedded solution far more secure than a typical PC based soft phone.
- IP based voice and video networks are already in wide use today in networks where there is little or no control over competing network traffic. A closed network environment on board a ship would provide an optimal environment for IP based communications.

15. SUBJECT TERMS VoIP, SIP, H.323, US Navy, Vessels, Communications, Network, SVoIP, VoSIP, SVoSIP, AS-SIP, Assured Services SIP, Blackfin					
16. SECURITY CLASSIFICATION OF: U			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Christopher P. Rigano (CISSP)
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE** 2	UU		**19b. TELEPHONE NUMBER** *(include area code)* 703.696.65942

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

This page intentionally left blank.

TABLE OF CONTENTS

TABLE OF CONTENTS - Continued

LIST OF ILLUSTRATIONS

This page intentionally left blank.

Abstract

The objective of this research paper is to document our experiences building an assured services Session Initiation Protocol (SIP) (AS-SIP) end instrument from a commercial, embedded SIP stack as defined by the DOD Unified Capabilities Requirements specification. The SIP stack chosen came from a company called Unicoi Systems. We chose an existing reference design based on the Analog Devices *Blackfin 537* DSP (Digital Signal Processor) (ADSP-BF537) running Unicoi's Fusion RTOS (real time operating system), SIP and related networking stacks. The reference design also included a complete SIP phone application embedded into a Fanstel ST-3116 phone. It should be noted that while this was an embedded solution, the Unicoi source code could also be cross compiled for nearly any host based platform such as Windows.

This report is based on the following research and validation testing:

- L-3 Maritime Systems internal research,
- L-3 Maritime Systems VoIP lab research,
- L-3 Maritime Systems prototyping of a VoIP Communications Terminal,-
- Unified Capabilities Requirements 2008,
- L-3 Maritime Systems Technical Report *Intelligent Advanced Communications IP Telephony Feasibility for the US Navy*, Volumes 1 and 2 (ISRN L3COM/Maritime Systems/TR-2007/001).
- Unicoi Fusion RTOS, SIP and networking stacks, call manager and voice engine
- REDCOM Slice2100 soft switch with Transip, HDX
- GL Communications VoIP quality test tool.
- Unicoi Licensing requirements

The major findings are:

- SIP stack behavior and characteristics vary substantially among vendors, be it commercial or open source and the Unicoi stack was no exception.
- Unlike traditional TDM based phones, SIP stacks and consequently SIP phones are somewhat switch specific in terms of compatibility and vary substantially in terms of what they chose to support. Where you can easily plug a TDM based phone into just about any TDM network, the same is not true for IP phones integrated with the SIP stack or at all.
- The inherent nature of the embedded environment makes the embedded solution far more secure than a typical PC based soft phone.
- IP based voice and video networks are already in wide use today in networks where there is little or no control over competing network traffic. A closed network environment on board a ship would provide an optimal environment for IP based communications.

This page intentionally left blank.

CHAPTER 1 Embedded Proof-of-Concept

1.1 Embedded Proof-Of-Concept

1.2 Introduction

Much of the information and findings for our embedded prototype are identical to that of the open source stacks developed. The fact that the Unicoi Session Initiation Protocol (SIP) stack was embedded was irrelevant in terms of the performance and supported features of the stack. Very little of the Unicoi stack worked out of the box against the REDCOM switch in terms of basic SIP phone functionality. This should not be read as an indictment of the Unicoi SIP stack. Like every SIP stack investigated, the SIP features and soft switches they chose to support did not include a militarized SIP switch like the REDCOM SLICE used to test all of our applications. The majority of our time was spent just getting the stock Unicoi SIP stack to behave properly when connected to the REDCOM SLICE soft switch.

One major advantage to the embedded approach was the security that is inherent to the embedded environment over traditional hosted based platforms like Windows. Embedded environments are typically headless systems that are physically locked down in addition to the traditional software hardening capabilities most hosted based platforms employ.

1.2.1 SIP and Assured Services SIP

Understanding the differences between Assured Services SIP and standard SIP is not a trivial task. While the DOD Unified Capabilities Requirements (UCR) documentation does an excellent job of identifying the various networking components that each requirement applies to (LSC, EI, MFSS, GEI, TA, IAD, MG etc...) there was no attempt made to highlight the actual changes made to the original IETF RFCs and standards that define assured services Session Initiation Protocol (SIP) (AS-SIP). As it turns out, the SIP related UCR requirements are no different than the numerous RFCs and standards they apply to and in many cases are simply quoted word for word in the UCR.

RFC 5638 (Simple SIP) specifies the support of only 11 RFCs necessary to create a SIP appliance with presence, instant messaging, audio and video communications. The RFC blames "the emulation of telephony and its model of the intelligent network" as the reason there are now more than100 RFCs that can define the behavior of a SIP appliance. Unfortunately for us, "emulating the existing telephony model" is exactly what the UCR requires us to do. The UCR requires support for nearly 200 RFCs although it should be noted that not all pertain directly to SIP. It is the substantially large number of requirements for the end instrument that make AS-SIP different from your typical SIP stack for an end instrument. This is not to say there are no variations from the original RFCs because there are plenty although most are subtle.

1.2.2 AS-SIP Messages

AS-SIP as defined by the DOD UCR document is no different than standard SIP in terms of the overall structure and messaging. A brief summary of assured services SIP related requirements are as follows.

> An AS-SIP message is the same as a standard SIP message and is either a request from a client to a server, or a response from a server to a client.
>
> generic-message = start-line
> *message-header
> CRLF
> [message-body]
> start-line = Request-Line / Status-Line
>
> AS-SIP Requests
>
> Request-Line = Method SP Request-URI SP SIP-Version CRLF
>
> Required methods for AS-SIP:
>
> | | |
> |---|---|
> | ACK | (RFC 3261) |
> | BYE | (RFC 3261) |
> | INVITE | (RFC 3261) |
> | CANCLE | (RFC 3261) |
> | REGISTER | (RFC 3261) |
> | OPTIONS | (RFC 3261) |
> | PRACK | (RFC 3262) AS-SIP req. |
> | UPDATE | (RFC 3311) AS-SIP req. |
> | REFER | (RFC 3515) AS-SIP req. |
> | NOTIFY | (RFC 3265) AS-SIP req. |

Example AS-SIP message:

> INVITE sip:bob@biloxi.com SIP/2.0

A valid AS-SIP request formulated by a UAC MUST, at a minimum, contain the following header fields: **To**, **From**, **CSeq**, **Call-ID**, **Max-Forwards** and **Via.**

These above six header fields are the fundamental building blocks of all AS-SIP messages and are defined as follows.

1. **To**:

The **To** header field first and foremost specifies the desired "logical" recipient of the request. For AS-SIP, the **userinfo** part must contain the 10 digit DSN number plus the hostname. The **To** header field contains the address of record whose registration is to be created, queried, or modified. The **To** header field and the Request-URI field typically differ, as the former contains a user name. This address-of-record MUST be a SIP URI or SIPS URI. AS-SIP: requires **userinfo**

Example **To** headers:

To: The Operator <sip:operator@cs.columbia.edu>;tag=287447
t: sip:+12125551212@server.phone2net.com
(AS-SIP must contain the 10 digit number)

2. **From**:

The **From** header field contains the address-of-record of the person responsible for the REquest(eg. register). The value is the same as the **To** header field unless the request is a third-party registration for Registration.

3. **Call-ID**:

All registrations from a UAC SHOULD use the same **Call-ID** header field value for registrations sent to a particular registrar. If the same client were to use different **Call-ID** values, a registrar could not detect whether a delayed REGISTER request might have arrived out of order.

4. **CSeq**:

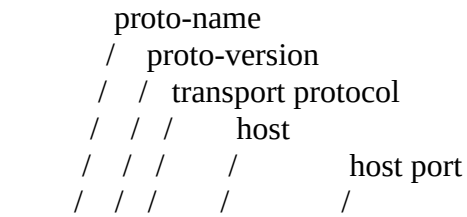
The **CSeq** value guarantees proper ordering of REGISTER requests. A UA MUST increment the **CSeq** value by one for each REGISTER request with the same **Call-ID**.

5. **Max-Forwards**:

Serves to limit the number of hops a request can make on the way to it’s destination. It consists of an integer that is decremented by 1 at each hop.

Example: Max-Forwards: 70

6. **Via**:
The **Via** header field indicates the transport used for the transaction and identifies the location where the response is to be sent. A **Via** header field value is added only after the transport that will be used to reach the next hop has been selected.



Example: Via: SIP/2.0/TLS atlanta.example.com:5060
v: SIP/2.0/TCP client.atlanta.example.com:5060
;branch=z9hG4bK74b76
;received=192.0.2.101

The **Request-URI** and **Contact** headers may also be required depending on the type of request:

7. **Request-URI**:
The **Request-URI** names the domain of the location service for which the registration is meant. The **userinfo** and **@** components of the SIP URI must not be present.

Example: "sip:chicago.com"

8. **Contact**:
REGISTER requests must contain a **Contact** header field with zero or more values containing address bindings.

9. AS-SIP Responses:

Status-Line = SIP-Version SP Status-Code SP Reason-Phrase CRLF

Example: SIP/2.0 200 OK

1.2.3 SIP State

There are two types of client transaction state machines, depending on the method of the request passed by the TU.

1. Client transactions for INVITE requests.
2. Client transactions for all requests except INVITE and ACK or "non-INVITE client transactions"

Note that there is no client transaction for ACK. If the TU wishes to send an ACK, it passes one directly to the transport layer for transmission. The INVITE transaction is different from those of other methods because of the human input normally required to complete it and consequently its relatively long life time.

Support for the following SIP headers is required for AS-SIP:

Accept	(RFC 3261)
Alert-Info	(RFC 3261)
Allow	(RFC 3261)
Allow-Events (u)	(RFC 3265) AS-SIP req.
Authorization	(RFC 3261 generate and send only AS-SIP)
Call-ID (i)	(RFC 3261)
Contact (m)	(RFC 3261)
Content-Disposition	(RFC 3261)
Content-Length (l)	(RFC 3261)
Content-Type (c)	(RFC 3261)
CSeq	(RFC 3261)
Date	(RFC 3261)
Event (o)	(RFC 3265) AS-SIP req.
Expires	(RFC 3261)
From (f)	(RFC 3261)
Max-Forwards	(RFC 3261)
Min-Expires	(RFC 3261)
Min-SE	(RFC 4028) AS-SIP req.
P-Asserted-Identity	(RFC 3325 receive only) AS-SIP req.
Proxy-Authenticate	(RFC 3261 receive only)
Proxy-Authorization	(RFC 3261 generate and send only)
Proxy-Require	(RFC 3261 generate for Classified network only)
Rack	(RFC 3262) AS-SIP req.
Reason	(RFC 3326) AS-SIP req.
Record-Route	(RFC 3261)

Refer-To (r)	(RFC 3515, 4508) AS-SIP req.
Replaces	(RFC 3891) AS-SIP req.
Require	(RFC 3261)
Resource-Priority	(RFC 4412) AS-SIP req.
Retry-After	(RFC 3261)
RSeq	(RFC 3262) AS-SIP
Session-Expires (x)	(RFC 4028) AS-SIP
Subscription-State	(RFC 3265) AS-SIP
Supported (k)	(RFC 3261)
To (t)	(RFC 3261)
Unsupported	(RFC 3261)
Via (v)	(RFC 3261)
Warning	(RFC 3261)
WWW-Authenticate	(RFC 3261 generate and send only)

PRACK or Provisional Response Acknowledgement plays the same role as ACK except that it is for provisional responses only. PRACK is a standard SIP message and has its own set of responses. It was surprisingly difficult to find true support for PRACK in most of the stacks evaluated and the *Unicoi* stack was no exception. Support for PRACK was a reoccurring problem with the *Unicoi* stack and ultimately required significant changes to their core stack implementation.

IPsec or *Internet Protocol Security* was not supported in any of the stacks surveyed, commercial or open source. It is a protocol for securing and authenticating each individual IP packet. Internet security protocols like SSL (secure sockets layer) or TLS (transport layer security) operate at higher layers in TCP/IP which means that IPsec can secure any application traffic.

1.2.4 AS-SIP Multilevel Precedence and Preemption (MLPP)

AS-SIP end instruments tag SIP communications with a precedence level or priority. This is accomplished by populating the **Resource-Priority** header**. Resource-Priority** consists of a namespace and a priority. The namespace is divided into 2 subfields separated by a "-". The value left of the "-“ represents the "network domain" and the value to the right of the "-" is the "precedence domain". The "precedence domain" is currently set to "000000" for the time being. The namespace is then prefixed with ".priority" where priority is one of the following values:

0 Routine (lowest priority)
2 Priority
4 Immediate
6 Flash
8 Flash-Override (highest priority)

Example:

INVITE sip:user5001@192.168.0.138:54912 SIP/2.0
Via: SIP/2.0/UDP 192.168.0.11;branch=z9hG4bKe4580000056b90f5;rport
From: "ISDN Phone 2b" <sip:1015@192.168.0.11;user=phone>;tag=8a3b000000220d55b2c6
To: "SIP 5001" <sip:user5001@192.168.0.11;user=phone>
Contact: <sip:1015@192.168.0.11;CCA-ID=redcom;user=phone>
Call-ID: 7c405a43f9989320@192.168.0.11
CSeq: 159 INVITE
Max-Forwards: 70
Session-Expires: 3600;refresher=uac
Min-SE: 300
Allow:
REGISTER,INVITE,ACK,CANCEL,BYE,OPTIONS,INFO,REFER,SUBSCRIBE,NOTIFY,PR
ACK,
UPDATE
Allow-Events: dialog,message-summary
Supported: replaces, timer, 100rel, resource-priority
Resource-Priority: dsn-000000.6
Content-Type: application/sdp
Content-Length: 347
User-Agent: REDCOM SLICE 2100 4.0a R3P3 18-Jun-2010

The example INVITE message above has a **Resource-Priority** header with a network domain of "dsn", a precedence domain of "000000" and a priority of "6" meaning this is a "FLASH" priority call. Our AS-SIP phone supported both the legacy dial prefix method of prioritizing a call and also took advantage of the software GUI to allow the user to visually select a call priority at the touch of a button. To manually set the priority of a call the dialed number is prefixed with one of the following 2 digits:

90 (FLASH OVERRIDE)
91 (FLASH)
92 (IMMEDIATE)
93 (PRIORITY)
94 (ROUTINE)

For our AS-SIP phone, these digits have no purpose and were simply stripped out and used to set the outbound Resource-Priority in software to whatever the user indicated on the fly. In the absence of these prefix digits, the priority is set to whatever was selected via the GUI on the phone. By default, the phone comes up with a priority of ROUTINE. It should be noted that with AS-SIP, much of the burden of supporting priority is shifted from the switch to the end

instrument.

AS-SIP end instruments are required to support two lines. Preemption of a call does not occur unless both lines are occupied and a 3rd call arrives with a higher priority than either of the 2 lines in use and the arriving call has the same precedence domain as either of the calls on the lines in use. Preempted calls must be sent a Reason header with a cause code of 1 when the call is hung up. RFC 4411 defines 4 cause codes as shown below. Cause code 5 is a UCR requirement only and effects server side SIP appliances only.

Reason: preemption ;cause=1 ;text="UA Preemption"
Reason: preemption ;cause=2 ;text="Reserved Resources Preempted"
Reason: preemption ;cause=3 ;text="Generic Preemption"
Reason: preemption ;cause=4 ;text="Non-IP Preemption"
Reason: preemption; cause=5; text="Network Preemption"

Example BYE with cause code = 1 to indicate to the receiving end instrument that the call was preempted:

BYE sip:user5001@192.168.0.138:54912 SIP/2.0
Via: SIP/2.0/UDP 192.168.0.11;branch=z9hG4bKe4580000056b90f5;rport
From: "ISDN Phone 2b" <sip:1015@192.168.0.11;user=phone>;tag=8a3b000000220d55b2c6
To: "SIP 5001" <sip:user5001@192.168.0.11;user=phone>
Contact: <sip:1015@192.168.0.11;CCA-ID=redcom;user=phone>
Call-ID: 7c405a43f9989320@192.168.0.11
CSeq: 159 INVITE
Max-Forwards: 70
Session-Expires: 3600;refresher=uac
Min-SE: 300
Allow:
REGISTER,INVITE,ACK,CANCEL,BYE,OPTIONS,INFO,REFER,SUBSCRIBE,NOTIFY,PR
ACK,
UPDATE
Reason: preemption ;cause=1 ;text="UA Preemption"
Allow-Events: dialog,message-summary
Supported: replaces,timer,100rel,resource-priority
Resource-Priority: dsn-000000.6
Content-Type: application/sdp
Content-Length: 347
User-Agent: REDCOM SLICE 2100 4.0a R3P3 18-Jun-2010

The Reason header with cause code=1 was used to trigger preemption events in our end instrument, specifically the visual indication of preemption on the screen.

1.2.5 AS-SIP Precedence Rules

When a higher precedence call arrives with a different priority domain, no preemption occurs at the end instrument. The serving Local Session Controller (LSC) is required to handle this scenario. The LSC is also responsible for handling preemption on single line end instruments and for handling a 480 response from the EI when the EI has no lines available and the caller is of lower precedence. Precedence occurs regardless of media type as well, meaning audio and video for example are treated equally.

When a call arrives of higher precedence than either of the 2 existing calls, the lower precedence call is preempted. The preempted call will display a preemption message as well as receive a preemption tone until the onhook is received. The called party also receives a preemption tone for a minimum of 3 seconds. The called party must also receive a preemption ring tone after going on hook on the preempted call. Upon going back off hook again, the called party must be connected to the priority call.

When a higher precedence call arrives on a multi-line phone that still has one appearance available, then a precedence ring tone is provided however call preemption is entirely manual at this point and up to the called party to decide what to do with the call.

When the preempted call is a held call the preemption tone must still be played to the held call. The preemption ringtone is played immediately after the held call is dropped.

Preemption still applies to call forwarding in some instances. The UCR defines three call forward settings:

1. Call Forward Unconditional
2. Call Forward No Answer
3. Call Forward Busy.

In all instances, the precedence level of an incoming call must be preserved.
No preemption occurs when the phone is forwarded "unconditionally".
Preemption and precedence levels are still in force for call forward "busy".
Both the legacy method of setting call forward with manual prefix digits and the graphical user interface (GUI) to the SIP phone were supported. To manually set call forward the following dial prefixes were supported:

1. Call Forward Unconditional: *72 (*73 to deactivate)
2. Call Forward No Answer: *92 (*93 to deactivate)
3. Call Forward Busy: *90 (*91 to deactivate)

As with the preemption prefix digits, these digits had no meaning and were simply stripped out

and used to set the call forward state of the SIP phone only. Users could also take advantage of the AS-SIP GUI and simply press a soft key to set or unset call forwarding.
The tangible differences between AS-SIP and SIP, those differences seen by the end user, could best be summed up as:

1. Precedence level marking of each call
2. Permission to use that marking
3. Preferential treatment of calls
4. The diversion (forwarding) of calls on no answer, busy and unconditional
5. Preemption notifications
6. The security and protection of both signaling and routing information

1.3 *Unicoi* Fusion Embedded Solutions

As previously stated, the project was started with a complete application from *Unicoi* based on the Analog Devices *Blackfin 537 DSP* (ADSP-BF537) running *Unicoi* Fusion RTOS (real time operating system), SIP stack and related networking stacks (IPV4/IPV6). The phone application we started with was a standard "C" application written for the Fanstel ST-3116 VoIP phone (see Figure 1). This application was neither written nor maintained by *Unicoi* and was refactored to meet the requirements of the UCR.

Figure 1 Fanstel ST-3116

The Fanstel ST-3116 is equipped with the following stock features.

1.3.1 Telephony Features

1. Loud, high performance speakerphone
2. 3-party local conferencing, no need to use processing power of a PBX for conferencing;
3. RJ22 and 2.5 mm headset ports.
4. Six lines with LEDs (2 trunk lines and 4 Busy Lamp Fields (BLFs))
5. Four BLF buttons to speed dial and pick up ringing extension.
6. User option to display BLFs on LCD.
7. Six soft buttons to access PBX features:

   Transfer, Park, Hold/Unhold, Conference, Send to Voicemail, Monitor Calls.
8. Pick Up, DND, Unpark, Voice mail, Intercom, Meeting room
9. Hard feature buttons: Speaker (LED), Mute (LED), Headset (LED), Directory, Redial, Goodbye, Mic/Receiver
10. Customizable ring tones and ring volume
11. Adjustable handset, headset, and speakerphone receiver/microphone volume
12. 4 programmable memory buttons.
13. Caller ID, Call Waiting ID, VMWI
14. Download features by XML messaging
15. Download directory by an XML file
16. High contrast backlit, 7.4 sq. inch LCD, adjustable viewing angle.
17. 26 characters x 9 lines display
18. Scalable font size for easy viewing.
19. Supports IEEE 802.3af PoE.

20. Includes an AC adapter;
21. Desk or wall mount;
22. Size: 235 x 182 (mm); 9.27 x 7.18 (inches)
23. Warranty: 1 year (parts and labor)
24. Approval: FCC, CE, IC

1.3.2 Networking Features

1. Two 10/100 Mbps switched Ethernet ports
2. SIP (RFC 3261 and companion RFCs)
3. Supports Real Time Text (RTT)- new 1Q2009
4. SIP security using TLS- new 1Q2009
5. SRTP/SDES for voice encryption- new 1Q2009
6. DHCP (Dynamic Host Configuration Protocol)
7. HTTP server for web administration and maintenance
8. Auto-provisioning in less than 2 minutes.
9. G.711u, G.711a, G.722, and G.729ab codecs.
10. TFTP firmware upgrade and configuration

The *Unicoi* software is comprised of several well defined components and can best be described in layers:

a. RTOS: Fusion Real Time Operating System can be thought of as the base l layer that allows our application to take advantage of the BF-537 DSP hardware.

b. Network: IPV4/IPV6 networking stacks and related ethernet drivers

c. SIP/SIPS: the SIP stack

d. VE: the *Unicoi* voice engine

e. CM: the *Unicoi* call manager

f. Cougar: the SIP phone application for the Fanstel ST-3116, this is not part of *Unicoi*'s software but an application that uses it.

It should be noted that all of the above components comprise and are built into the executable that is run on the BF-537 DSP. It's an important distinction. The application in this embedded environment includes everything from the OS (RTOS) to the low level drivers and networking stacks. It is in a sense the operating system at run time unlike a traditional host based application that runs within the confines and protection of an OS like Windows or Linux. Bugs and programming errors are not caught like the way Windows would do for a mal formed program. Mistakes bring down the entire board. Complete rebuilds of our AS-SIP client application can take up to 20 minutes to complete because of all the software involved.

1.3.3 Building the Embedded Software Image

Building and loading new software onto the phone is a highly custom embedded process developed by *Unicoi*. We spent a week at their facility learning their embedded development environment and related tools. Applications for this platform are developed, compiled, linked and run using Visual DSP++. Visual DSP++ is an integrated development environment (IDE) with the same look and feel as most common IDE's for host based platforms such as Microsoft Dev Studio. It is a C/C++ compiler that generates binary code images for specific Analog Devices based DSP (digital signal processors).

Binary images, or "programs" can be loaded and run via 2 methods in this embedded environment: using the embedded boot loader software or manually through a JTAG interface. "JTAG" stands for "Joint Test Action Group" and is an IEE standard that has been adopted by electronic device makers all over the world. As printed circuit boards became smaller and components even smaller, it was no longer possible to test connectivity and or components directly on the board. Printed circuit boards are multi-dimensional, where circuit paths are commonly buried within a multiple layers on the board and not visible or even accessible to a voltmeter, scope and other standard test equipment for electronic devices. IEEE (i.e. read as i-triple-e) is the Institute of Electrical and Electronics Engineers. Refer to IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture for a complete description of JTAG. For our purposes, it was simply a USB connection from our development PC to a 13 pin JTAG connector on the *Blackfin 537* board. It allowed us to connect to the board to load, execute and debug our AS-SIP application. The application could also be loaded and run using a custom telnet based shell. This method burns the image into flash memory permanently or until another image is loaded.

1.3.4 Building Loading and Executing from VDSP++

The project groups for our embedded application are located under:

$INSTALL_DIR\fusion\refdesigns\voip\ip_phone\ports\cougar\bf537

Project groups are essentially the **make** files that tell VDSP++ what is included in the build and also how to compile and link the application. Our **Cougar** reference design included three

different project groups as shown below. VDSP++ project groups all have the **.dpg** file extension:

cougar.dpg

cougar_sips.dpg

cougar_sips_srtp.dpg

As the names imply, the cougar.dpg does not include secure mode or SRTP, the cougar_sips.dpg includes support for TLS only and the 3rd project includes both TLS and SRTP. Our AS-SIP end instrument is built with the all inclusive project: cougar_sips_srtp.dpg. It's important to note that simply building this project does not enable the security features. Both external configuration and compilation flags must be set to enable SRTP and or TLS and these topics are covered in later sections of this document. Each of these separate project groups includes many smaller projects as show below. The cougar_sips_srtp.dpg project is comprised of 20 different packages, everything from RTOS to the SIP stack (Figure 2):

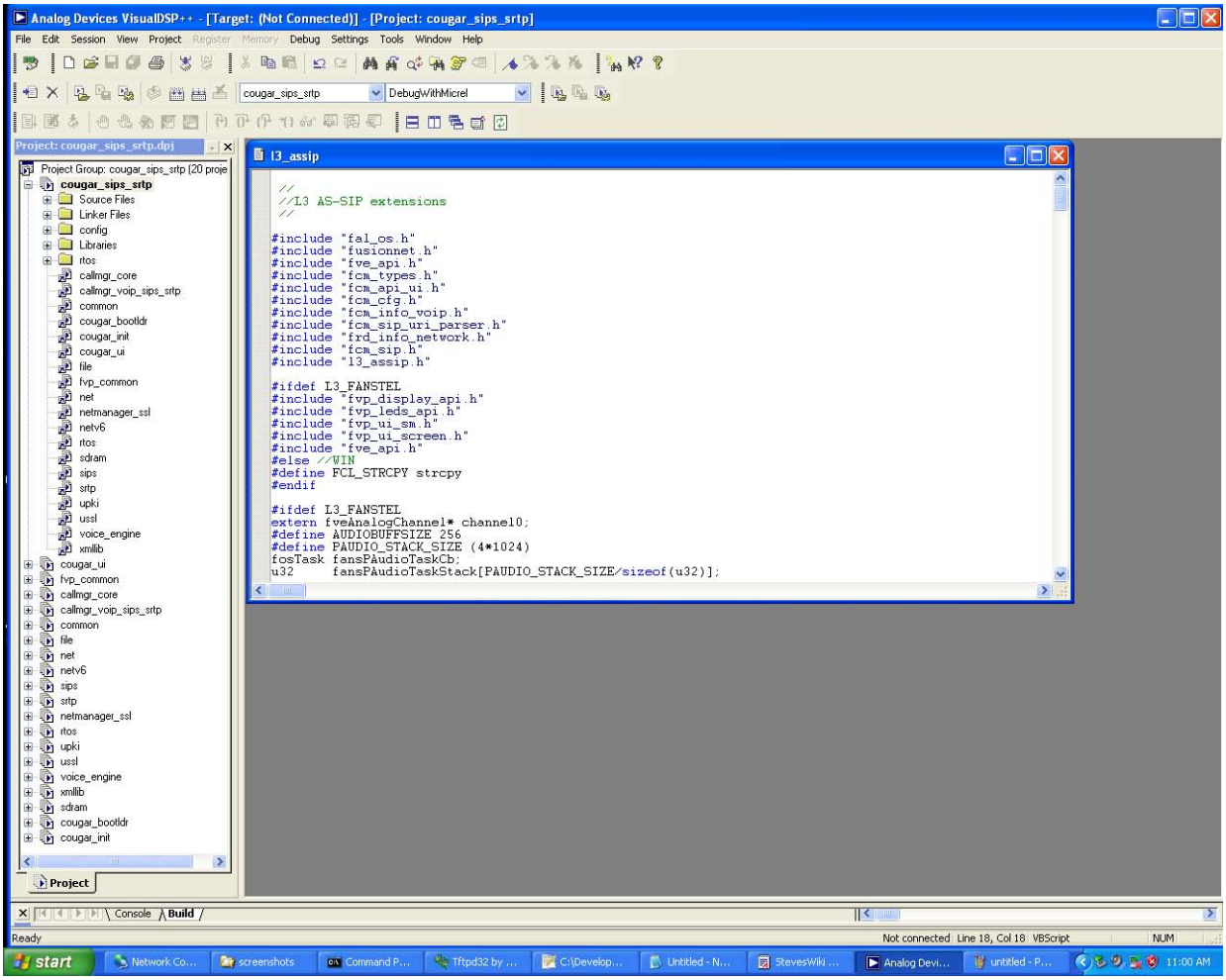


Figure 2 VDSP

To compile and build the AS-SIP end instrument, start the VDSP++ development environment, use File/Open to open the **cougar_sips_srtp.dpg** project and then right button down on the **cougar_sips_srtp** project to reveal the various build options as shown in Figure 2.

A complete rebuild will take approximately 20 minutes. VDSP++ also maintains a list of what's actually changed to do incremental builds. You can follow the progress of the build in Output display window. Select the **View** tab to open the Output display window if not already open.

The build will produce several files of particular interest in the *$UNICOI_INSTALLDIR\fusion\refdesigns\voip\ip_phone\ports\cougar\bf537\DebugWithMicrel* folder:

sdram.dxe

cougar_sips_srtp.dxe

cougar_sips_srtp.ldr

The **.dxe** files are use to load and run the application from within the VDSP++ environment in emulator mode. Before this can be done however, a VDSP++ "session" must be created. A "session" tells VDSP++ which DSP processor to the code will be run against. The session is created in the following steps is for *BF-537 Blackfin* DSP processor.

1. To create the session, open the VDSP++ development environment and select **New** under the **Session** tab along the top toolbar as shown in Figure 3:

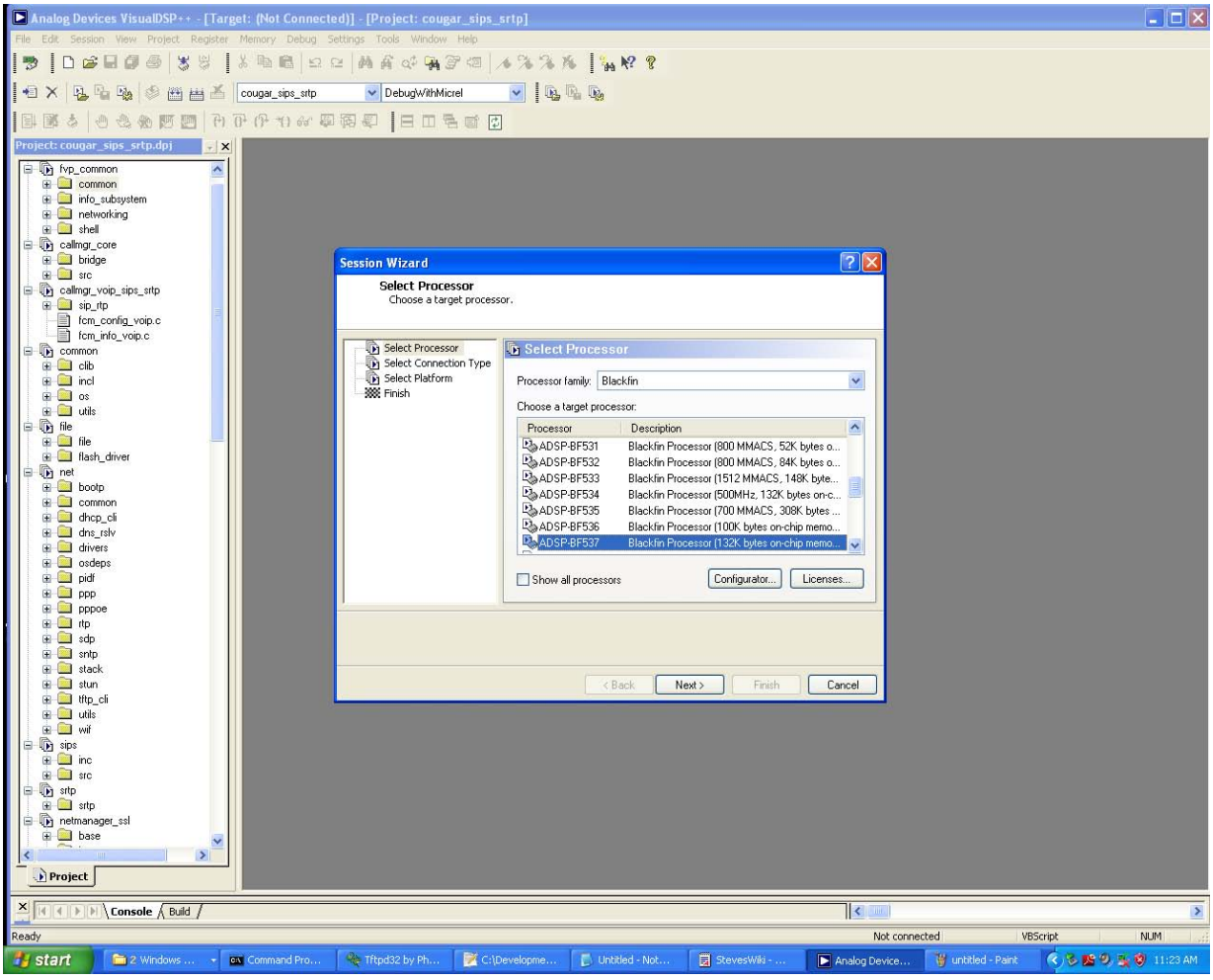


Figure 3 VDSP Session 1 Screen

2. Select the ADSP-BF537 *Blackfin* processor from the list of supported processors and then click next. Now select the **Emulator** type connection method as shown below and click next (Figure 4):

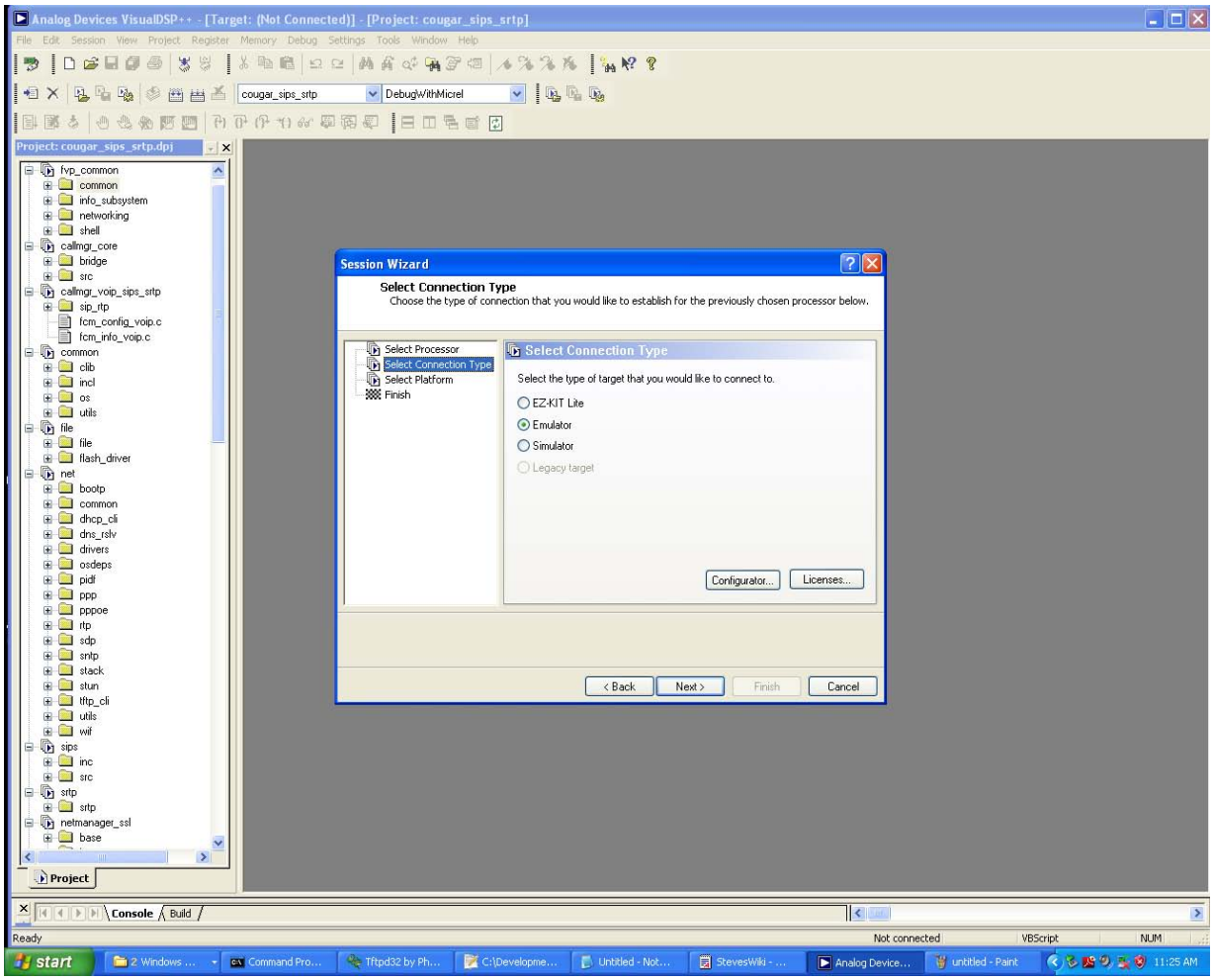


Figure 4 VDSP Session 2 Screen

3. Now give your session a unique name in the *Session Name* box shown, select it and click on the **Configurator** button as shown in Figure 5:

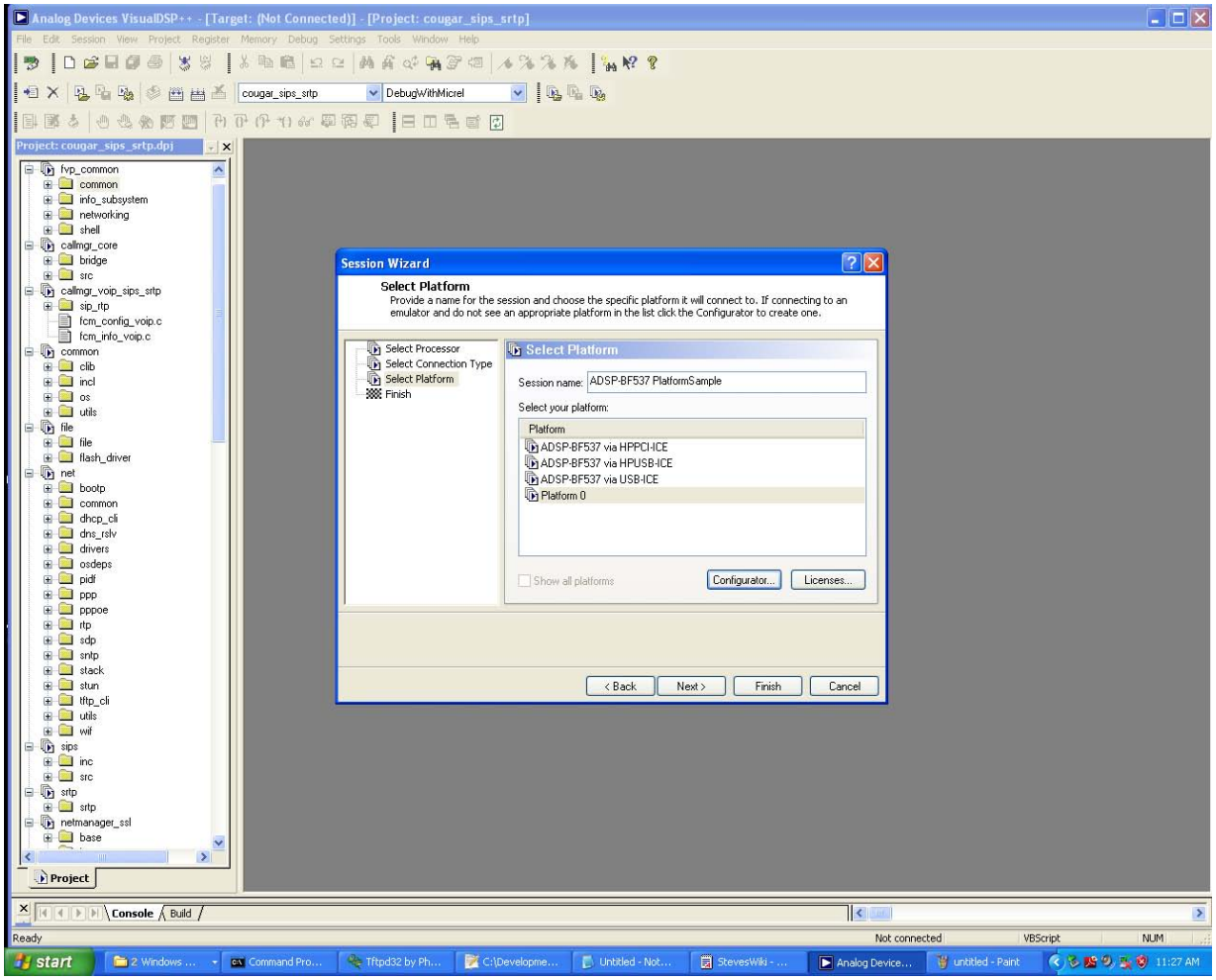


Figure 5 VDSP Session 3 Screen

4. Now click the **New** button as shown below (Figure 6):

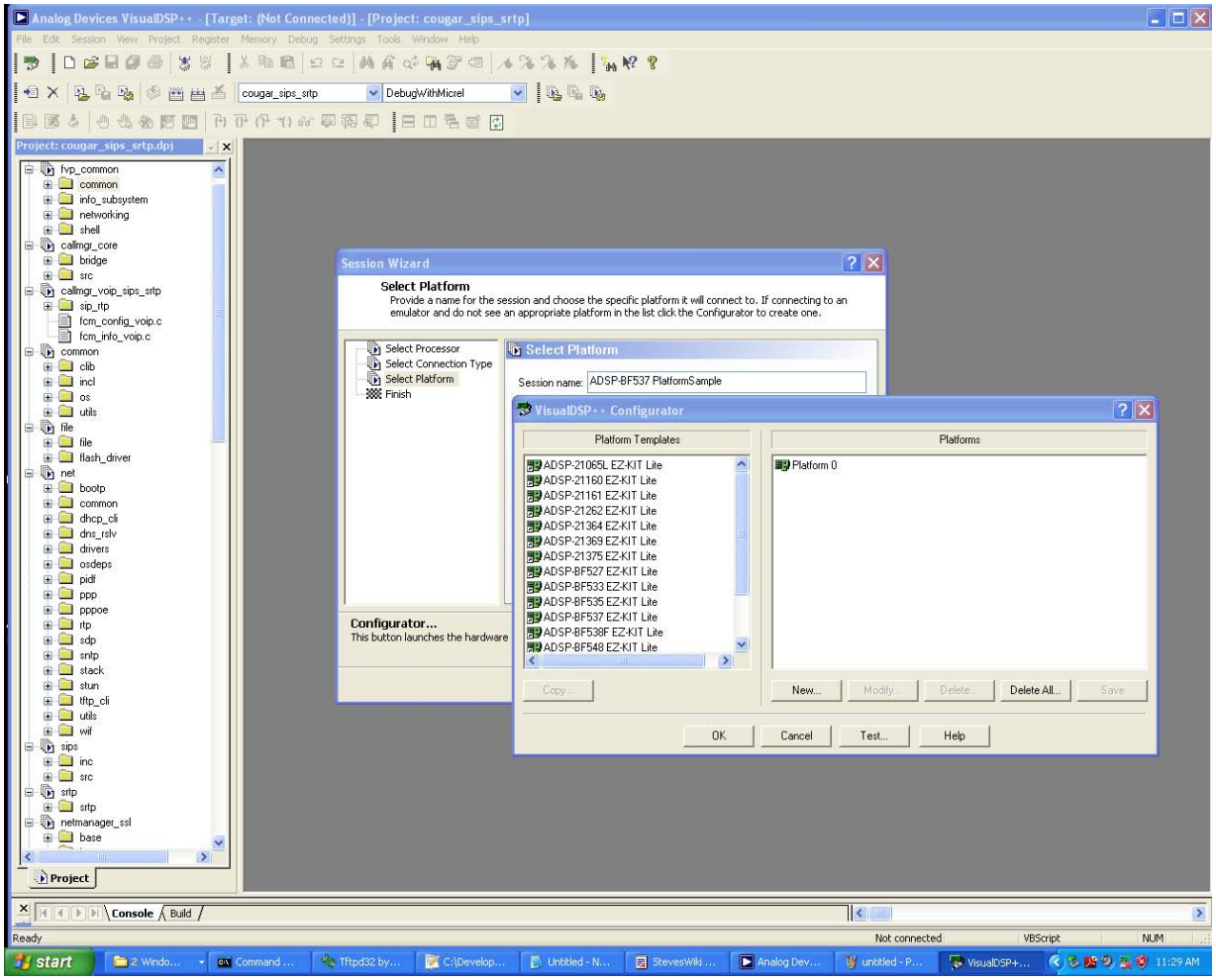


Figure 6 VDSP Session 4 Screen

5. Provide a name (e.g. **PlatformSample**), select the HPUSB-ICE connection type and then click on the **Device 0** and then select **Modify** (Figure 7):

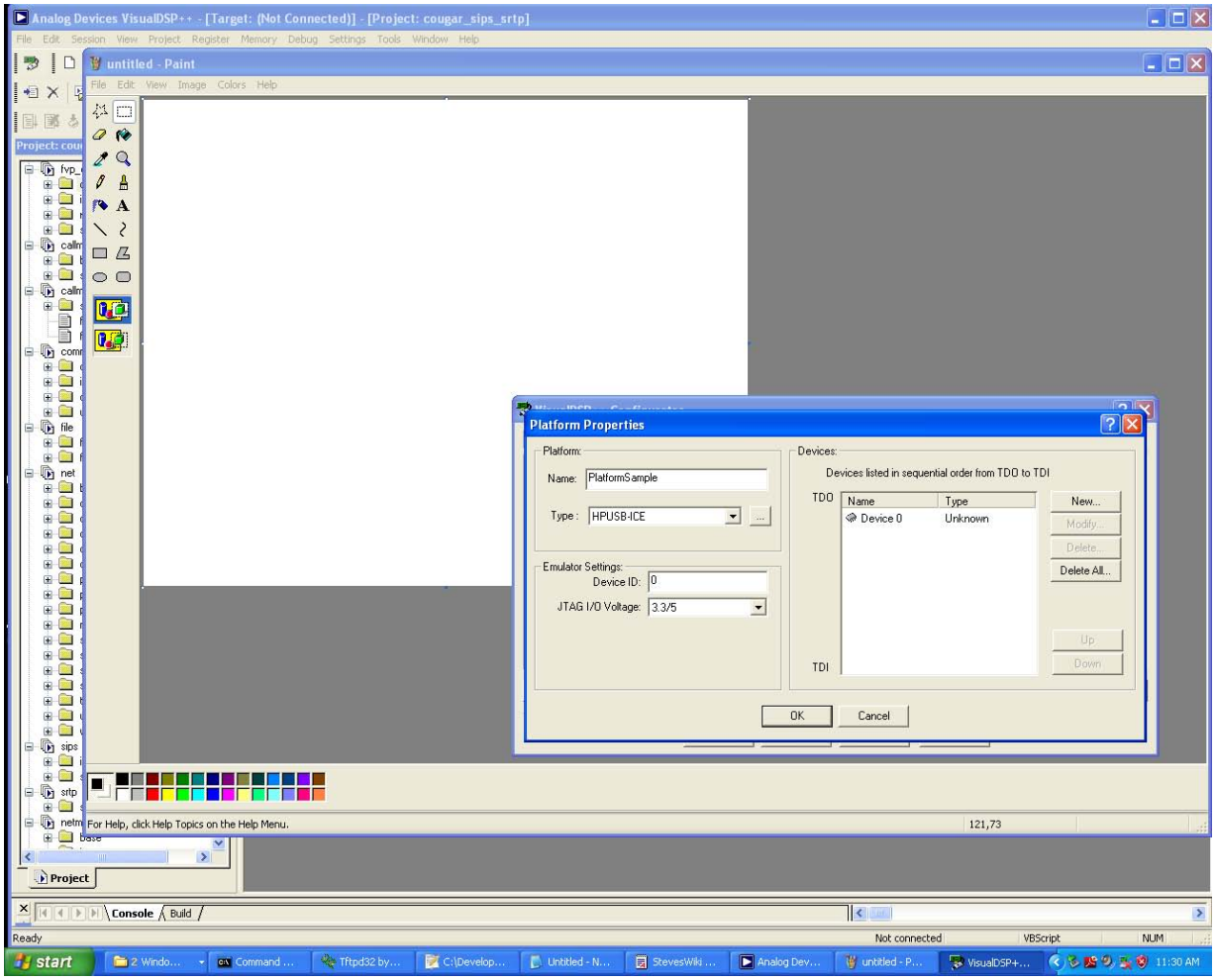


Figure 7 VDSO Session 5 Screen

6. Now change the Device Properties type to ADSP-BF537 and select the **Do Not Disturb** option (Figure 8):

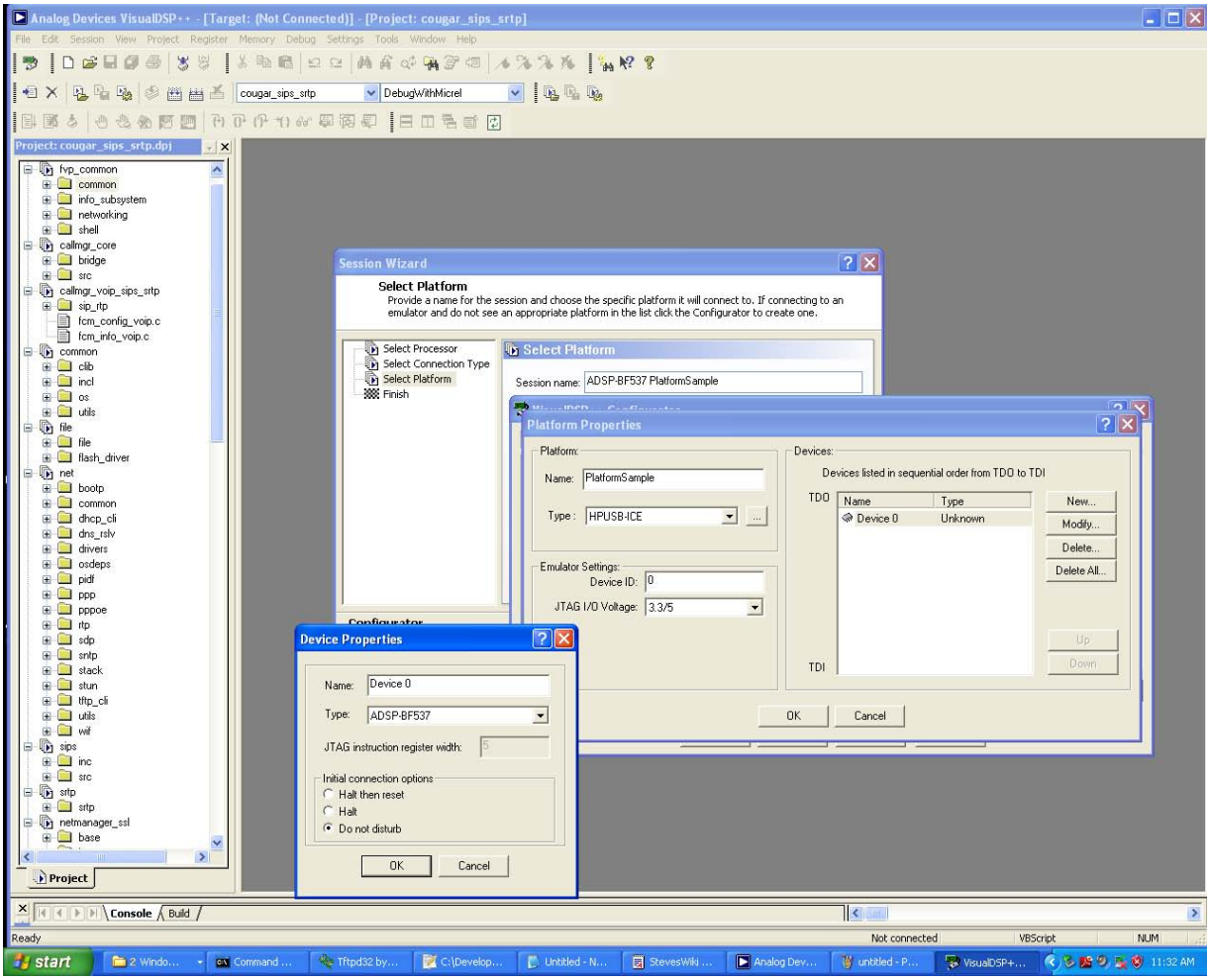


Figure 8 VDSP Session 6 Screen

7. Select **OK** to close the *Device Properties* window. Select **OK** again to close the *Platform Properties* window. Now select the newly created session platform from the list, in our example this would be the **PlatformSample**, and then select **OK** as shown in Figure 8.

8. Click finish. Now that we have created a session object for VDSP++ we can connect to our target. The target is the Fanstel ST-3116 phone. The phone must be opened to connect the ADZS-HPUSB-ICE to the phone's JTAG interface. Remove the 6 screws located on the bottom of the phone and carefully lift up the top half of the phone but no so far that you pull any of the phones wiring harnesses. The inside of the Fanstel will have a 13 pin connector that the HPUSB-ICE connector will seat into as shown in Figure 9:

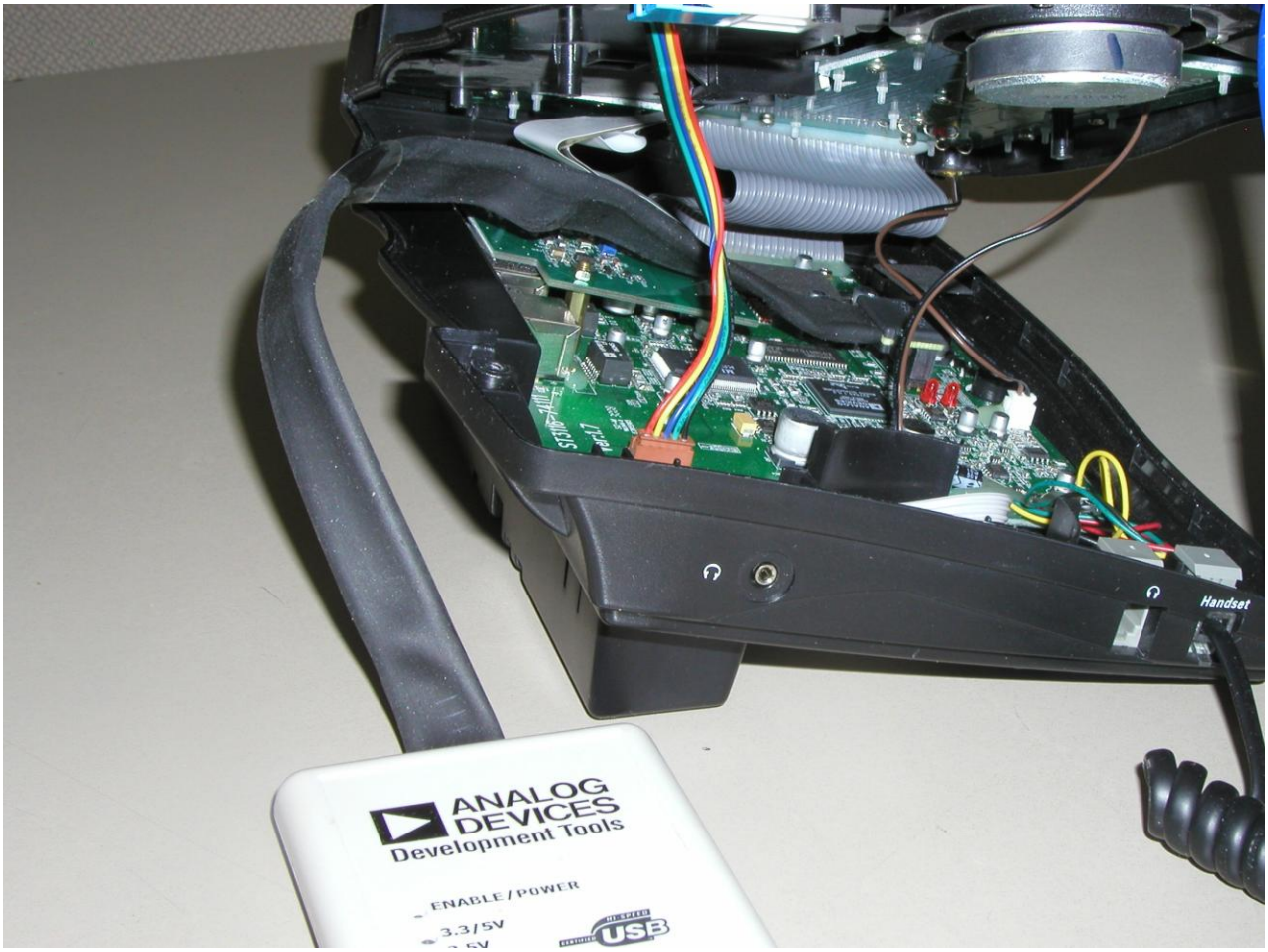


Figure 9 Fanstel Pod

9. Plug the USB end of the JTAG connector into your development box and plug in the power cord for the ADZS-HPUSB-ICE device. Now that we have our session created and physical JTAG interface we can run the AS-SIP end instrument application from within the VDSP++ development environment. Start VDSP++ if not already in it and select the **Session** and **Connect To Target** option as shown in Figure 10:

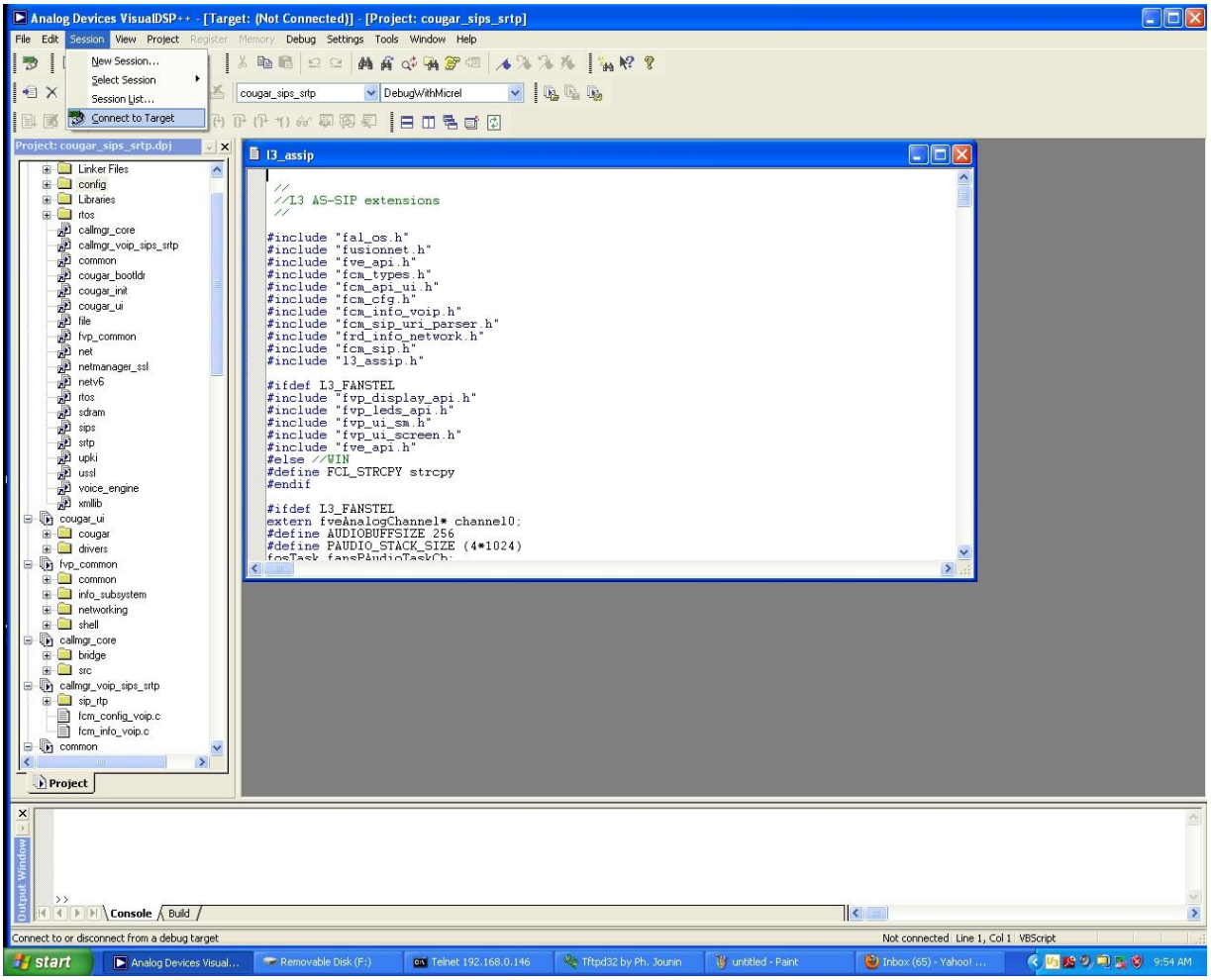


Figure 10 Connect-to-Target Screen

10. Now that you are connected to the target (Fanstel) new options will appear under the **Settings** tab. There are a few additional settings needed to make to our session object to make it work properly. Select the **Settings** tab followed by the **Target Options.** The following screen is displayed, set all of the options the same as shown in Figure 11, and be sure to uncheck **Use Xml...**

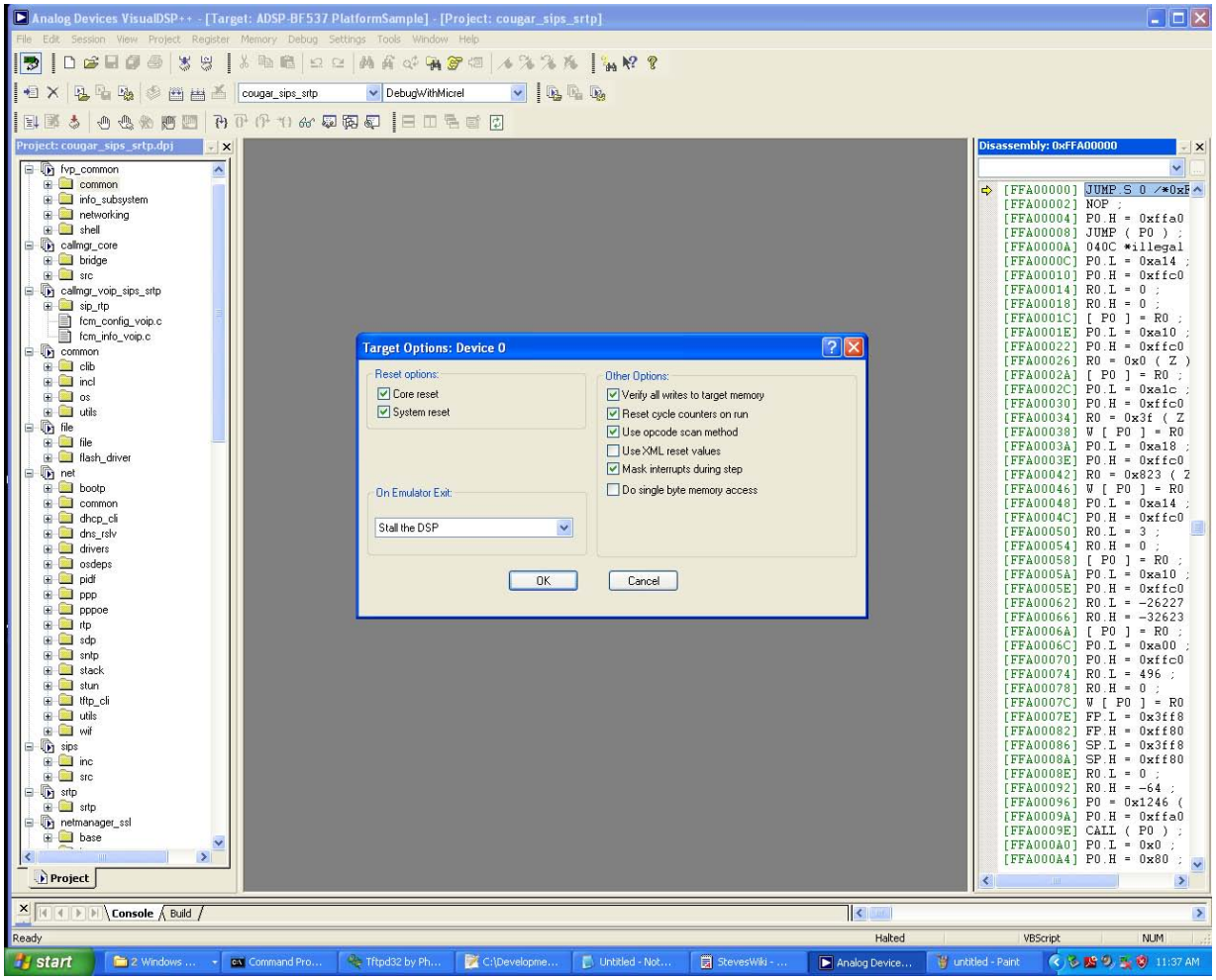


Figure 11 VDSP Targettop1 Screen

11. Next, open the **Settings Preferences** tab and under the **General** category make sure all of the settings match what are shown in Figure 12:

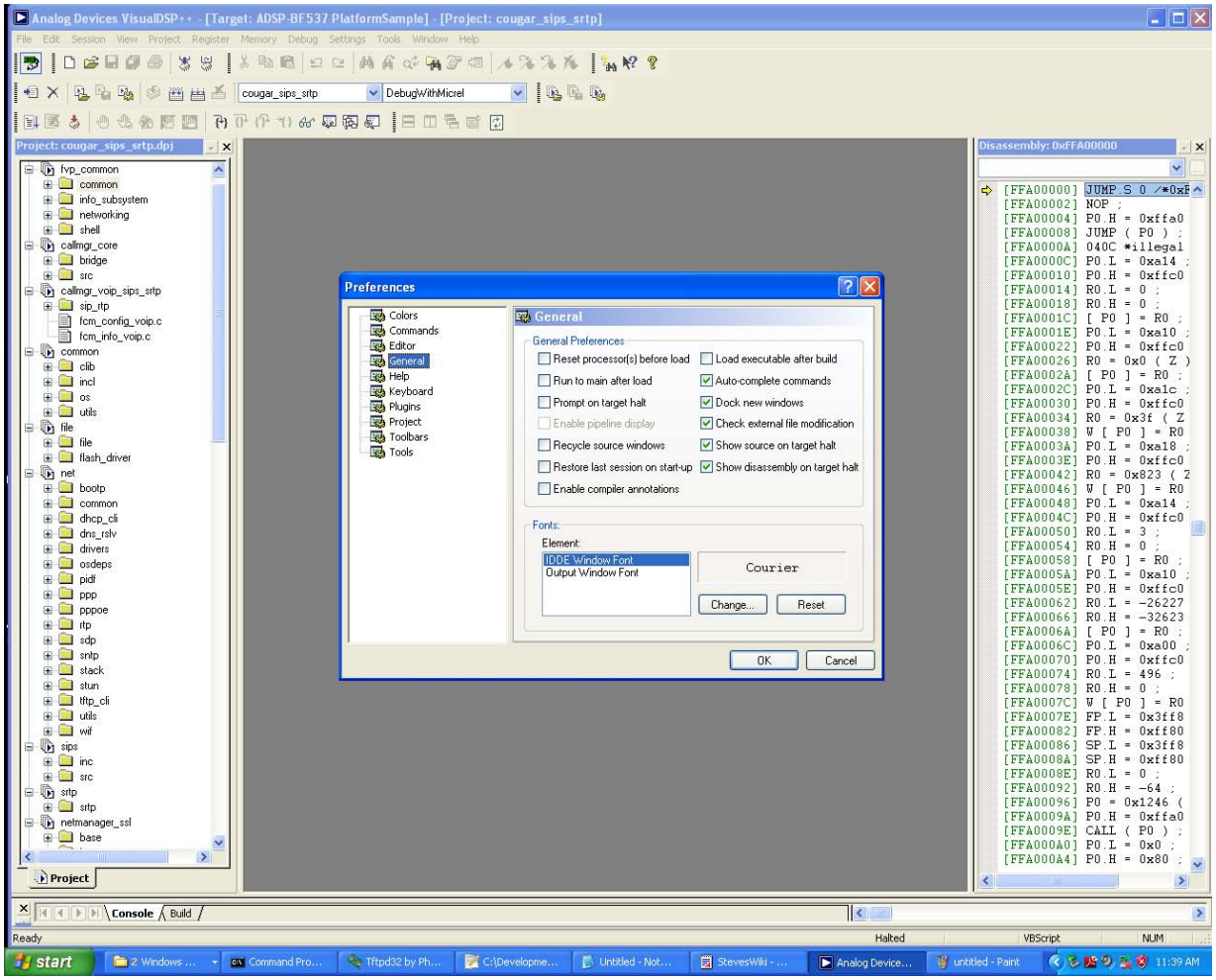


Figure 12 VDSP Pref1 Screen

12. Finally, adjust the speed of the interface by once again selecting the **Settings** tab followed by the **Frequency Selection** tab as show in Figure 13. The idea here is to use the fastest possible frequency for your development machine. Incrementally select and test each frequency to find what works for your particular development machine:

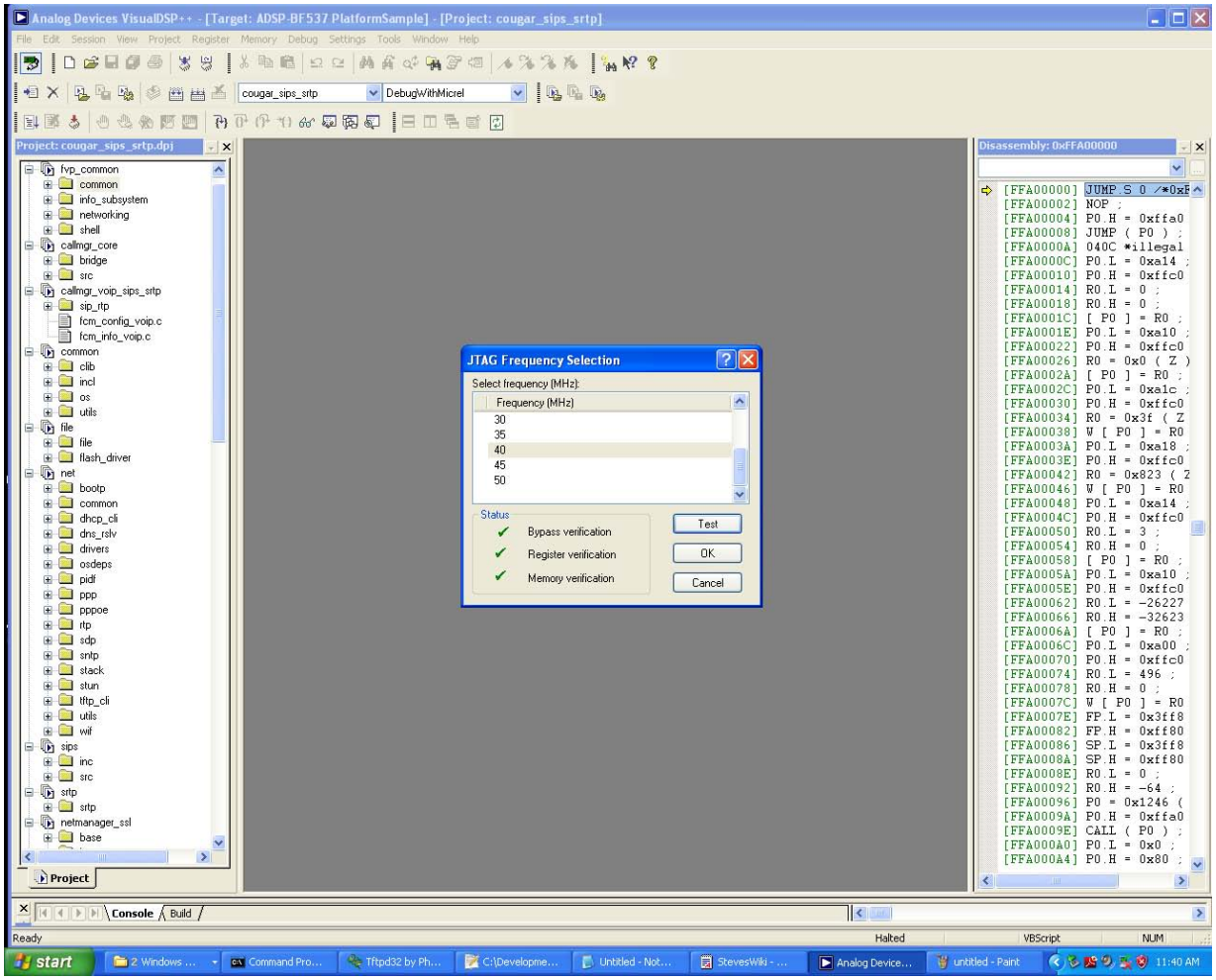


Figure 13 VDSP jtagfreq Screen

13. Note that none of the **Session** settings will appear if you are not properly connected to the target, they will all be greyed out. Now that the embedded development environment has been configured, run the AS-SIP application. In Figure 14, you'll notice 3 distinct tool bars along the top of the screen and 2 debug windows on the right side of the screen. The **Expressions** and **Disassembly** debug windows are opened under the **View** and **Debug Windows** tabs. In the **Expressions** window, enter the 3 expressions to keep track of, **$sp** (stack pointer), **$pc** (program counter) and **$imask** (interrupt masks). Halt and reset the target using the 2nd and 3rd icons from the left on the 3rd (bottom) toolbar:

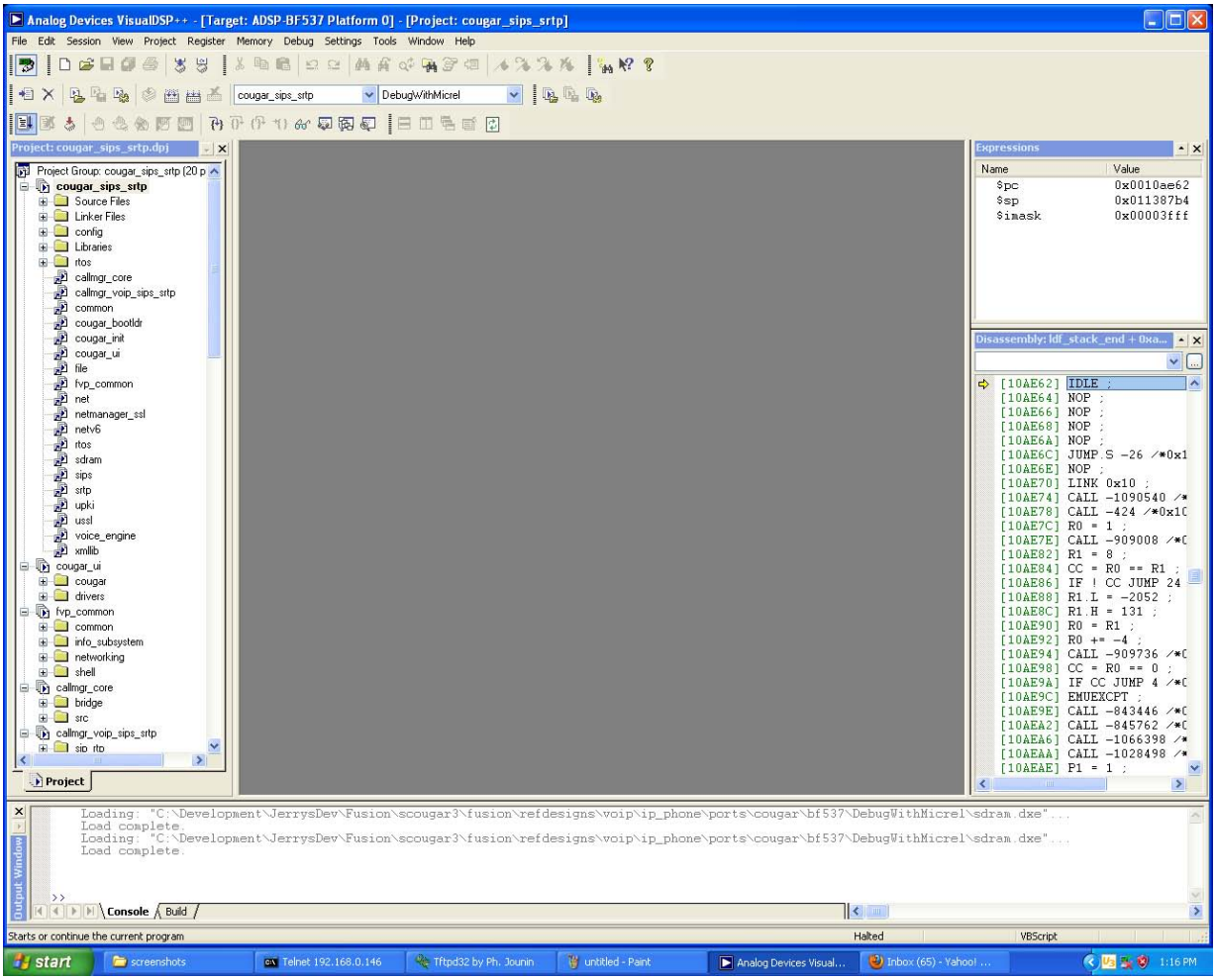


Figure 14 VDSP Run Screen

Note that if the target was not running the **halt** button (third row down, second from left) will not be highlighted. All of the buttons will tell you their function as you pass over them.

14. Next load the **sdram.dxe** program we built in the previous compile step. The **load** button is located in the 1st (top) toolbar and is highlighted in blue in the diagram, it's the 7th button in from the left (Figure 15):

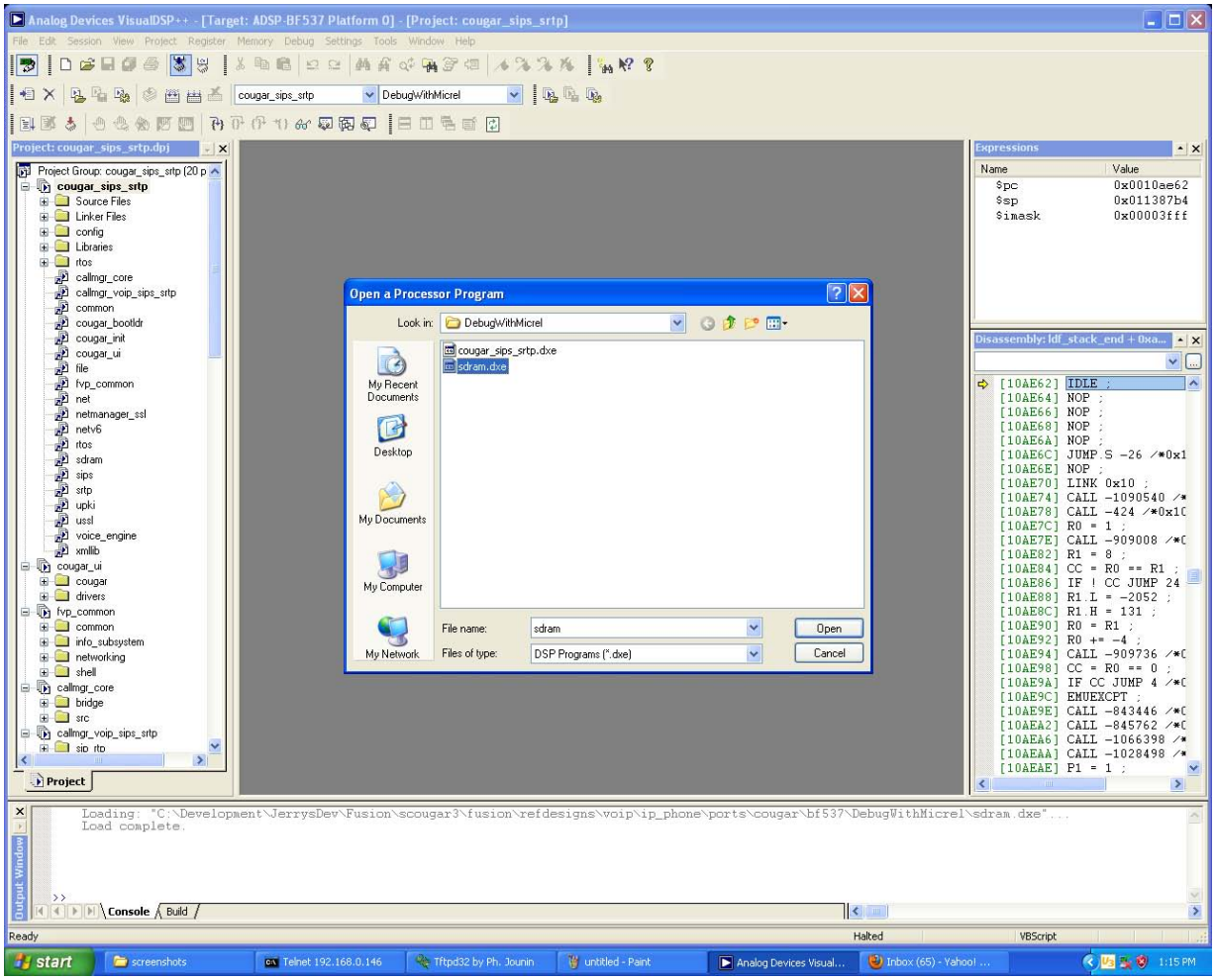


Figure 15 VDSP Load Screen

15. Select the **sdram.dxe** and **Open.** Next, run the **SDRAM.DXE**, this button is the left most button in the 3rd toolbar. Notice the value of the program counter (**$pc**) in the **Expressions** window, change the last 2 digits to **00**. So for example if you see **0x0010ae62**, change it to **0x0010ae00**. Now repeat the load steps from above except load the **cougar_sips_srtp.dxe** program this time. This is our AS-SIP application. After the program loads, and this could take up to a minute because of the size, click on the **run** button (bottom left icon) and you should see the following screen (Figure 16):

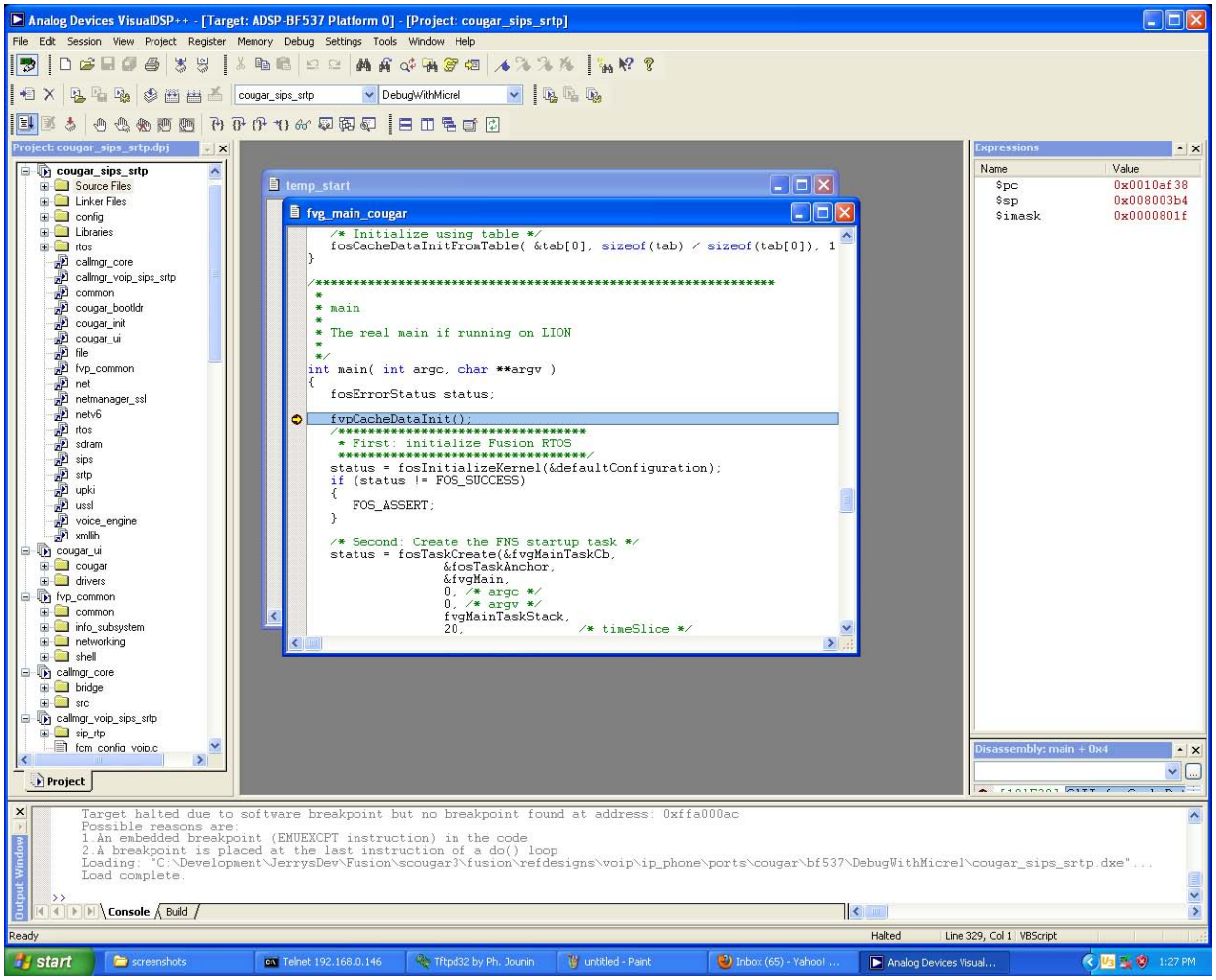


Figure 16 VDSP Run Cougar SIPS SRTP Screen

16. Notice that the program stopped at the **fvpCacheDataInit()** function in the display window. This is a hard break point and was purposely added. Click the **run** icon again to finally start the application, watch the display on the Fanstel phone as it boots up. The phone is now running under the jtag interface. All of the normal debugging features are readily available prior to starting the application. Any source file may be opened in the display window and break points set by clicking on the left border of the display window on the desired line of code. Should the program crash, set the **$imask** expression to **0x0** and take a single **step** in the execution, this button is found in the 3rd (bottom) tool bar and is the 9th button in from the left. The status of the program is always visible on the bottom of the screen, **running, halted** etc...

1.3.5 Installing and Running Image From Memory

The previous process for building and running the application from VDSP++ is quite lengthy and cumbersome to use but can be invaluable when software related problems arise as it is the only way to debug the program. A much simpler way to run the application is to install the image on the phone which is how it would be done in a production environment. The previously discussed process for compiling and linking the application is no different for this method of running the AS-SIP end instrument. The biggest difference in the build is you do not need to connect to the target, we'll simply be loading and running the image directly on the phone.

Burning new images onto the phone permanently is also a custom procedure developed by *Unicoi*. The previous section used the **sdram.dxe** and **cougar_sips_srtp.dxe** images to run the phone. Here we'll be using the **cougar_sips_srtp.ldr** image that was also generated when compiled and built the **cougar_sips_srtp** project. Before upload and install the **cougar_sips_srtp.ldr** image must be run through a *Unicoi* custom program to create a **.bin** file. The bin creator program is a Windows console program called **UNicoiReleaseCreator** and is used to create a permanent image for the phone. From a DOS command line window you would issue the following command as shown in Figure 17 below:

C:\$UNICOI_INSTALL_DIR\$UNICOI_IMAGE_DIR> UnicoiReleaseCreator cougar cougar.ldr scougar3.bin

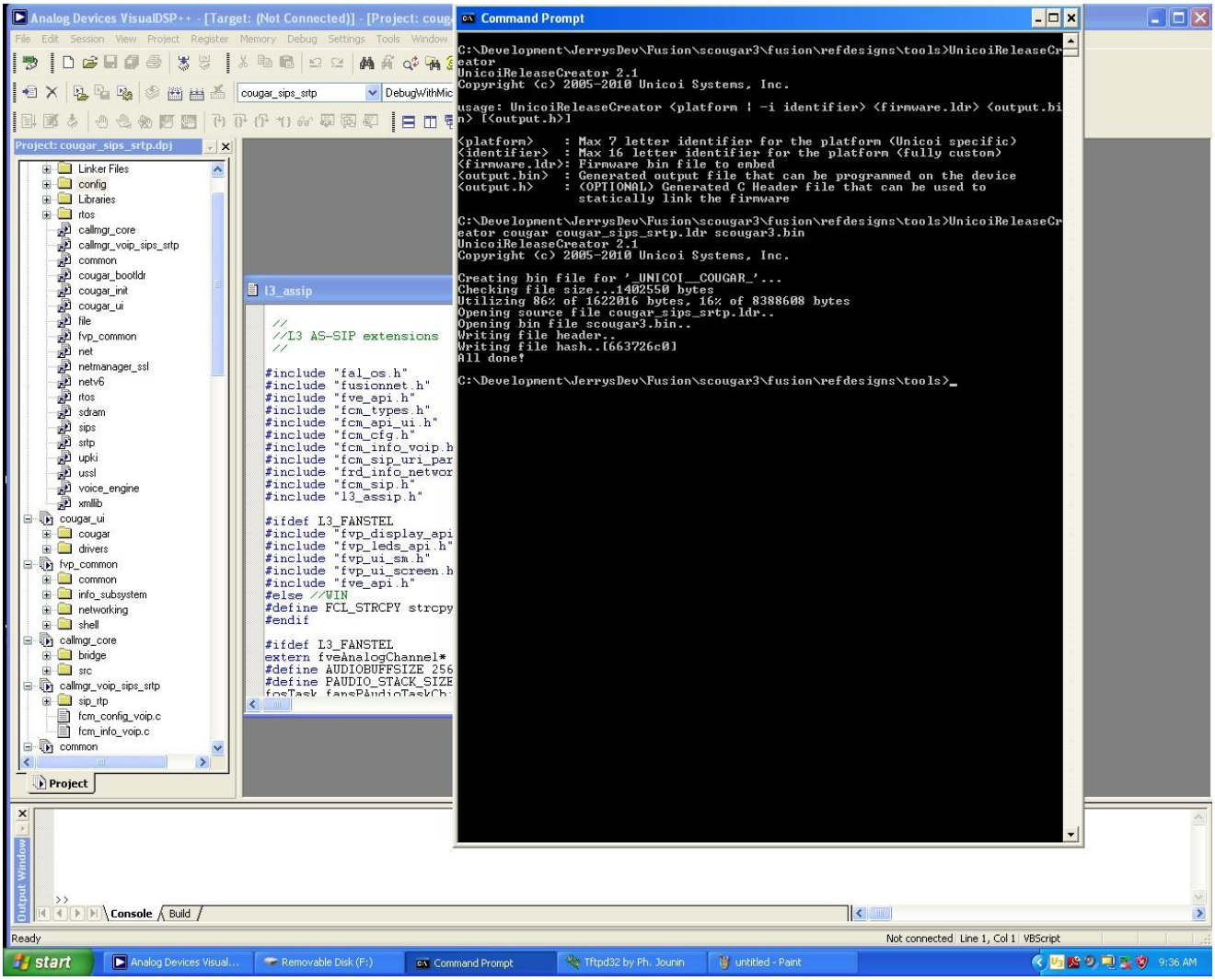


Figure 17 Create Image Screen

The above diagram shows a usage message for the **UnicoiReleaseCreator** program followed by the actual creation of the .bin file: **scougar3.bin.** The name chosen is not important and simply reflects that this binary is from the 3rd version of the secure *Unicoi* SIP stack release. Next, load the **scougar3.bin** file on to the phone. This is done via a custom *Unicoi* telnet shell command: l**oad**. Using the IP address of the phone, telnet into the phone and issue the following command as shown in Figure 18:

C:\$UNICOI_INSTALL_DIR\$UNICOI_IMAGE_DIR> telnet [phone IP address]

Next, from the custom *Unicoi* telnet shell, issue the following command as shown in Figure 18. Note that the **-h** flag is for the IP address of the development machine, an tftp server needs to be

running on the development machine and pointed to the location of **scougar3.bin**:

load -h 192.18.0.123 cougar.bin

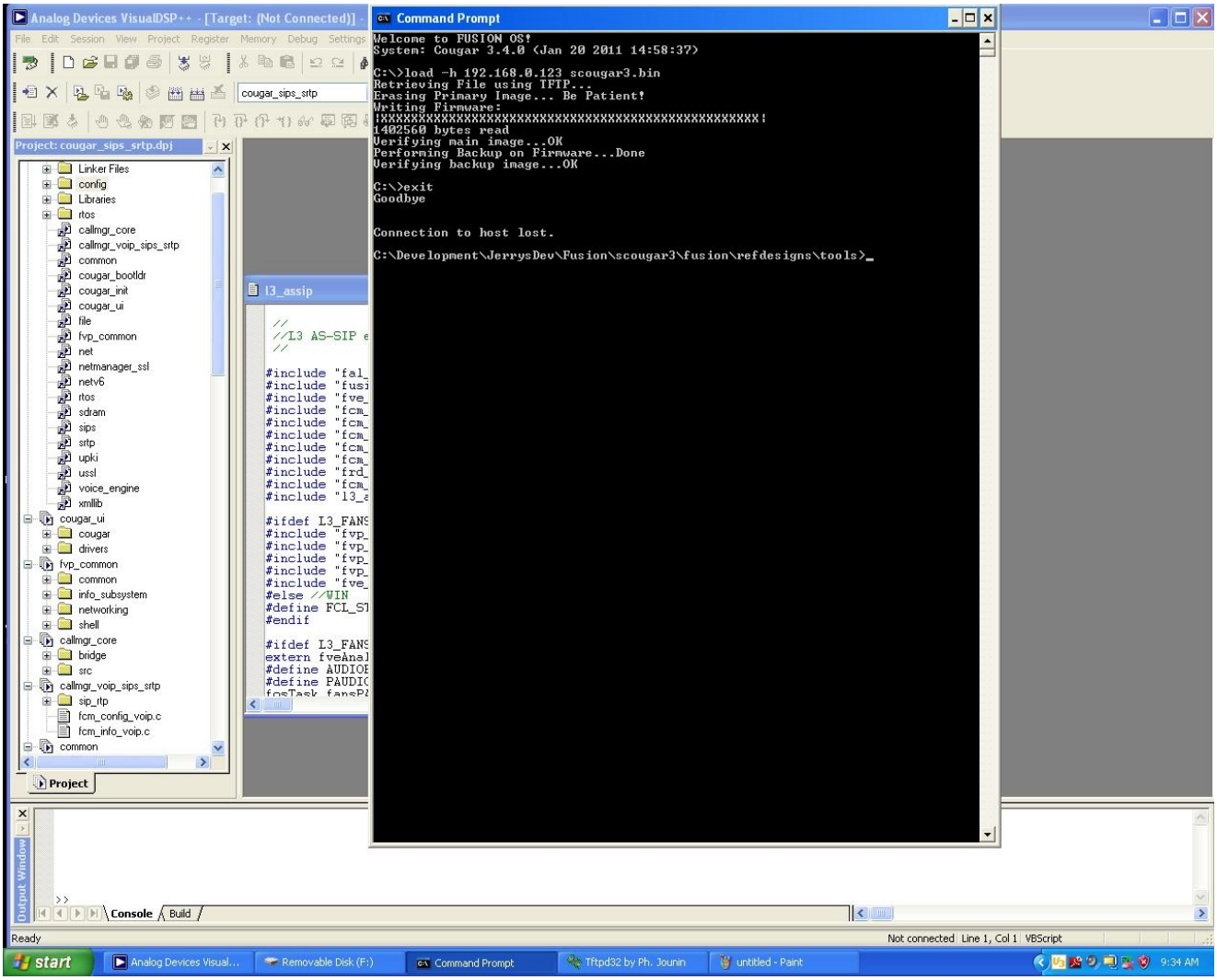


Figure 18 Load_Binary Screen

The previous step has now loaded the image permanently into flash memory so that it will be run every time the phone is powered up. Note also that burning the image into flash is not always successful. When a load fails, simply reissue the load command. A bad load is shown in Figure 19:

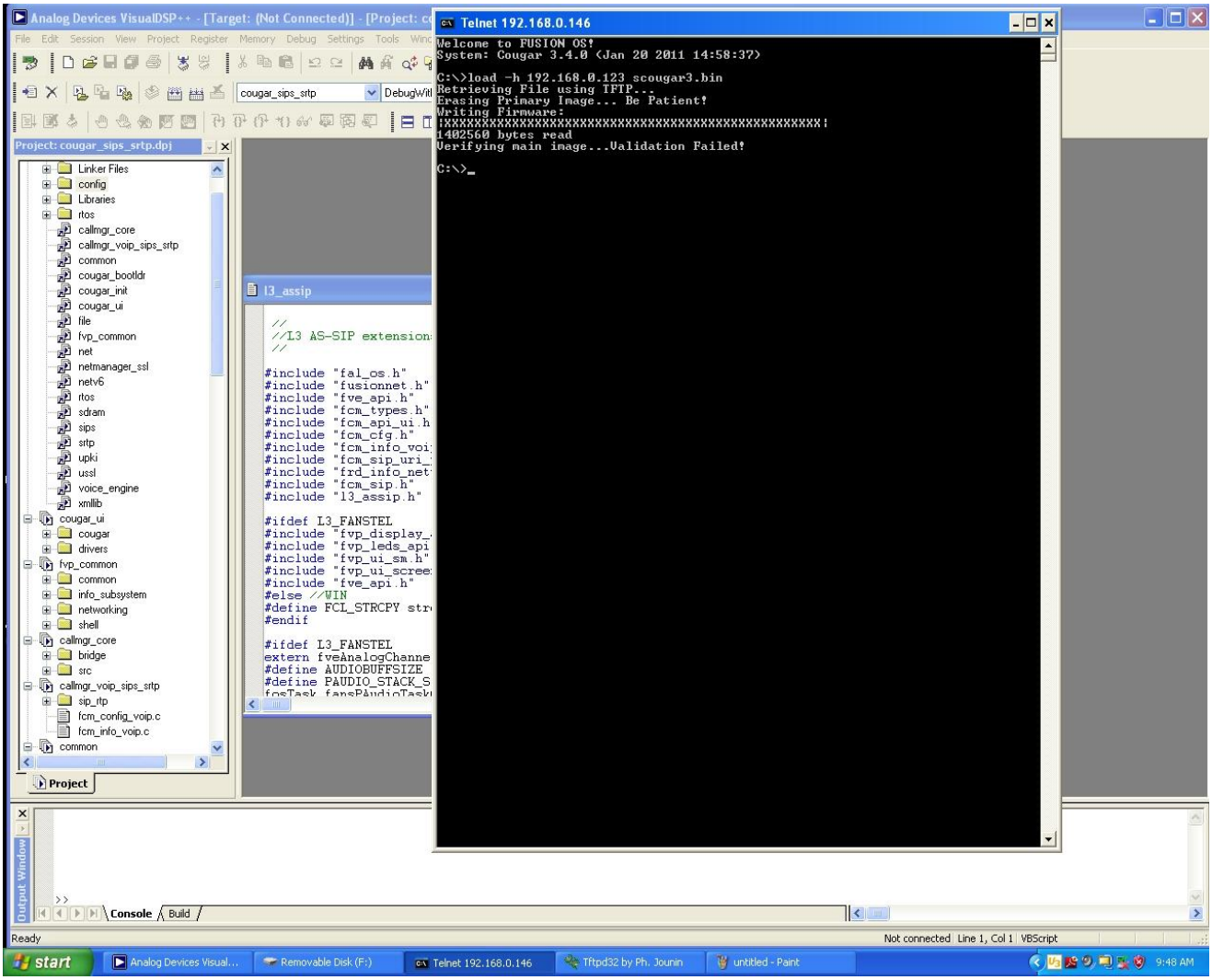


Figure 19 Load_Binary_Fail Screen

Again, the IP address above is the host IP address of the development machine, not the phone. It is not unusual for the firmware installation to fail. You must verify that the firmware was successfully installed via the output display. Repeat the command as necessary until the software is successfully installed. The phone must be rebooted for the newly installed image to take effect. The phone has both a soft reboot feature or you can simply unplug the NIC cable momentarily. To reboot the phone from the Unico telnet shell, issue the **.advanced** command followed by **reboot**.

After rebooting the phone, follow the progress of the phones' restart from the display. If a programming error was made or a possible bad burn installing the image, the only way to correct the problem is via the VDSP++/JTAG environment since there will no longer be any means of

interfacing with the phone.

1.3.6 Configuring the AS-SIP End Instrument

For the phone to work properly it must be configured to work with a proxy server of some kind or perhaps peer to peer. Our AS-SIP phones were all configured to use the REDCOM SLICE 2100 switch configured with what is called a military load.

The phones can be configured via any WEB browser interface application such as Windows Explorer or Mozilla. Enter the phones IP address in the URL of a browser: example: http://192.168.0.121. You will be prompted for the administrative user id and password which was set to **admin** and **admin** on our phones. The following screen (Figure 20) will be displayed:

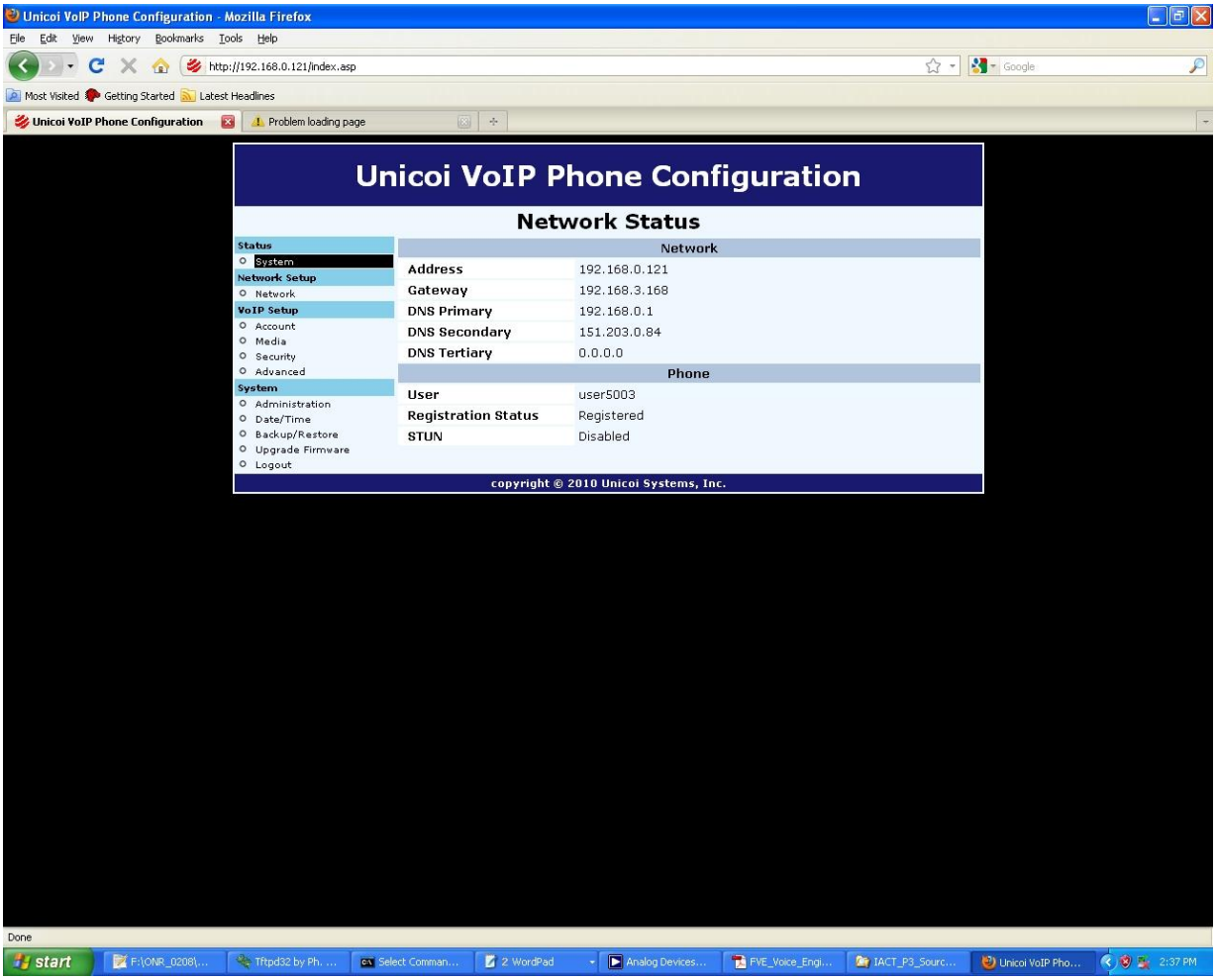


Figure 20 Main_Config_Screen

Note that this configuration utility is a convenience only. The browser interface makes changes to an XML configuration file that could also be modified through most text editors and then loaded via *Unicoi*'s custom telnet shell using the **putfile** and **getfile** utility commands. This could also be done via a simple script from the host. It is important to note that the configuration file is also burned into flash memory so that the setup is recalled (persistent) when the phones are power cycled. *Unicoi* has a small file system implemented in flash memory.

The **Account** configuration is the most critical. Under the VoIP setup, select **Account** and the following configuration screen (Figure 21) is displayed:

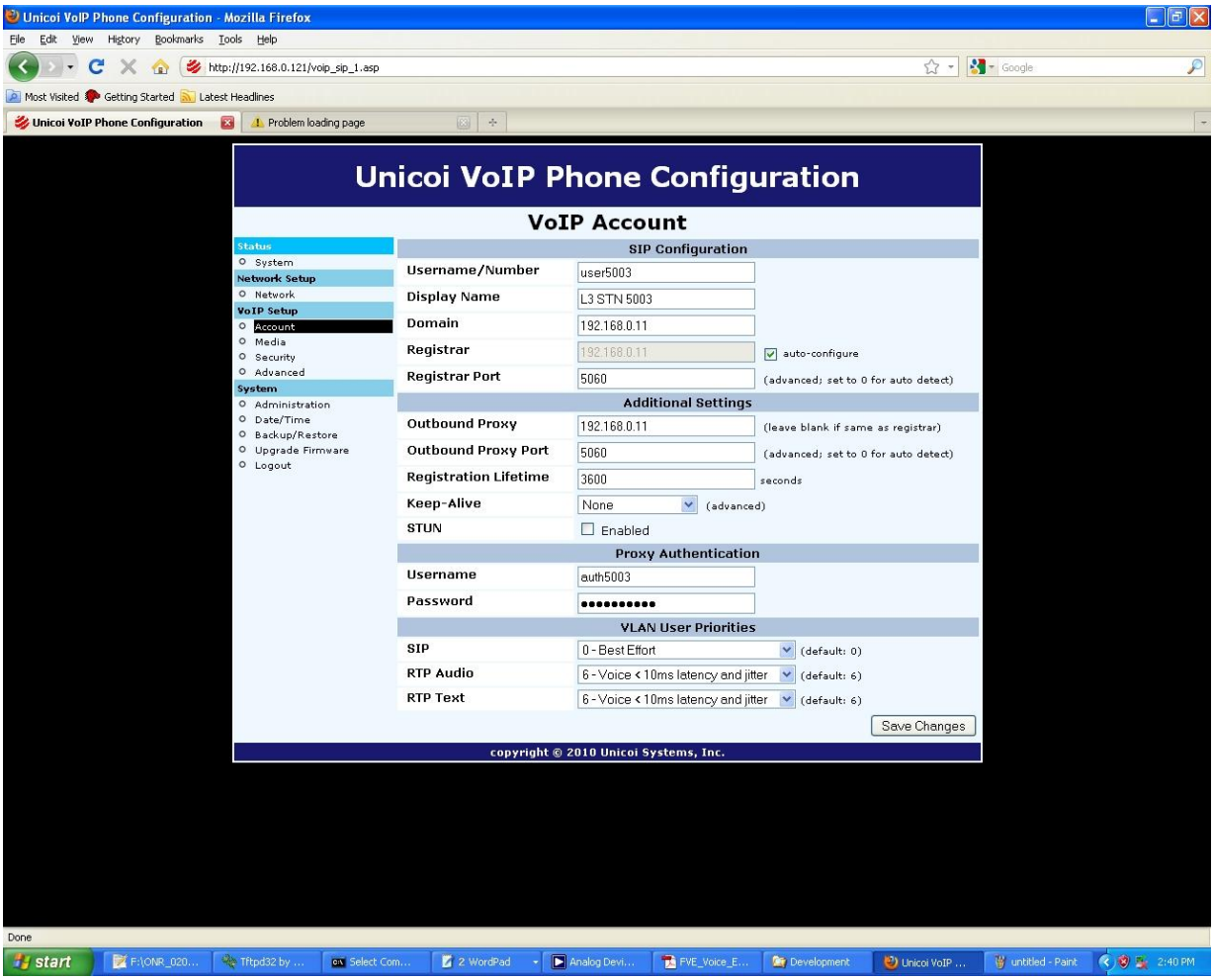


Figure 21 Account_Config_Screen

The **Username/Number** is a combination of a configured DSTN station number on the REDCOM switch, in this case **5003** prefixed with the word **user.** The **Display Name** can be set to anything. The **Domain** is set to the IP address of the REDCOM switch. We did not use an **Outbound Proxy** but setting to the address of the *Unicoi* switch does not hurt anything, it could otherwise be left blank for now. The **Username** and **Password** must be set to how ever they were configured on the REDCOM switch. These are used for all authentication challenges. The **Registration Lifetime** is the amount of time in seconds, to reregister. We did not use the *Unicoi* **Keep Alive** feature because it generates too much traffic however it does work by repeatedly sending either **REGISTER** or **UPDATE** requests. **STUN** was not used for our purposes but would otherwise allow the application to discover the presence of a network address translator to obtain IP addresses.

The **Media** configuration allows you to select which codecs to support and in order of preference. This is determined by what the LSCsupports, in our case the REDCOM switch. The codecs shown below on the left side are all of the audio codec formats supported by *Unicoi*. G.711 uLaw was our preferred audio format with REDCOM. **ULaw** incidentally is the commanding algorithm used primarily in North America and Japan. The rest of the world uses **aLaw** commanding for E1 type circuits (Figure 22).

Figure 22 Media_Config_Screen

The **Security** configuration allows you to enable various levels of security within the *Unicoi* IP stack. In the configuration below, SRTP and secure mode are all disabled. The list up supported encryption algorithms is also shown and as with the audio codecs they are listed in order of preference.

Figure 23 Security_Config_Screen

The **Advanced** configuration is used to setup STUN if it was enabled in the **Account** configuration and to set **Accept All** for security related certificates.

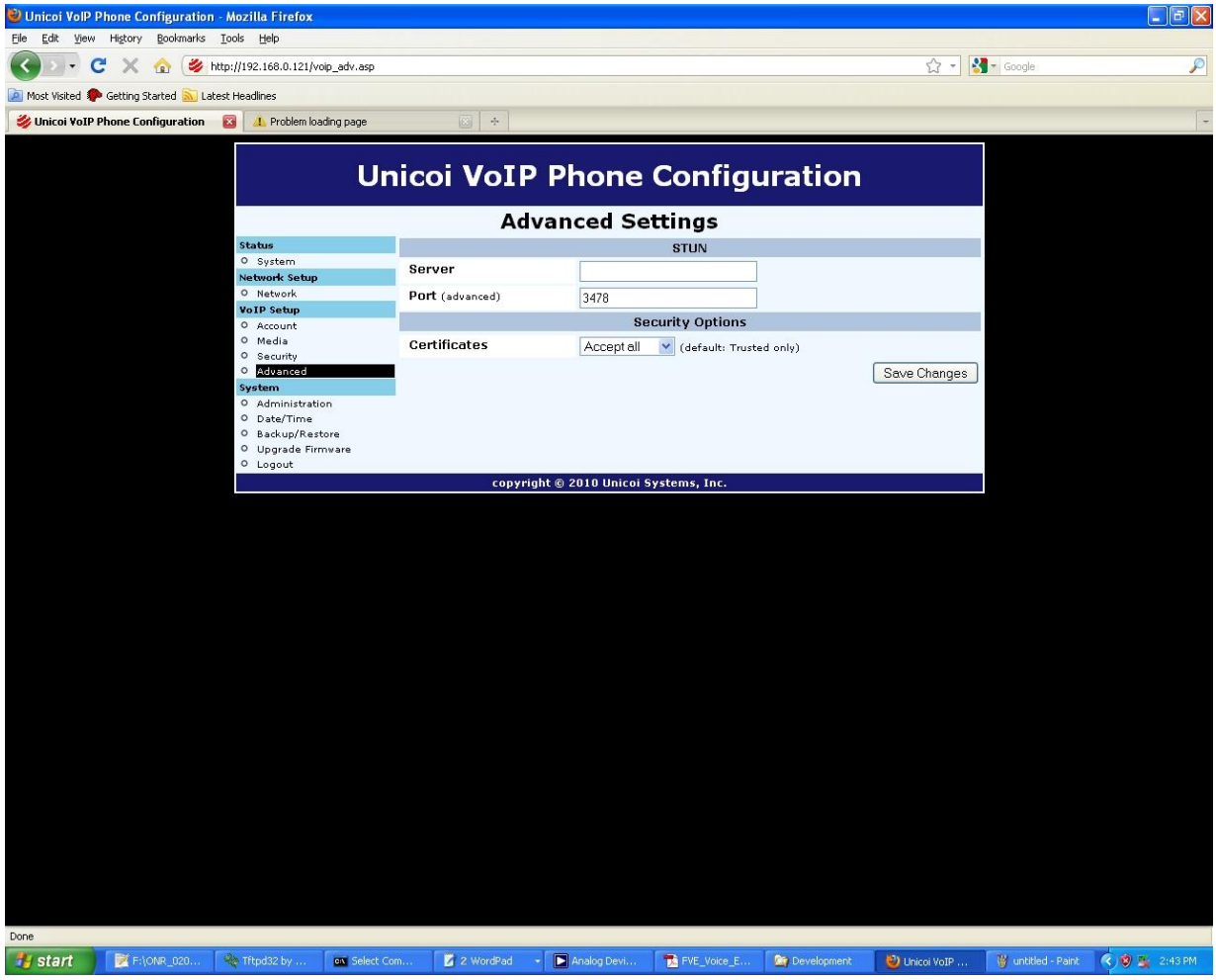


Figure 24 Advanced_Security Config_Screen

The **Administration** configuration is used primarily to set the username and password for access to the phones configuration. Again, set all username/passwords to **admin/admin** and never use this again as it has nothing to do with the AS-SIP client application. The screen does display some other useful information such as the phones MAC address and software version.

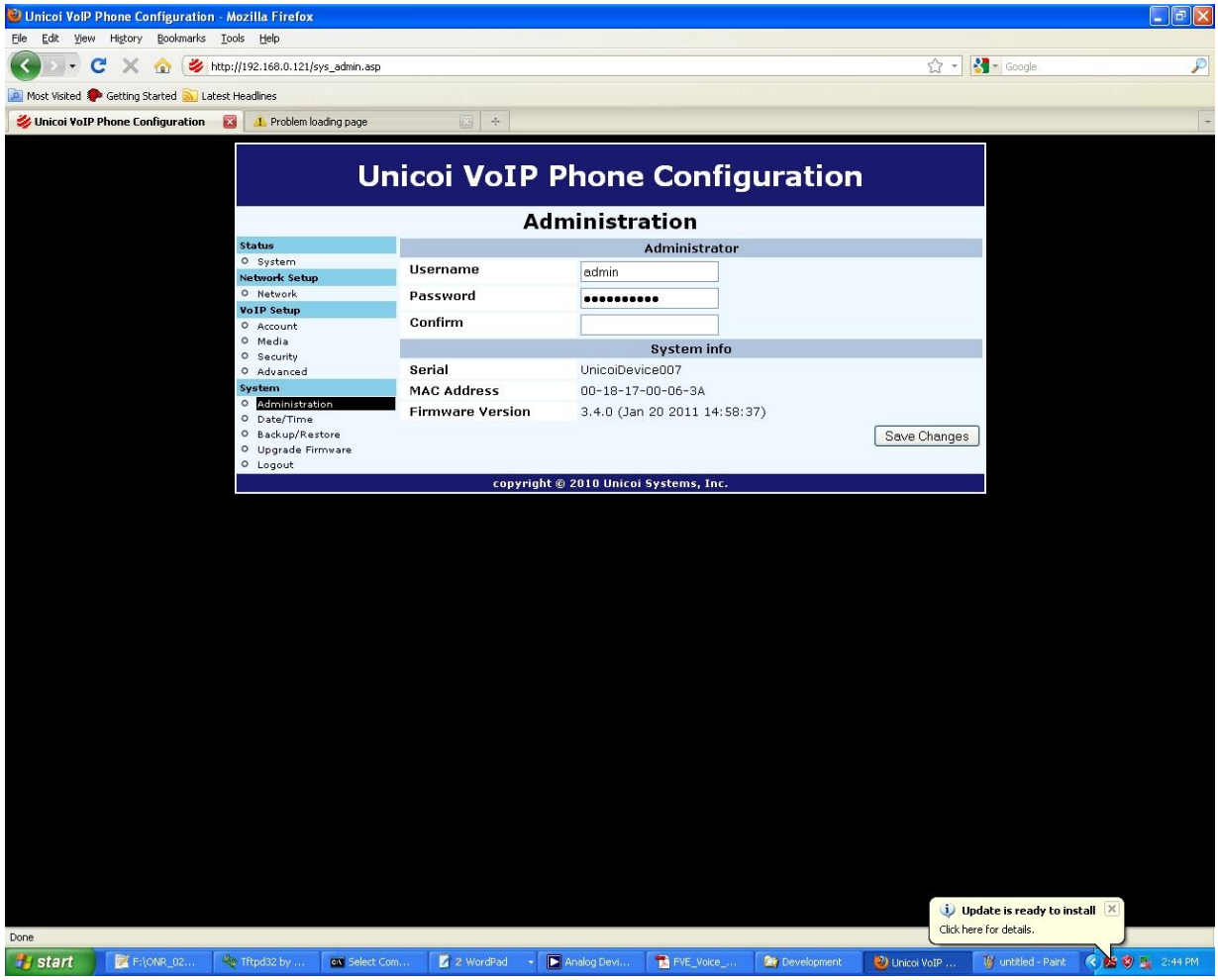


Figure 25 Admin_Config_Screen

The **Date./Time**, **Backup/Restore** and **Upgrade Firmware** configuration screens are not used.

1.4 *Unicoi* SIP stack

1.4.1 Unicoi Overview

The **InstaVoIP** family adds VoIP capabilities to any product. Available pre-optimized for select platforms or with full source code for porting to any platform, **InstaVoIP** provides a robust VoIP core with a flexible API which allows customization into nearly any end product.

NOTE: InstaVoIP is an evolving product with active, on-going development. Roadmap items are marked with this icon: ; please contact us for more information on the current status of these items.

1.4.2 InstaVoIP Family

1.4.2.1 InstaVoIP Embedded

A full ANSI C source code release allows **InstaVoIP** Embedded to be used on any platform. Simply porting the Fusion Common Layer (FCL) abstraction to a platform's RTOS/OS, network stack, and file system (optional), and implementing an audio driver channel for the platform's audio hardware is all that is necessary to start making VoIP calls. Additionally, having full source code allows customer changes to be made to the code (such as implementing newer or less popular RFCs), and aids in debugging low-level problems.

An optimized library release for Windows, Linux, and Mac OS desktops allows for rapid integration without dealing with any porting or audio issues. Standalone SoftPhone application examples are provided, or the libraries can be integrated into existing applications.

1.4.2.2 InstaVoIP Mobile

An optimized library release for iOS, Android, and Windows Mobile environments allows for rapid integration without dealing with any porting or audio issues. Standalone SoftPhone application examples are provided, or the libraries can be integrated into existing applications.

1.4.2.3 InstaVoIP Module

A hardware/software combination, **InstaVoIP** Module is a standalone hardware module ready to be dropped into an existing hardware platform or used as the foundation of a new product. Based on the powerful Analog Devices *Blackfin* processor, **InstaVoIP** Module includes an Ethernet connection, 4 MB of firmware and file system storage and a 48 kHz capable stereo audio codec. The efficient Fusion RTOS, robust Fusion Network TCP/IP stack, and InstaVoIP Embedded are pre-integrated; out-of-the-box the **InstaVoIP** module powers up and is immediately capable of creating 5-way conference calls using the G.729 codec and while doing so it still has enough free processing power to run user applications.

The software for **InstaVoIP** module is available in two varieties: a combination object/source code release, or a full source code release. In the combination release, the majority of the core code is provided in object format but key sections are provided as source to allow for user expansion. The full source code release is for customers that typically need to modify the core code to implement new functionality, or for those customers who prefer to have the source for debugging purposes.

1.4.2.4 Example Applications

There are virtually no limits to how **InstaVoIP** can be used to create VoIP-capable applications. Some example applications are:

SoftPhone

InstaVoIP can easily be made into a standalone SoftPhone application. Example projects are included in the Embedded, Desktop, and Mobile versions which provide instant functionality and are ready for customer differentiation.

Desktop Phone

InstaVoIP Embedded or Module can be used to create enterprise-ready desktop phones that interoperate with the most popular PBX solutions in the market.

Call Box

InstaVoIP's User Interface API allows for traditional interfaces as well as highly differentiated user interfaces such as a Call Box with its simpler "press this button in case of emergency" interface. Various aspects of a call workflow can be controlled by a user or controlled by custom code which allows for the implementation of any required UI.

FXS/FXO Analog Gateway

InstaVoIP can be used to create an FXS Gateway (where analog phones plug into the device) or an FXO Gateway (where the device plugs into an existing POTS line) thanks to the FXS/FXO workflows included in the InstaVoIP Call Manager.

RoIP

Radio-over-IP is a growing market that gives traditional radios connectivity into corporate PBXs and/or traditional POTS lines. **InstaVoIP** Embedded or Module can be used to quickly create devices that bridge these environments for military, public emergency systems, or businesses that use radios in their daily environment.

Analog-to-Digital Transition

There are many products in the marketplace today that have analog phone interfaces and these products need to be modernized for use VoIP. With **InstaVoIP** you can quickly add VoIP where it can replace the existing analog interface or work alongside it to provide a smooth technology transition.

Add-On VoIP Capabilities

InstaVoIP can be embedded into an existing application to provide VoIP capabilities. Use the Desktop or Mobile versions if integrating into an application on one of the supported platforms, or use the Embedded version for any other platform.

1.4.3 Features Overview

InstaVoIP includes a core set of basic features and optional features that enable further customer differentiation:

- Core VoIP Networking Protocols
  - SIP, SDP, RTP, STUN
  - *Optional*: SIPS, SRTP for secure communications

- Call Management
  - *Supported Workflows*: SoftPhone, Desktop Phone, POTS FXS, POTS FXO
  - *Actions*: place calls, answer calls, disconnect calls, on/off hold, transfer, conference, generate DTMF, etc.
  - *Events*: incoming call, peer on/off hold, peer disconnect, being transferred, DTMF received, registered/unregistered, etc.
  - Call management control via HTTP/JSON-based web service for remote control

- Voice engine
  - *Codecs*: G.711, G.726 (16/24/33/40 kbps), G.722, DVI4 (narrow/HD/Ultra HD), Linear PCM
  - *Optional Codecs*: G.729, iLBC, G.723, Lockheed TDVC
  - *Algorithms*: Gain, Automatic Gain Control (AGC), DC Blocker, High-Pass Filter, Voice Activity Detector (VAD), Acoustic Echo Suppressor, Sample Rate Conversion, DTMF (Generator/Detector), Call Progress Tone Generator, Custom Ring Tone Generator, Comfort Noise Generator, Packet Loss Compensation
  - *Optional Algorithms*: Custom Tone Generator, Acoustic Echo Canceller, Line Echo Canceller (currently Module only), Noise Reduction, Frequency Equalizer

- Info Subsystem
  - Configuration Information Management
    - File-based by default
    - Can integrate with platform's configuration style
  - Runtime Information Management (e.g. call status)

  - Access via HTTP/JSON-based web service for remote configuration and status monitoring

- Web-Based Configuration UI
  - *Optional*: HTTPS for secure access
  - Expandable to include user-application configuration

- Module Only
  - Includes **InstaVoIP** Embedded, plus:
  - Fusion RTOS
  - Fusion Embedded™ TCP/IPv4 Networking Stack, ***optional*** IPv6
  - Fusion Embedded File System
  - Hardware
    - BF516 running at 300MHz, 8MB RAM, 4MB Flash
    - SSM2603 high-fidelity stereo audio codec
    - 10/100Mbps Ethernet via RMII
    - SD Card Support, Digital GPIO
  - All software and hardware already integrated and optimized

1.4.4 Architecture

The architecture of **InstaVoIP** is shown in Figure 26, and explained further in the following subparagraphs:

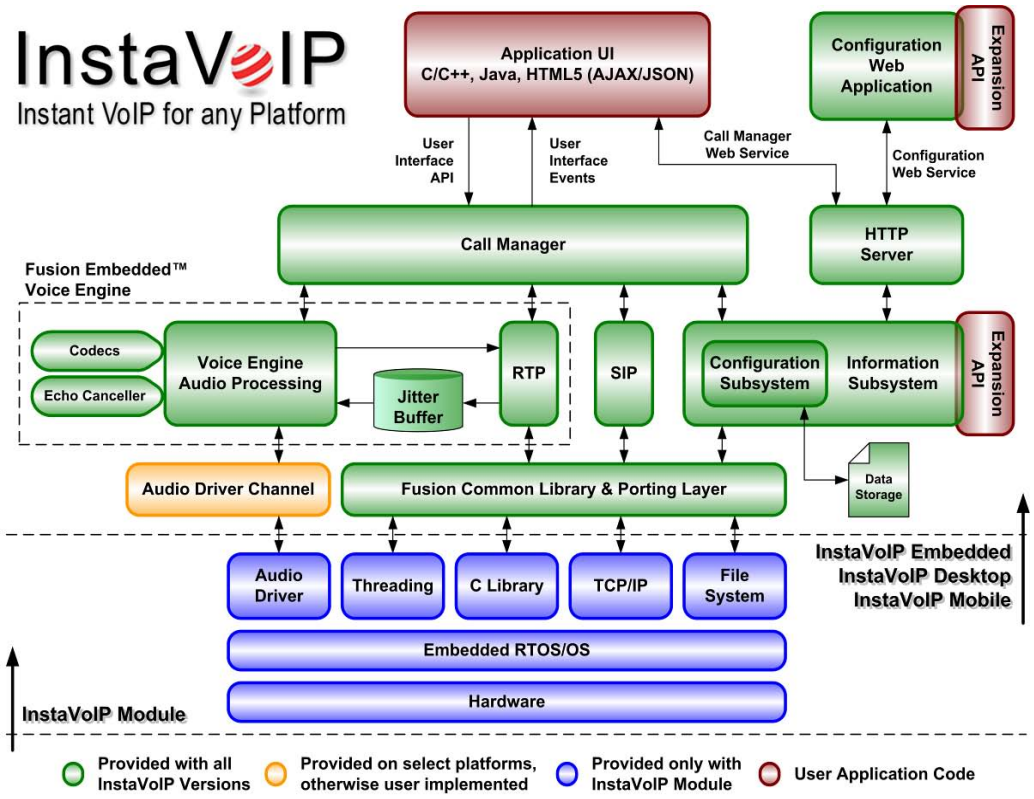


Figure 26 InstaVoIP

1.4.5 RTOS/OS, Drivers, Networking Stack, and File System

For **InstaVoIP** Embedded, Desktop, and Mobile this layer is assumed already in place and InstaVoIP integrates with this layer via the Fusion Common Library and the Audio Driver Channel.

For **InstaVoIP** Module, these capabilities are provided by Fusion RTOS, Fusion Net IPv4 (optional IPv6), Fusion File System, and an integrated and optimized set of drivers for flash storage and the audio codec.

1.4.6 Fusion Common Library (FCL) and Porting Layer

FCL provides a set of common utilities and a porting layer that is an abstraction over the standard C runtime library, threading and thread synchronization, file system access, and TCP/IP networking.

All Fusion Embedded products are written to the FCL porting layer interface, an implementation

of which must be available on each platform. Existing ports are available for Fusion Embedded, Win32/CE, Linux (or other POSIX), MacOS, iOS, Android, and uCOS-I/2/3; unsupported example ports are available for select other embedded RTOSs.

See the porting section below for a brief overview of the porting process.

1.4.7 Audio Driver Channel

FCL does not provide a standard abstraction for audio hardware access, so **InstaVoIP** defines its' own in the form of an "audio driver channel". This logical object receives analog data from the microphone and passes it to the Voice Engine; it also gets analog data from the Voice Engine and delivers it to the speakers. How this is done is platform dependent.

InstaVoIP is available with a few Audio Driver Channel implementations:

- A generic implementation using the open-source project PortAudio; this works immediately on platforms where PortAudio has been ported (Win32/CE, Linux, MacOS, etc.).
- An optimized implementation for Android
- An optimized implementation for MacOS and iOS.

For platforms where an existing port is not available, an Audio Driver Channel must be implemented. More details can be found in the *Fusion Embedded Voice Engine User's Guide*, but the general idea is that 10ms chunks of analog audio are put into and taken from voice engine queues and this content is mapped from/to appropriate buffers on the platform.

1.4.8 VoIP Networking Protocols

Full, expandable implementations of core VoIP networking protocols are included and pre-integrated with the Call Manager and Voice Engine. These protocols are:

- *SIP/SDP*: The Session Initiation Protocol (SIP) provides the functionality to register with SIP proxy servers, is used in managing individual calls (connect/disconnect, on/off hold, transfers, etc.), used for managing presence, and provides event notification (such as message-waiting indication). The Session Description Protocol (SDP) is used inside of certain SIP messages to provide details used during the different workflows.
- *RTP*: The Real-time Transport Protocol (RTP) is used to send real-time content (such as audio or out-of-band data like DTMF). The protocol provides metadata used to help receivers deal with network conditions such as jitter and lost packets.
- *IAX*: The Inter-Asterisk eXchange (IAX) protocol is an optional protocol used by

Asterisk servers which provides similar services to SIP/RTP. This is used for interoperability with Asterisk-based environments configured to use this protocol; the significant majority of new implementations of VoIP will use SIP/RTP.

- *STUN*: The Session Traversal Utilities for NAT (STUN) protocol helps SIP/RTP properly transition through Network Address Translation (NAT) modification done by most firewalls. This is important when "external" users need to connect to an "internal" system protected by a firewall.

- *SIPS/SRTP*: The Secure SIP (SIPS) and Secure RTP (SRTP) protocols are used when communications must be secure from eavesdropping; to be fully secure both protocols must be used in conjunction with one another. SIPS by itself is also useful for NAT/firewall traversal.

1.4.9 Voice Engine Audio Processing

The Voice Engine Audio Processing system, the core of the Fusion Embedded Voice Engine, implements the workflow which controls the encoding/decoding of audio packets, the application of various algorithms, and the mixing of multiple audio channels.

Analog channels (microphone, speaker, FXO/FXS, etc.) and digital channels (RTP) are implemented with different algorithmic combinations with fine control for each algorithm in the workflow. It has a plug-in architecture for adding additional codecs and echo cancellers (each of which may be available in optimized form on various platforms).

A jitter buffer is included to handle network jitter, packet bursts, and packet loss.

There is also an API to control Call Progress Tones (CPT) and play audio files (e.g. voice prompts for implementing an answering machine).

1.4.10 Call Manager

The Call Manager provides a simplified interface for dealing with standard call workflows such as a softphone, desktop IP phone, an FXS/FXO gateway, etc. The Call Manager hides the complexities and confusion of the SIP/SDP and RTP protocols which were not designed exclusively for VoIP applications. It also provides a consistent interface regardless of the underlying service protocol (i.e. SIP vs. IAX vs. FXO).

User Interfaces (UIs) are built using the call workflow API which is a combination of function calls and event notification. With this API you can also handle call transfers and call conferencing without the need to understand the nuances of the underlying protocol.

1.4.11 Info Subsystem

Centralized access to configuration data (which is persisted) and runtime data (which is not persisted) is provided by the Information Subsystem. This includes information such as server account configuration to active call information.

Configuration data is persisted and loaded at startup time; the default implementation uses JSON-based files 🦺 but can be replaced by a platform-appropriate solution.

An expansion API is provided that allows custom user information to be tracked and additional configuration information to be persisted and loaded. 🦺

1.4.12 Configuration Web Application and Web Service

A web-based interface for accessing information subsystem data and managing configuration data is provided both as an HTML-based web application and JSON-based web service.

The web application uses standard HTML/CSS for its design which allows for simple re-skinning, but it also uses the web service as its data channel. This keeps the presentation layer separate from the data layer which allows for complete restructuring of the look and feel without worrying about tightly coupled data access.

The web service also allows for access and configuration via an external application or device.

1.4.13 Call Manager Web Service

Similar to the Configuration Web Service, there is a JSON-based web service providing access to the Call Manager Interface API. This allows control of InstaVoIP from an external application; some customers have used centralized management applications to control banks of **InstaVoIP**-powered devices.

1.4.14 Expansion APIs

InstaVoIP has expansion APIs in key areas in the architecture which allow great flexibility in creating new or integrating into existing products.

1.4.15 Call Manager Interface

This is the main API that is used to integrate **InstaVoIP** into a product; it is used to instigate actions (place a call, disconnect a call, etc.) and handle asynchronous events (incoming call, peer disconnected, etc.).

As a short generic example of how the API works, let’s follow the workflow of an application initiating an outgoing call on a SIP-based service. One thing to realize is that the entire API is event driven because of the need to synchronize actions between various APIs. So every function

call returns immediately and at some point in the near future an event is received with the result of the function call.

The first thing an application would do is create a new outgoing-call structure. When this structure is fully initialized by SIP an "outgoing ready" event is received. At this point the application could choose to manipulate the call structure in some way (e.g. add custom headers such as those required by Assured-Services SIP (AS-SIP)) or just invoke the dial action. When SIP processes this call and the INVITE has been sent a "dialing" event is received.

At this point, events will come in as things happen down in the SIP layer. In normal scenarios, for example, a "ringback" event will come in and when the call is answered an "active" event is received.

1.4.16 Info Subsystem

The Info Subsystem is implemented as a hierarchal data model to which applications can add their own nodes; information can be added as runtime storage or as data that should be persisted. Using the info subsystem is beneficial because it allows use of helper functions that make passing JSON-formatted data to/from web pages very simple.

1.4.17 Configuration Web Application

The Configuration Web Application can be modified or expanded on many levels. Tweaks to CSS allow for quick branding without changing the general style or content, or HTML access allows the addition or removal of data fields or a complete replacement of the default style.

Additional "process functions" can be added to the underlying HTTP server to provide completely new functionality.

Data access/modification is done through the Info Subsystem which has helper functions to go to/from JSON format. Using a naming convention for input field IDs allows JavaScript helper functions to automatically populate fields and submit field values back to the server.

1.4.18 Voice Engine

If a platform has optimized versions of existing or new codecs, they can easily be integrated using the Voice Engine's Codec API. Similarly, the Voice Engine's Echo Canceller API allows optimized or third party echo cancellation implementations (this includes full-duplex or half-duplex acoustic echo cancellation or G.168-style line echo cancellation).

1.4.19 Porting

The porting effort for **InstaVoIP** can be broken down into the following areas: FCL, Audio Driver Channel, and Info Subsystem network configuration access.

1.4.20 Fusion Common Library

The FCL port consists of mapping macros to functions, and if necessary writing wrapper functions to make platform implementations conform to the FCL porting layer interface. This is done for C runtime library functions, RTOS/OS threading and thread synchronization, and networking.

The C runtime library porting is already done for most platforms. On occasion a few specific functions may not be available on a particular platform and in these cases the macro can be defined to use the FCL implementation (FCL comes with a basic C runtime library implementation).

RTOS/OS threading functions are already ported for several different platforms, but on new platforms wrapper functions must be written (and can be modeled after existing ports) to create new threads and manage semaphore and critical section synchronization objects.

The FCL networking interface follows the Berkeley standard, so any stack that follows this standard is already ported. For stacks that do not support this standard, wrapper functions must be written to convert the native API to the FCL interface.

1.4.21 Audio Driver Channel

The Audio Driver Channel port, at a high level, consists of dealing with two queues for each analog audio channel.

The speaker queue gets loaded by the Voice Engine ever 10ms with audio content and it is the channel's responsibility to remove this content (as quickly as the platform will allow) and send it to the platform's audio driver. The microphone queue must be loaded by the channel with 10ms of audio content as efficiently as possible.

Buffering of data must be done in some cases to match a platform's capability to the Voice Engine packets. For example, a speaker driver on a particular platform may take data in buffers of 70ms. In this case, the 10ms packets from the Voice Engine must be buffered into a 70ms buffer before delivery to the hardware driver.

Similarly if a microphone driver provides 70ms of data this must be split into seven 10ms packets and fed into the microphone queue.

An implementation of an Audio Driver Channel for the open-source project PortAudio is provided and can be used as a template for other platforms.

1.4.22 Info Subsystem

Certain aspects of the VoIP protocol require the network address of the device. In these cases, the VoIP code asks the info subsystem for the device's address and obtaining this information is

platform specific. A device must implement a function that will provide its external network address.

1.5 UCR Modifications to the *Unicoi* SIP stack

Like most of the "complete" SIP end instrument applications, the *Unicoi* SIP stack and call manager did not lend itself very well to adding support for additional RFCs. SIP headers in particular are well known hard coded field names within their stack. A release in late December of 2010 was totally incompatible with the AS-SIP phone application developed. Critical changes to the stack and voice engine required a brand new stock (i.e. non-AS-SIP) phone application for the Fanstel ST-3116. The release did include many fixes including the ability to do transfers. Consequently, two separate versions of AS-SIP phone prototype exist. Only the second prototype is being documented as the first is no longer valid.

The ability to add headers and values to the *Unicoi* SIP stack, or any SIP stack, was a critical step in terms of adding the UCR assured services requirements. The ability to receive and process unexpected headers was also critical. The *Unicoi* SIP stack handled unexpected SIP headers appropriately by simply ignoring them.

For outbound SIP messages, specifically messages initiated by our phone, a set of macros were used that allowed us to add custom header information to any SIP method without modifying the stack. Each macro (below) is associated with a particular SIP method and is present throughout the *Unicoi* call manager source code. The values assigned to each macro, in this case **L3_SetSipHeader** are the name(s) of functions defined that supply the header name and value pair added to the outgoing SIP message. The function has to be defined as follows:

void L3_SetSipHeader (fcmSipCall_t *pCall, void **pCustomHeaders);

The macros for custom header support are defined as follows in **fcm_sip.h**:

```
//#define FCM_SIP_REGISTER_CB

#define FCM_SIP_CALL_MAKE_CB L3_SetSipHeader

#define FCM_SIP_CALL_REPLACE_CB L3_SetSipHeader

#define FCM_SIP_CALL_MODIFY_CB L3_SetSipHeader

#define FCM_SIP_CALL_END_CB L3_SetSipHeader

#define FCM_SIP_CALL_END_AUTH_CB L3_SetSipHeader

#define FCM_SIP_NOTIFY_CB L3_SetSipHeader

#define FCM_SIP_SUBSCRIBE_CB L3_SetSipHeader
```

```
#define FCM_SIP_REFER_CB L3_SetSipHeader

#define FCM_SIP_OPTIONS_CB L3_SetSipHeader

#define FCM_SIP_CALL_REDIRECT_CB L3_SetSipHeader

#define FCM_SIP_CALL_RINGING_CB L3_SetSipHeader

#define FCM_SIP_CALL_ACCEPT_CB L3_SetSipHeader

#define FCM_SIP_CALL_REJECT_CB L3_SetSipHeader

#define FCM_SIP_CALL_MODIFY_RESP_CB L3_SetSipHeader

#define FCM_SIP_MESSAGE_CB L3_SetSipHeader
```

Adding custom header support in this manner meant not having to customize the stock *Unicoi* call manager to support custom headers. The code works the same way whether they are defined or not as illustrated below. In this code fragment, from source file: **fcm_sip_call_make.c**, use of the **FCM_SIP_CALL_MAKE_CB** and **FCM_SIP_CALL_REPLACE_CB** macros add custom headers to all **INVITE SIP** messages. Note that the function is only a fragment of the original and has been edited for brevity:

```
fcmCallId fcmSipMakeCall( fcmAccountId lineID, const char * pszRemoteURIc, void *
cookie )

{

    fcmCallId      callID;

    eRErr errSIP;

    char    *pszRemoteURI = (char*)pszRemoteURIc;

    int      err;

    void    *pCustomHeaders = NULL;


    fcmSipLine_t          *pLine;

    fcmSipCall_t          *pCall;


  if (cookie != NULL){
```

```
    fcmSipReplaces_t* replaces = (fcmSipReplaces_t*)cookie;
    FCM_SIP_CALL_REPLACE_CB(pCall, &pCustomHeaders);
    errSIP = fnsSipCallReplaceEx ( pLine->lineID, pCall->pszRemoteURI,
                        pCall->pszLocalSDP, &(pCall->pszCallID),
                        0, 0, 0, replaces->ptsReplaces,
                        replaces->pszReferredBy, pCustomHeaders);
    FCL_FREE(replaces->pszReferredBy);
    FCL_FREE(replaces->ptsReplaces->pszToTag);
    FCL_FREE(replaces->ptsReplaces->pszFromTag);
    FCL_FREE(replaces->ptsReplaces->pszReplaceCallId);
    FCL_FREE(replaces->ptsReplaces);
    FCL_FREE(replaces);
  }else{
    /* Make the call via SIP API */
    pCall->pszCallID = NULL;

    FCM_SIP_CALL_MAKE_CB(pCall, &pCustomHeaders);
    errSIP = fnsSipCallInvite ( pLine->lineID, pCall->pszRemoteURI,
                        pCall->pszLocalSDP, &(pCall->pszCallID),
                        NULL, NULL, NULL, pCustomHeaders );
  }
    FCM_SIP_TRACE("FCM-SIP: fnsSipCallInvite Success" ENDL);
  }
  callID = pCall->callID;
  fcmSipDetachFromCall( pCall );
```

```
    return callID;
}
```

Nearly all of the **active** or outbound SIP call manager functions were modified to include the appropriate macro above. From the application, a user-defined macro becomes a defined function for passing the custom header and value.

Capturing inbound SIP header information required modifications to the original SIP stack code. Our own scheme for receiving additional SIP header info was implemented.

1.5.1 GUI Modifications

The actual Fanstel phone application was not written or maintained by *Unicoi* nor did it come with any documentation. *Unicoi* did however warn us that the code was not well structured. No attempt was made by the authors to abstract the very low level machine like instructions for controlling and manipulating the display or IO buttons which consequently lead to thousands of lines of repetitive and unnecessary code. No attempt was made to come up with a finite state machine for controlling and operating the phone which also lead to thousands of lines of redundant and unnecessary code. The numerous states of the phone were instead controlled with hundreds of standalone global variables.

1.5.2 Host Based Builds

As previously stated, the *Unicoi* stack can be built for a non-embedded Windows platform. *Unicoi* provides three different Windows based prototype SIP applications. Each of them use different amounts of the Fusion codebase. We refer to these prototypes as the Cougar Win32 Net Client, the Headless Client and the Fusion SipPhone. Each of these prototypes has different features and functionality. The Cougar Win32 Net Client is the latest iteration and the Fusion SipPhone is the oldest iteration.

The Cougar Win32 Net Client shares the most code with the embedded prototype. Its GUI is built using Windows Forms and Microsoft .Net 2.0 Framework. It also uses WinSock2 for network connections. It is provisioned using the same HTTP interface as the embedded prototype and like the embedded prototype also offers a telnet connection for debugging and logging. Supported features include the following:

- Make / Receive VoIP Calls
- Blind Transfer, Consultant Transfer, Call Hold and Retrieve
- Multiple Call Appearances
- Secure SIP using TLS with both simple and mutual authentication
- Secure RTP and RTCP
- SIP Registration / Authentication

- HTTP Provisioning Interface
- Telnet Shell
- Fusion Call Manager
- Fusion Voice Engine
- WinSock2
- Static or Dynamic IPv4 Dressing

The Windows MS Dev Studio solution for building the Win32 Net Client is located in $UNICOI_INSTALL_DIR\fusion\refdesigns\voip\ip_phone\ports\cougar\win32. The project solution was built with Dev STudio 2005 and was converted to our 2008 version of Dev STudio. In order to successfully build the project a few environment variables need to be set. **FUSION_DIR** needs to be set to the root directory of the fusion source, this is the directory that contains the sub-directories of fusion packages such as {algorithms, common, config, customer, docs, drivers, file, net, **…}** **FUSION_LIBRARY_DIR** needs to be set to the directory that contains the portaudio library.

This page intentionally left blank.

CHAPTER 2 Security

2 Security

Unicoi supported both SRTP and most of the SSL/TLS based protocols required by the Information Assurance Requirements of the UCR. There was no support for IPsec in the *Unicoi* stack or in the REDCOM switch. We were able to implemented SRTP and both the Simple TLS handshake and the Client-authenticated TLS handshake (Mutual Authentication) from our *Unicoi* based end instrument application. The *Unicoi* software requires specific builds and recompilation to include features like secure RTP/ SSL/TLS, IPV4 and or IPV6.

The Simple TLS handshake is where the server side of the connection is authenticated by transmitting its certificate to the client whom then makes a decision whether or not to accept the certificate thereby establishing a trusted connection with the server. Following are the steps necessary to set the LSC and the EI up to use the Simple TLS handshake. The following 3 steps were taken to setup our EI for TLS hand shaking:

2.1 Generating Certificates Using Open SSL

1. OpenSSL can be used to generate certificates. There are a number of helper scripts written in Perl that help with the creation and maintenance of the certificate and keys. OpenSSL is available from a number of sites, i.e., slproweb.com. The default installation is adequate although it may be useful to add the OpenSSL\bin directory to the system environment variable PATH. Perl is also available from a number of sites, we used activestate.com. The default installation is adequate. OpenSSL reads a configuration file (openssl.cfg) which helps define the certificate configuration that will be generated. The following is an example open SSL configuration file:

#

 # OpenSSL example configuration file.

 # This is mostly being used for generation of certificate requests.

#

 # This definition stops the following lines choking if HOME isn't

 # defined.

HOME = .

RANDFILE = $ENV::HOME/.rnd

```
# Extra OBJECT IDENTIFIER info:
#oid_file        = $ENV::HOME/.oid
oid_section      = new_oids

# To use this configuration file with the "-extfile" option of the
# "openssl x509" utility, name here the section containing the
# X.509v3 extensions to use:
# extensions       =
# (Alternatively, use a configuration file that has only
# X.509v3 extensions in its main [= default] section.)
[ new_oids ]
# We can add new OIDs in here for use by 'ca' and 'req'.
# Add a simple OID like this:
# testoid1=1.2.3.4
# Or use config file substitution like this:
# testoid2=${testoid1}.5.6

####################################################################
[ ca ]
default_ca      = CA_default          # The default ca section

####################################################################
[ CA_default ]

dir          = ./demoCA            # Where everything is kept
certs        = $dir/certs          # Where the issued certs are kept
crl_dir      = $dir/crl            # Where the issued crl are kept
database     = $dir/index.txt      # database index file.
#unique_subject = no                # Set to 'no' to allow creation of
```

```
                        # several ctificates with same subject.
new_certs_dir   = $dir/newcerts         # default place for new certs.

certificate     = $dir/cacert.pem       # The CA certificate
serial          = $dir/serial           # The current serial number
crlnumber       = $dir/crlnumber        # the current crl number
                        # must be commented out to leave a V1 CRL
crl             = $dir/crl.pem          # The current CRL
private_key     = $dir/private/cakey.pem# The private key
RANDFILE        = $dir/private/.rand    # private random number file

x509_extensions = usr_cert              # The extentions to add to the cert

# Comment out the following two lines for the "traditional"
# (and highly broken) format.
name_opt        = ca_default            # Subject Name options
cert_opt        = ca_default            # Certificate field options

# Extension copying option: use with caution.
# copy_extensions = copy

# Extensions to add to a CRL. Note: Netscape communicator chokes on V2 CRLs
# so this is commented out by default to leave a V1 CRL.
# crlnumber must also be commented out to leave a V1 CRL.
# crl_extensions        = crl_ext

default_days    = 365                   # how long to certify for
default_crl_days= 30                    # how long before next CRL
default_md      = sha1                  # which md to use.
```

```
preserve       = no                # keep passed DN ordering

# A few difference way of specifying how similar the request should look
# For type CA, the listed attributes must be the same, and the optional
# and supplied fields are just that :-)
policy         = policy_match

# For the CA policy
[ policy_match ]
countryName            = match
stateOrProvinceName    = match
organizationName       = match
organizationalUnitName  = optional
commonName             = supplied
emailAddress           = optional

# For the 'anything' policy
# At this point in time, you must list all acceptable 'object'
# types.
[ policy_anything ]
countryName            = optional
stateOrProvinceName    = optional
localityName           = optional
organizationName       = optional
organizationalUnitName  = optional
commonName             = supplied
emailAddress           = optional

#####################################################################
```

[req]
default_bits = 1024
default_keyfile = privkey.pem
distinguished_name = req_distinguished_name
attributes = req_attributes
x509_extensions = v3_ca # The extentions to add to the self signed cert

 # Passwords for private keys if not present they will be prompted for
 # input_password = secret
 # output_password = secret

 # This sets a mask for permitted string types. There are several options.
 # default: PrintableString, T61String, BMPString.
 # pkix : PrintableString, BMPString.
 # utf8only: only UTF8Strings.
 # nombstr : PrintableString, T61String (no BMPStrings or UTF8Strings).
 # MASK:XXXX a literal mask value.
 # WARNING: current versions of Netscape crash on BMPStrings or UTF8Strings
 # so use this option with caution!
string_mask = nombstr

 # req_extensions = v3_req # The extensions to add to a certificate request

[req_distinguished_name]
countryName = Country Name (2 letter code)
countryName_default = US
countryName_min = 2
countryName_max = 2

stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = Massachusetts

localityName = Locality Name (eg, city)
localityName_default = Newburyport

0.organizationName = Organization Name (eg, company)
0.organizationName_default = L-3 Maritime Systems

 # we can do this but it is not needed normally :-)
#1.organizationName = Second Organization Name (eg, company)
#1.organizationName_default = World Wide Web Pty Ltd

organizationalUnitName = Organizational Unit Name (eg, section)
#organizationalUnitName_default =

commonName = Common Name (eg, hostname or IP address)
commonName_max = 64

emailAddress = Email Address
emailAddress_default = account@domain.com
emailAddress_max = 64

 # SET-ex3 = SET extension number 3

[req_attributes]
challengePassword = A challenge password
challengePassword_min = 4
challengePassword_max = 20

unstructuredName = An optional company name

[usr_cert]

 # These extensions are added when 'ca' signs a request.

 # This goes against PKIX guidelines but some CAs do it and some software
 # requires this to avoid interpreting an end user certificate as a CA.

basicConstraints=CA:FALSE

 # Here are some examples of the usage of nsCertType. If it is omitted
 # the certificate can be used for anything *except* object signing.

 # This is OK for an SSL server.
 # nsCertType = server

 # For an object signing certificate this would be used.
 # nsCertType = objsign

 # For normal client use this is typical
 # nsCertType = client, email

 # and for everything including object signing:
 # nsCertType = client, email, objsign

 # This is typical in keyUsage for a client certificate.
keyUsage = nonRepudiation, digitalSignature, keyEncipherment

```
# This will be displayed in Netscape's comment listbox.
nsComment                       = "OpenSSL Generated Certificate"

# PKIX recommendations harmless if included in all certificates.
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer

# This stuff is for subjectAltName and issuerAltname.
# Import the email address.
# subjectAltName=email:copy
# An alternative to produce certificates that aren't
# deprecated according to PKIX.
# subjectAltName=email:move

# Copy subject details
# issuerAltName=issuer:copy

#nsCaRevocationUrl              = http://www.domain.dom/ca-crl.pem
#nsBaseUrl
#nsRevocationUrl
#nsRenewalUrl
#nsCaPolicyUrl
#nsSslServerName
[ v3_req ]
# Extensions to add to a certificate request
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
```

```
[ v3_ca ]
# Extensions for a typical CA
# PKIX recommendation.
subjectKeyIdentifier=hash

authorityKeyIdentifier=keyid:always,issuer:always

# This is what PKIX recommends but some broken software chokes on critical
# extensions.
#basicConstraints = critical,CA:true
# So we do this instead.
basicConstraints = CA:true

# Key usage: this is typical for a CA certificate. However since it will
# prevent it being used as an test self-signed certificate it is best
# left out by default.
keyUsage = cRLSign, keyCertSign

# Some might want this also
# nsCertType = sslCA, emailCA

# Include email address in subject alt name: another PKIX recommendation
# subjectAltName=email:copy
# Copy issuer details
# issuerAltName=issuer:copy

# DER hex encoding of an extension: beware experts only!
# obj=DER:02:03
# Where 'obj' is a standard or added object
```

 # You can even override a supported extension:
 # basicConstraints= critical, DER:30:03:01:01:FF

[crl_ext]

 # CRL extensions.
 # Only issuerAltName and authorityKeyIdentifier make any sense in a CRL.

 # issuerAltName=issuer:copy
authorityKeyIdentifier=keyid:always,issuer:always

[proxy_cert_ext]
 # These extensions should be added when creating a proxy certificate

 # This goes against PKIX guidelines but some CAs do it and some software
 # requires this to avoid interpreting an end user certificate as a CA.

basicConstraints=CA:FALSE

 # Here are some examples of the usage of nsCertType. If it is omitted
 # the certificate can be used for anything *except* object signing.

 # This is OK for an SSL server.
 # nsCertType = server

 # For an object signing certificate this would be used.
 # nsCertType = objsign

 # For normal client use this is typical

```
# nsCertType = client, email

# and for everything including object signing:
# nsCertType = client, email, objsign

# This is typical in keyUsage for a client certificate.
# keyUsage = nonRepudiation, digitalSignature, keyEncipherment

# This will be displayed in Netscape's comment listbox.
nsComment                       = "OpenSSL Generated Certificate"

# PKIX recommendations harmless if included in all certificates.
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer:always

# This stuff is for subjectAltName and issuerAltname.
# Import the email address.
# subjectAltName=email:copy
# An alternative to produce certificates that aren't
# deprecated according to PKIX.
# subjectAltName=email:move

# Copy subject details
# issuerAltName=issuer:copy

#nsCaRevocationUrl              = http://www.domain.dom/ca-crl.pem
#nsBaseUrl
#nsRevocationUrl
#nsRenewalUrl
```

#nsCaPolicyUrl

#nsSslServerName

 # This really needs to be in place for it to be a proxy certificate.

proxyCertInfo=critical,language:id-ppl-anyLanguage,pathlen:3,policy:foo

2. To generate the Certificate Authority (CA) Certificate open a Windows command window and at the DOS prompt, issue the following command:

C:\Documents and Settings\My Documents>perl c:\OpenSSL\bin\CA.pl -newca

3. Follow the prompts. When finished a new **demoCA** directory will be created in the *\Documents and Settings\My Documents* directory. The CA certificate will be named **cacert.pem** in the **demoCA** directory, and contains both a text description of the certificate, and the certificate itself in PEM (privacy enhanced email) format. Make a copy of it, edit the copy, and delete everything before the line that contains **-----BEGIN CERTIFICATE-----.** This is the **ca.pem** that will be installed on the switch. It may be stored in a public place, so it can be copied and installed easily. The next step is to create a certificate.

4. To generate a certificate request and private key, again from a Windows command window, issue the following command:

C:\Documents and Settings\My Documents> perl c:\OpenSSL\bin\CA.pl -newreq-nodes

5. Follow the prompts. When finished the request will be in **newreq.pem** and the private key will be in **newkey.pem**. Next a certificate is generated for the switch using the request generated from the **ca.pem** and **newkey.pem**.

6. To generate and sign the certificate issue the following command from the same Windows command line prompt:

C:\Documents and Settings\My Documents>Perl c:\OpenSSL\bin\CA.pl -sign

7. Follow the prompts. The new certification is **newcert.pem** in *C:\Documents and Settings\My Documents* and will be the **cert.pem** file for the REDCOM switch. Like the CA cert, this file contains both a text description and the PEM encoded certificate, so the copy must be edited, and everything before "-----BEGIN CERTIFICATE-----" is removed. Next, the key must be converted to a private RSA. "RSA" is an algorithm for public-key encryption and gets its name from the 3 authors who defined it: "Rivest", "Shamir" and "Adleman".

8. To convert the private key to RSA, again issue the following command from the Windows command prompt:

C:\Documents and Settings\My Documents> openssl rsa -in newkey.pem -outform PEM -out key.pem

2.2 Installing Certificates on the LSC

The LSC in this case is the REDCOM SLICE 2100 switch used all along to test the AS-SIP end instrument. The REDCOM switch also has to be configured to open a well known port for TLS based connections and TLS related parameters.

1. The three files generated in paragraph 2.1: **ca.pem**, **cert.pem** and **key.pem** need to be uploaded to the REDCOM SLICE using tftp. Use tftp to copy over the certificates and keys. The recommended place to put them is in the */sys/CERTIFCATES* folder. Create a sub-folder *certs* for the switch credentials (**cert.pem, key.pem**) and another sub-folder **ca** for the CA credentials **{ca.pem}.** Note that after changing the file system it is necessary to duplicate it in ROM by issuing the following REDCOM SLICE command:

rsh> dup /sys /sysrom
(Confirm the action by answering with a **Y**)

The Sys/CERTIFICATES directory should look like this:

rsh /sys/CERTIFICATES> dir ca
Directory listing for /sys/CERTIFICATES/ca
17-Nov-2010 16:03:04 1379 REG ca.pem
2.03M bytes free of 2.09M
114 files free of 124

rsh /sys/CERTIFICATES> dir certs
Directory listing for /sys/CERTIFICATES/certs
17-Nov-2010 16:02:43 887 REG key.pem
17-Nov-2010 16:02:52 1208 REG cert.pem
2.03M bytes free of 2.09M
114 files free of 124

2. Next configure TLS on the REDCOM SLICE using the **adm** tool. REDCOM has a telnet port and custom command shell for connecting to and administrating the switch. The user must be the **highroot** user and supply a password to reach the REDCOM command window. Tools like **telnet**, **putty** are all adequate.

3. After connecting to the REDCOM switch, issue the **adm** and **tls** commands to get to the TLS configuration interface as shown in Figure 27:

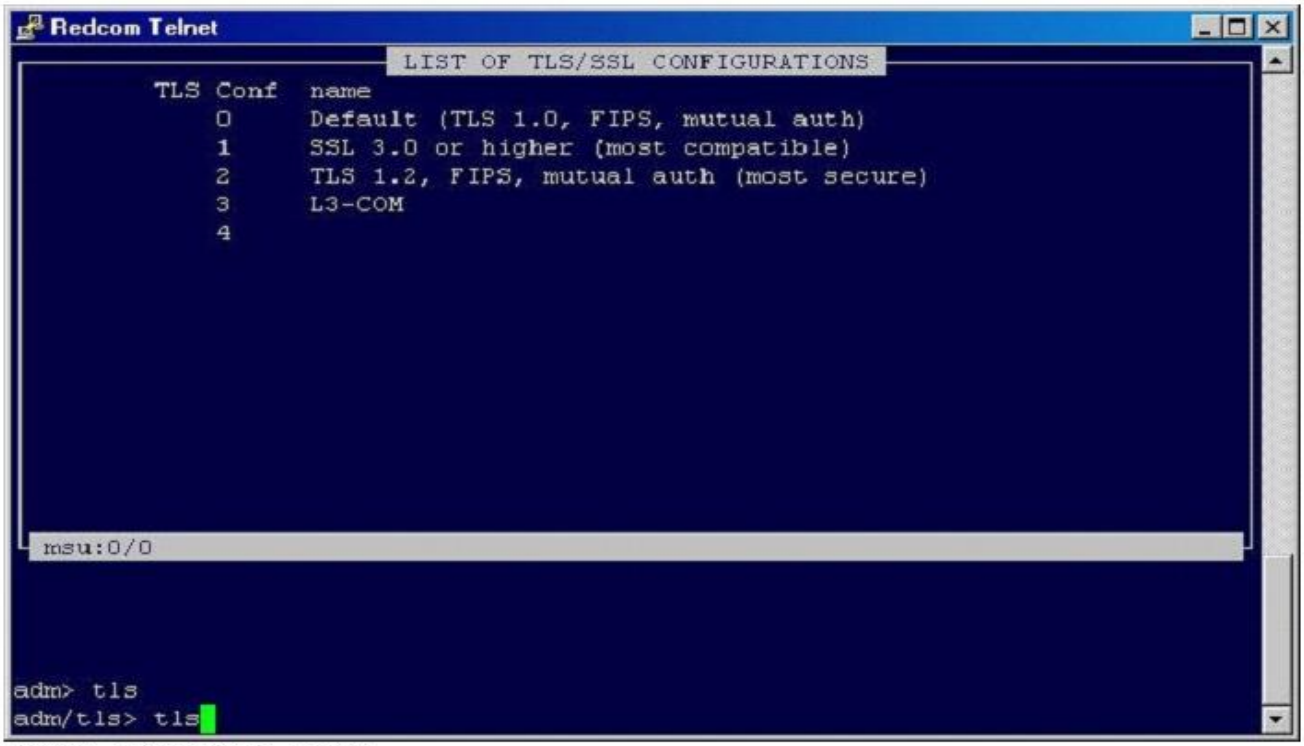


Figure 1 - Redcom Slice - adm/tls

Figure 27 Admin_Config_Screen

4. Next, set up a TLS entry as shown in Figure 28:

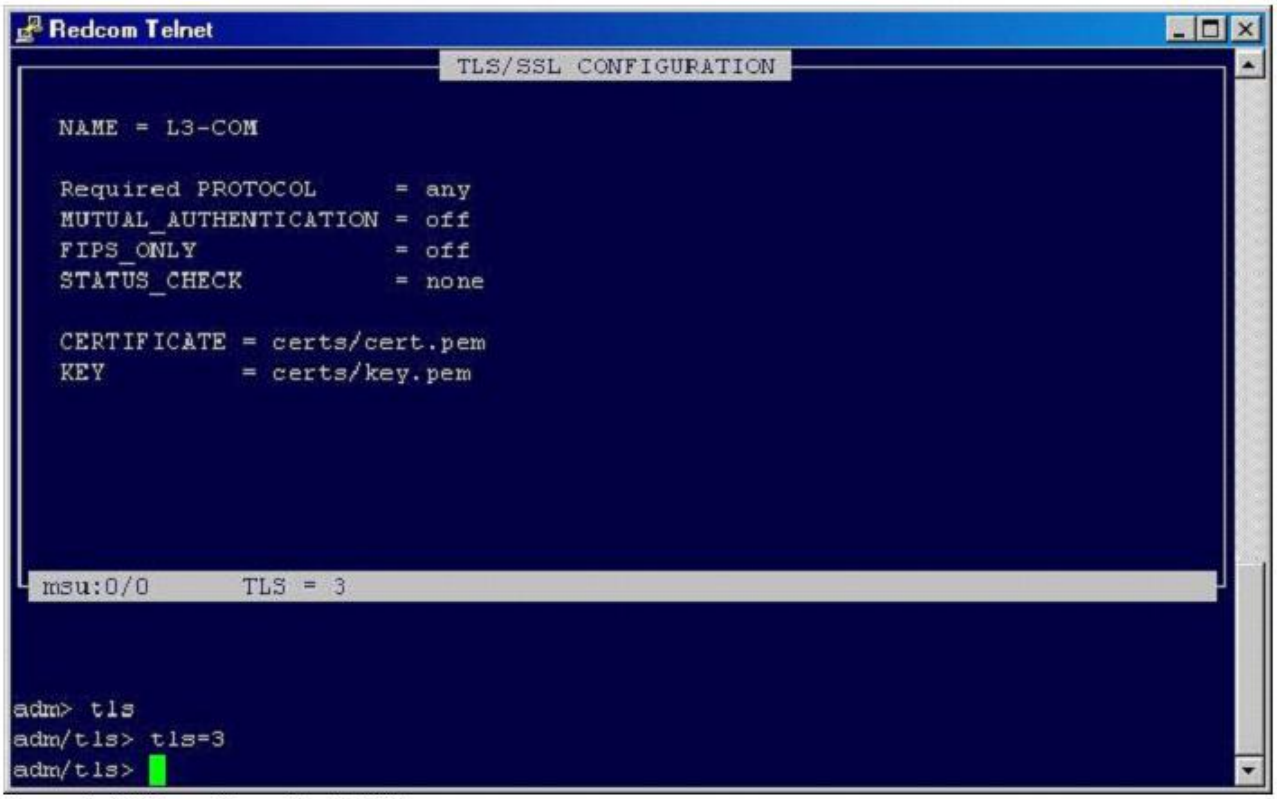


Figure 1 - Redcom Slice - Simple TLS

Figure 28 rcom_tls_entry Screen

5. Now issue the REDCOM shell command: **act; ex** which will activate the changes before leaving this tool.

6. Next, setup and activate the devices associated with secure SIP. Again, after logging into the REDCOM switch, issue the **adm** command followed by **devices** as shown in Figure 29 to get to the device management screen:

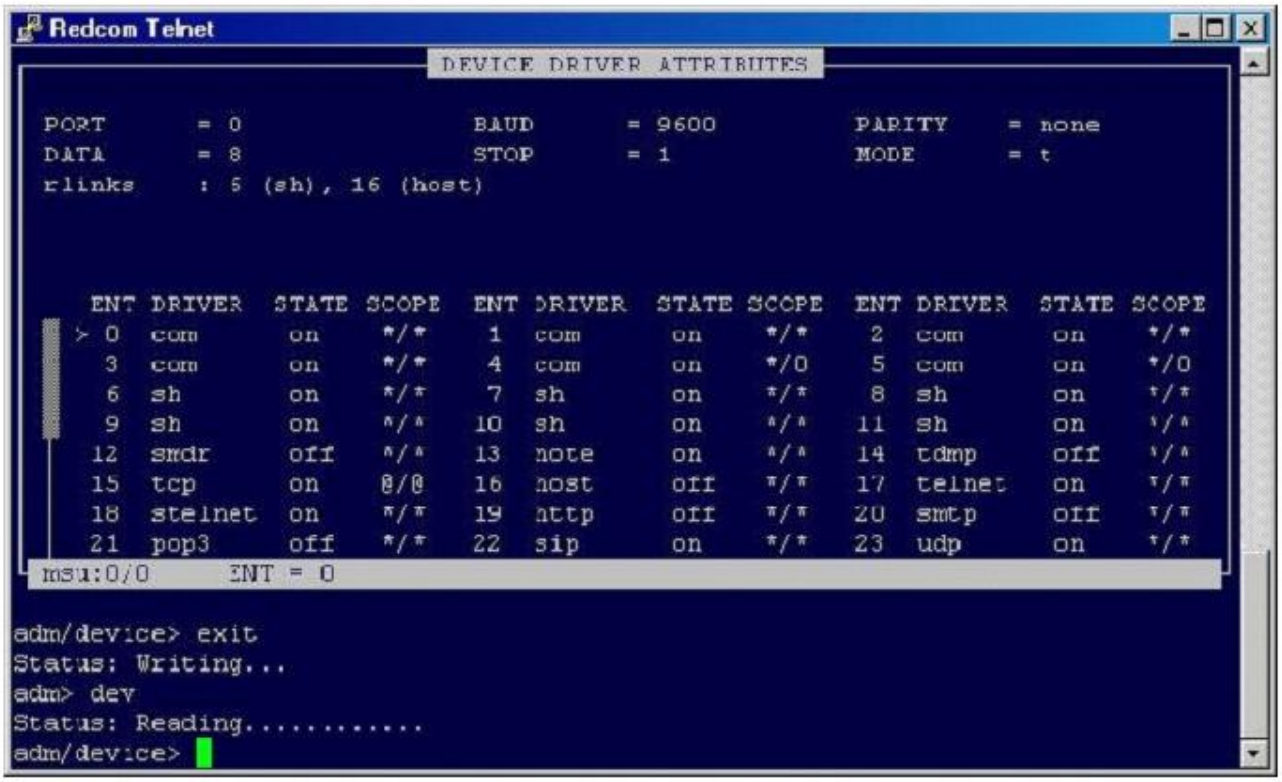


Figure 1 - Redcom Slice - Devices

Figure 29 rcom_device_mgmnt Screen

7. To set up secure TCP, specify the entry **ent** and desired parameter (e.g. **state**). As shown in Figure 1-29, enable secure TCP by issuing **ent=26** followed by **state=on**, set the port **port=5061** and TLS # from the very first devices screen **tls=3** (Figure 30):

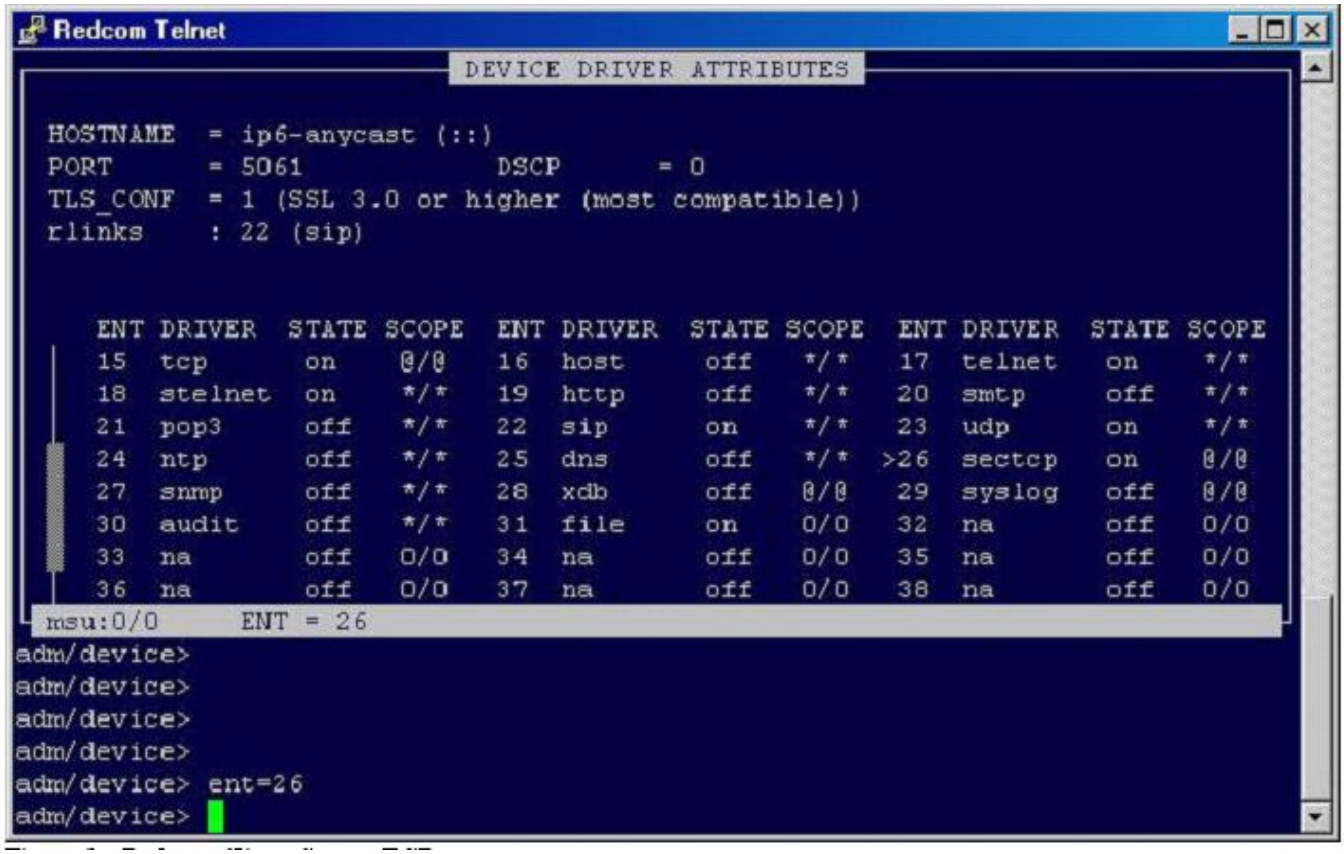


Figure 30 rcom_secure_tcp Screen

8. Using the same devices screen above, link SIP with secure TCP with the command

ent=22 then **link=26**

9. Again, using the same devices screen above, turn on secure telnet with the commands: **ent=18** followed by **state=on**. Set TLS to 3 again: **tls=3**. Exit the devices tool by issuing the **act; ex** command to activate and save the changes.

10. REDCOM must be reset at this point, issue the **reset** command or power on/off the switch.

11. To confirm all of the REDCOM settings, log back into the switch, issue the **adm** command, and from the adm prompt issue **netstat -t** as shown in Figure 31:

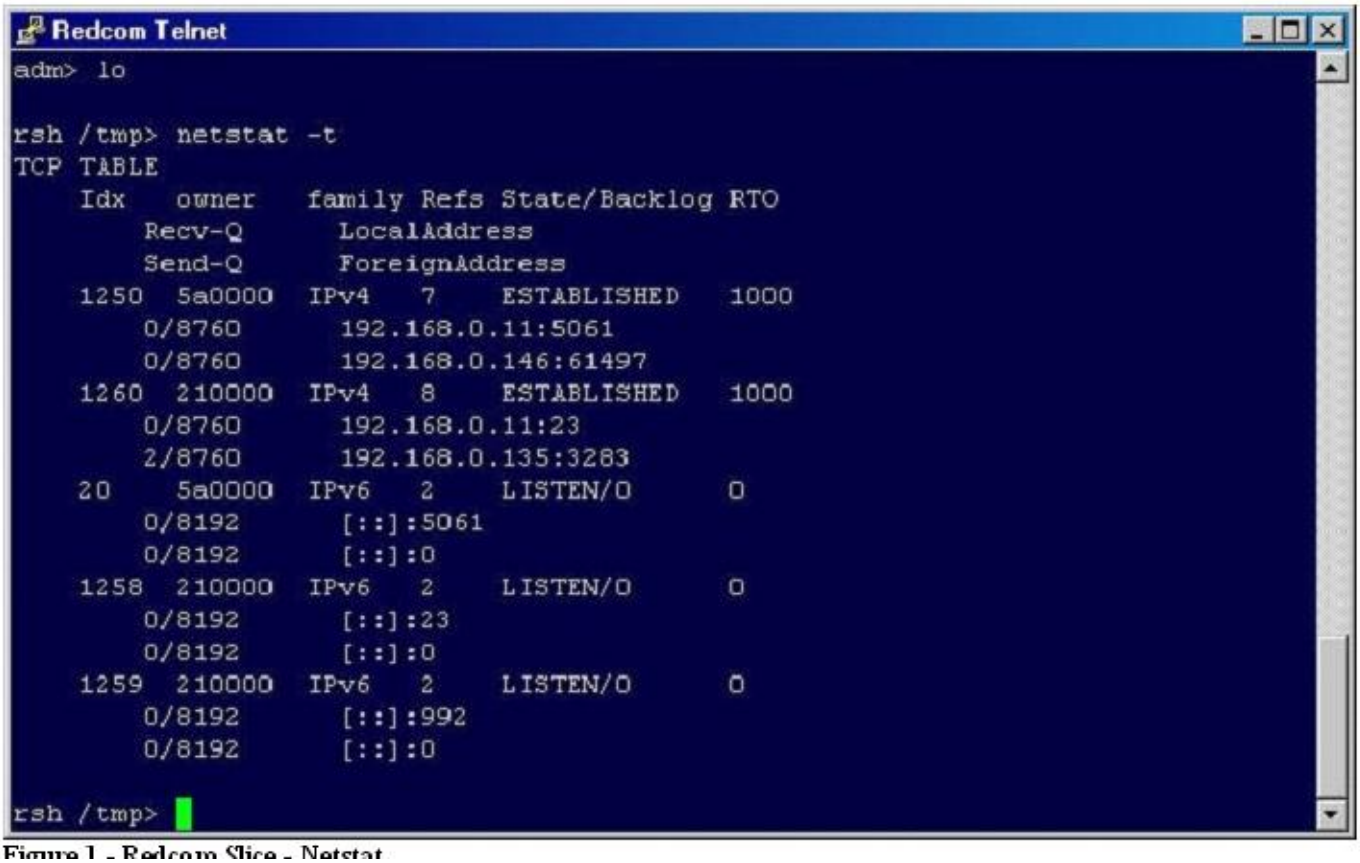


Figure 31 rcom_netstat Screen

2.3 Installing the Certificates on our *Unicoi* EI

Unicoi provides several different VDSP++ project groups for the purpose of including or excluding certain features like secure mode and or SRTP. The available project groups for the embedded application are located under:

$UNICOI_INSTALL_DIR\fusion\refdesigns\voip\ip_phone\ports\cougar\bf537 and they are as follows:

cougar.dpg, this will build a version of the application with no SRTP or TLS

cougar_sips.dpg, this will build a version with support for TLS only.

cougar_sips_srtp.dpg, this will build a version with support for both TLS and SRTP.

Our AS-SIP application is built using the **cougar_sips_srtp.dpg** project. It should be noted that simply building the project does not include the features at run time, there are also both configuration and code changes necessary to enable both SRTP and secure sockets.

1. Modify source file:

$UNICOI_INSTALL_DIR\fusion\security\ussl\ussl\sessionoptions.h

2. Change:

#define USSL_STANDARD_SERVER_AUTHENTICATION ussl_authenticate_full

to:

#define USSL_STANDARD_SERVER_AUTHENTICATION ussl_authenticate_none

This page intentionally left blank.

Chapter 3 Conclusion

3 Introduction

The *Unicoi BF-537* embedded solution utilized in the Embedded Proof-of-Concept end instrument application was one of many readily available, commercial-off-the-shelf, DSP based reference designs that have been widely used for a number of years in the commercial sector for VoIP based communications. *Analog Devices* and *Texas Instruments* dominate the embedded DSP market today and have hundreds of value-added resellers like *Unicoi* that offer complete VoIP solutions for IP based communications. Reference designs like the *Blackfin-537* DSP board from *Unicoi* (that include source code) make it possible to construct a UCR compliant SIP end instrument.

3.1 Conclusion

The fact that our final Proof-of-Concept end instrument was embedded was of no consequence in terms of implementing a UCR-compliant SIP end instrument. Every commercial, open source or embedded SIP stack investigated had limited RFC support, loosely and or non-compliant RFC support, and a host of software-related bugs particularly when it was attempted to mimic traditional telephony functions as required by the UCR. Most were also written to support a specific soft switch or switches and the embedded *Unicoi* SIP stack client was no exception. It should be noted that for the purposes of this paper the term "SIP stack" is used to include all of the related networking stack software that makes SIP possible. This includes, from the bottom up, the "bottom" being the physical layer of the Open Systems Interconnection model (OSI): Ethernet, SSL/TLS, IP, TCP, UDP, RTP, SRTP and SIP. No attempt was made to build our end instrument from the "ground up"; we started with functional SIP stacks and modified the stock source to be UCR compliant. Our focus was predominately on the upper-most layer of the OSI model, in this case the Session Initiation Protocol layer. Building a stack from the ground up would more or less be like reinventing the wheel as there are hundreds of proven, readily available commercial and free networking stacks all the way up the OSI model.

Like most of the SIP stacks evaluated, the vast majority of our efforts were spent debugging the stock *Unicoi* SIP stack. The *Wireshark* tool employed to troubleshoot and verify the end instrument was indispensable. *Wireshark* is a free, open source, network packet analyzer with built-in filters for nearly every IP based protocol including SIP. Our development cycle consisted of repetitive captures of specific SIP scenarios, careful analysis of the **pcap** data, verification of the behavior against the effected RFC's, identification of the source of each issue, and presentation of the findings to the effected vendor (i.e. Unicoi or REDCOM) and coordinating and installing a resolution of the problem. REDCOM's server side SIP stack was a fairly mature product and obviously designed to be UCR-compliant so problems on their end were minimal. It should be

noted that we did not own the *Unicoi* stack and while we frequently diagnosed and provided solutions for many of the issues we found we were dependent on *Unicoi* to make and release most fixes and enhancements for us to keep our stack in sync and upgradeable.

It is important to make a distinction here about the type of embedded environment we employed. Traditional, large host-based operating systems like Windows and Linux now offer much smaller versions for the embedded environment but are not necessarily "hard" real time systems. Real time systems have nothing to do with the speed or services offered but rather the deterministic and predictable nature of each CPU cycle. For example, unless you choose to implement one, real-time operating systems like *Fusion* RTOS or *Texas Instruments* BIOSII do not provide a fair scheduling service for each process to get a slice of the CPU. It's entirely up to the application developer to make these decisions based on the needs of the application. *Fusion* RTOS (and others like it) simply provide the developer with the raw components of an operating system such as multithreading, preemptive scheduling, thread synchronization (mutexing), task prioritization and clocking. An RTOS like *Fusion* can be thought of as the tools you would use to build an operating system like Windows or Linux. There is no distinction between the application and the RTOS in an embedded environment, all are compiled and linked into a single binary image and become, in every sense, the operating system environment. In Windows or Linux the application runs separate from and within the confines of the operating system. Malfunctioning programs simply terminate in this environment leaving the underling OS and hardware still functioning. No such safety net exists in the embedded environment unless it is implemented into the application. The *Blackfin 537* DSP board does not have a hard disk storage device nor do most DSP based boards. Hard drives are the most common source of failures in any computer. The only persistent memory available is a small 4 MEG flash drive that is extremely difficult to tamper with without disabling the entire unit. Successful tampering of the flash drive would not gain much anyway, as there are no standard operating system services or programs to exploit.

The *Unicoi* stack is now the most UCR-compliant and ready COTS SIP stack available in our awareness. Packet level encryption, better known as IPSec, is the only UCR requirement not implemented. This is a protocol that lives in the bottom most "physical" layer of the OSI model and would need to be integrated into the IPV4/ IPV6 stacks. We did not find a single stack that supported IPSec for the client or server including *Unicoi* or the REDCOM switch that was tested. The lack of support for IPSec is a market driven phenomenon since few outside the military have a need for this level of security, in addition to the more common SSL/TLS and SRTP security measures. *Unicoi* and other stack vendors are more than willing to add support for IPSec; however the cost is considerable relative to what was already purchased. The inherent security, reliability, cost effectiveness and lack of maintenance required for embedded solutions would seem to be a perfect fit for ship board use.

This project has impacted the future direction of the US Navy vessels in the integration of voice, video and data. This was demonstrated in the symposiums that our investigators have attended

and in its presentations as well as conversations and requests for them to present to individual companies and military organizations. It has shown the feasibility of implementing integration of voice, video and data, in the areas of end device design, AS-SIP variations, IPv6 requirements as well as the need for their integration into a stable network platform.

The merging of the three phases and L-3 Maritime Systems (formerly Henschel) internal research, other US Armed Forces research, doctrines, and rapid advancements in technology, has resulted in a solution that will fulfill the needs of the US Naval Fleet. The release of the Unified Capability Requirements 2008 and the follow on Change 2 has brought the direct support for end device Assured-Services SIP, allowing for a complete solution for the US Navy. The newer technologies have been fielded already with the US Army as well as the US Marines. Unified information on the network, including communications and critical data, will enhance the ability of sailors to meet their responsibilities; and will result as well in the consolidation of devices with which they need to interface. This will also result in reduced training and reduced equipment, thereby reducing space and wiring requirements. While not at their stations, the wireless phones will give the sailor the same comprehensive functionality as a desk phone. The use of open source and open architecture products will keep the US Navy from being tied to a single vendor because of proprietary products. It will allow multiple vendors and their products to be combined into a unified network that will support voice, data and video. This direction is achievable; it needs to be implemented so that reliability for our sailors is never compromised in any tactical, as well as non-tactical, scenario.

This page intentionally left blank.

APPENDIX A Reallocating the Atmel Flash

A-1 Introduction

The *Unicoi* file system and software all use and reside on the same flash memory device. The memory must be configured to accommodate both. The Atmel Flash device on the BF-537 has only about 4MEG of space available. At times it was necessary to increase the amount of memory available to our AS-SIP application. Our application would not load at times during development because there was no longer enough space allocated for the image.

Reallocating the space is a relatively simple process. First, the software must be modified as follows.

A-1.1 Opening and Editing Source Header Files

1. Open and edit the source header file:

$INSTALL_DIR\fusion\refdesigns\voip\ip_phone\ports\cougar\bf537\source\atmel_dat_flash_port.c

The file will contain, among other things, the following macro definitions:

```
#define FFS_ATMEL_FLASH_PAGE_SIZE          528
#define FFS_ATMEL_FLASH_OVERHEAD           16
#define FFS_ATMEL_FLASH_BAUD_SETTING       3 /* Divide bus speed by 3 for spi clk
*/
#define FFS_ATMEL_FLASH_CHIP_SELECT        10 /* Port F pin 10 */

#define FLASH_BOOT_CODE_START_PAGE         0 /* 3 pages */
#define FLASH_CODE_IMAGE1_START_PAGE       3 /* 2979 pages */
#define FLASH_CODE_IMAGE2_START_PAGE       2982 /* 2979 pages */
#define FLASH_FILE_SYSTEM1_START_PAGE      5961 /* 2228 pages */
#define FLASH_SECURE_AREA_START_PAGE       8189 /* 3 pages */

#define FLASH_BOOT_CODE_RESERVED_PAGES     3
#define FLASH_CODE_IMAGE1_RESERVED_PAGES   2979
#define FLASH_CODE_IMAGE2_RESERVED_PAGES   2979
#define FLASH_FILE_SYSTEM1_RESERVED_PAGES  2228
#define FLASH_SECURE_AREA_RESERVED_PAGES   3
```

2. Before making any changes, determine how much space is currently allocated for our application, and how much space is available. The current space allocated to our application is determined by multiplying:

FFS_ATMEL_FLASH_PAGE_SIZE x FLASH_CODE_IMAGE1_RESERVED_PAGES
(528 x 2979) = 1,572,912 bytes

3. If our application is larger than 1,572,912 bytes, you will need to change the flash configuration. For the sake of example, let's say our application is now 1,610,000 bytes. To make a block large enough to hold this image we' need to add at least 76 new pages. This would give us a new *FLASH_CODE_IMAGE1_RESERVED_PAGES of 2979 + 76 = 3055*

We can check our configuration by again multiplying:

FFS_ATMEL_FLASH_PAGE_SIZE x FLASH_CODE_IMAGE1_RESERVED_PAGES
(528 X 3055) = 1613040

Since our new example image is 1,610,000 bytes, our new allocation has more than enough space.

4. Using our numbers from the previous example, change the following 2 lines to read as follows:

```
#define FLASH_CODE_IMAGE1_RESERVED_PAGES      3055   //+76 pages
#define FLASH_CODE_IMAGE2_RESERVED_PAGES      3055  //+76 pages
```

5. The new sizes obviously effects the **start pages** of our code and file system so we'll need to modify them as well by offsetting them with the number of pages we added (76). Edit the following 2 lines to read as follows:

```
#define FLASH_CODE_IMAGE2_START_PAGE          3058
#define FLASH_FILE_SYSTEM1_START_PAGE         6037
```

6. Now that the code image sections are properly configured we need to address the fact the new file system size will cause it to run into the secure area. To calculate the size of the file system, subtract:

FLASH_SECURE_AREA_START_PAGE - FLASH_FILE_SYSTEM1_START_PAGE
(8189 - 6037) = 2152

7. Using the flash file system size previous step edit the following line to read:

```
#define FLASH_FILE_SYSTEM1_RESERVED_PAGES     2152
```

At this point all of the code related to the firmware has been changed to meet the needs of the new and larger AS-SIP application image. We still need to do one more step to make the boot loader aware of the new memory allocations, specifically the new code start page.

8. Open for edit the following assembly language source file: $UNICOI_INSTALL_DIR\fusion\refdesigns\voip\ip_phone\ports\cougar\bf537\init\cougar_bootldr.asm

9. Change the following line to use the new F*LASH_CODE_IMAGE2_START_PAGE* start page(3058):

*#define IMG1_ADDR (3058*528)*

At this point we are done and will need to rebuild the project as previously detailed. Both the new boot loader as well as the AS-SIP image will need to be burned into memory however. Both a run through the **UnicoiReleaseCreator** utility as previously detailed. The new boot loader will require the "-b" option on the "load" command as shown in the following example:

load -b -h 192.168.0.123 cougar_bootldr.bin

The boot loader image is located under:

$INSTALL_DIR\fusion\refdesigns\voip\ip_phone\ports\cougar\bf537\init\debug\cougar_bootldr.ldr.

Note that the file system will be wiped out and need to be restored. Use the Unicoi telnet shell command: "putfile" to save copies of whatever configuration files you need so that they can be easily restored again with "getfile", for example:

Welcome to FUSION OS!
System: Cougar 3.4.0 (Mar 4 2011 13:55:28)

C:\>dir

Directory of C:

12/09/2010 04:46:18 <DIR> CONFIG
01/20/2011 04:56:50 <DIR> CERTS
0 File(s) 0 bytes
2 Dir(s) 1,110,528 bytes free

C:\>cd config
C:\config>dir

Directory of C:\config

12/09/2010 04:46:18 <DIR> .
12/09/2010 04:46:18 <DIR> ..
03/17/2011 04:57:06 1,362 VOIP.XML
01/08/2007 19:05:06 1,177 VE.XML
01/08/2007 19:05:00 1,739 NETWORK.XML
3 File(s) 4,278 bytes
2 Dir(s) 1,110,528 bytes free

C:\config>putfile -h 192.168.0.123 network.xml
Uploading "network.xml"... Be Patient!
Sending file using TFTP...
1739 bytes sent

1 files copied.

C:\config>

10. It may also be necessary to restore the MAC address. This is also done through the *Unicoi* telnet shell as follows:

Welcome to FUSION OS!
System: Cougar 3.4.0 (Mar 4 2011 13:55:28)

Username: admin
Password:

C:\>.advanced

ADVANCED>mac wan
WAN MAC Address: 00:18:17:00:05:1F
ADVANCED>
ADVANCED>mac wan 00:18:17:00:05:1F

The "mac wan" command by itself displays the current MAC address.
The 2nd "mac wan" command sets it.
The MAC address is located on the bottom of the Fanstel ST-3116 phones.

APPENDIX B VOICE QUALITY TESTING

B-1. Factory Test Results

Unicoi used Agilent J1987A_43 VQT to measure voice quality and package called NetDisturb to emulate network impairment scenarios such as latency, jitter, packet loss and bandwidth limitations. The following codes were tested by *Unicoi*: G.711A, G.711U, G.723-6.3, G723-5.3, G.726-16, G.726-24, G.726-32, G.726-40, G.729 and DVI4.

Unicoi repeated all test six times and took the average for the final results. The table below shows the results for a female voice under progressively worse network conditions.

Female Voice (AGAM1F01.WAV) Processing Mode	**KBPS**	**0% 7.5 ms 12.5 ms**	**1% 90 ms 110 ms**	**2% 180 ms 220 ms**	**3% 270 ms 330 ms**	**5% 370 ms 430 ms**
G.711 A-VAD	64	4.07/4.05	3.77/3.72	2.64/2.47	2.61/2.56	2.52/2.45
G711 u-VAD	64	4.09/4.08	3.80/3.77	2.60/2.47	2.74/2.72	2.78/2.71
G711 A	64	4.05/4.03	3.75/3.72	2.65/2.48	2.61/2.57	2.51/2.45
G.711 u	64	4.07/4.06	3.81/3.77	2.60/2.48	2.73/2.71	2.78/2.72
G723	6.3	3.12/3.08	2.98/2.90	2.58/2.51	2.59/2.49	2.26/2.17
G723	5.3	3.06/3.0	2.91/2.87	2.55/2.5	2.49/2.44	2.11/2.02
G726	16	2.66/2.66	2.59/2.58	2.04/1.98	1.72/1.57	1.92/1.84
G726	24	3.16/3.15	2.99/2.92	2.22/2.17	2.19/2.14	1.98/1.96
G726	32	3.59/2.58	2.49/2.40	2.47/2.39	2.27/2.21	2.18/2.07
G726	40	3.9/3.9	3.74/3.68	2.37/2.32	2.31/2.24	2.27/2.2
G.729AB	8	3.43/3.41	3.35/3.31	2.38/2.32	2.31/2.24	2.27/2.20

The table below shows the results for a male voice under progressively worse network conditions.

Male Voice (AGAM2M01.WAV) Processing Mode	**KBPS**	**0% 7.5 ms 12.5 ms**	**1% 90 ms 110 ms**	**2% 180 ms 220 ms**	**3% 270 ms 330 ms**	**5% 370 ms 430 ms**
G.711 A-VAD	64	4.07/4.05	3.77/3.72	2.64/2.47	2.61/2.56	2.52/2.45
G711 u-VAD	64	4.04/4.04	3.87/3.80	2.80/2.75	2.72/2.41	2.77/2.73
G711 A	64	4.06/4.05	3.78/2.72	2.63/2.46	2.61/2.55	2.54/2.47
G.711 u	64	4.04/4.02	3.85/3.80	2.81/2.75	2.73/2.41	2.79/2.74
G723	6.3	3.69/3.64	3.31/3.27	2.68/2.61	2.71/2.68	2.03/1.99
G723	5.3	3.68/3.61	2.97/2.92	2.99/2.96	2.44/2.40	1.91/1.86
G726	16	2.66/2.66	2.59/2.58	2.04/1.98	1.72/1.57	1.92/1.84
G726	24	3.51/3.47	3.43/3.40	2.38/2.31	2.20/2.04	2.34/2.27
G726	32	3.97/3.96	3.74/3.70	2.57/2.35	2.39/2.29	2.43/2.36
G726	40	3.9/3.9	3.74/3.68	2.43/2.35	2.40/2.37	2.26/2.02
G.729AB	8	3.68/3.67	3.45/3.37	2.50/2.34	2.41/2.33	2.44/2.41

The following table represents testing of the jitter buffer under progressively worse jitter conditions.

Mecho Delay Range	**G.711 u-Law VAD enabled**	**G.729AB VAD enabled**	**G.723.1 – 6.3k VAD enabled**
0 to 0 ms	4.10/4.09	3.68/3.67	3.69/3.64
0 to 10 ms	4.09/4.09	3.63/3.60	3.68/3.66
0 to 20 ms	4.13/4.10	3.64/3.63	3.60/3.55
0 to 30 ms	4.07/4.02	3.70/3.66	3.64/3.62
0 to 40 ms	4.11/4.05	3.64/3.62	3.68/3.64

Mecho Delay Range	G.711 u-Law VAD enabled	G.729AB VAD enabled	G.723.1 – 6.3k VAD enabled
0 to 50 ms	4.05/4.0	3.60/3.55	3.63/3.60
0 to 60ms	3.89/3.78	3.54/3.50	3.56/3.51
0 to 70 ms	3.74/3.66	3.50/3.47	3.39/3.10
0 to 80 ms	3.41/3.33	3.51/3.44	2.68/1.88
0 to 90 ms	3.05/2.87	3.40/3.36	3.25/2.58
0 to 100 ms	2.75/2.62	3.41/3.38	1.89/1.43

B-2. Test Plan

The scope of this test plan includes the interoperability of both AS-SIP and TDM based end instruments (phones) with a REDCOM SLICE 2100 SIP switch. Inter-operability testing between multiple switch/carries falls outside the scope of this test plan. The tests are all centric upon the proper operation of the AS-SIP end instrument only although failures not attributed to the AS-SIP end instrument are also noted. Applicable call flow tests are repeated for both AS-SIP to/from AS-SIP as well as TDM to/from AS-SIP end instruments. User authentication is assumed to be enabled. All tests were repeated for both IPV6 and secure mode. Basic call flow tests were performed with and without MLPP (Multilevel Precedence and Preemption). The nature of SIP required that some tests be verified with a network packet capturing tool. We used both Wireshark and application logging to verify SIP functionality like registration and also to determine the source of any issues encountered.

The following test plan covers the responsibilities of the end instrument only. Many of the tests, especially where preemption is concerned have numerous implications on the switch which are not covered here. Since 2 lines must be supported by out AS-SIP, only multi-line preemption support is tested here. Single line preemption is handled entirely by the switch. Except where noted, all preemption tests are assumed to have met the following general requirements:

1. 2 lines are supported by the AS-SIP phone.
2. Preemption only occurs when both lines are in use
3. Preemption occurs when the calls have the same priority domain.
4. The higher precedence calling party will also receive the precedence ring tone in all scenarios where the conditions for preemption have been met.
5. Precedence rules are independent of media type. This is to say that all media types are treated equally and have no impact on preemption.
6. The precedence of every inbound call will be displayed on the GUI (screen) whenever present.

Test No.	Test Case	Comments
AS001	Register with switch	Verify with Wireshark, also verify appropriate user authentication
AS002	Auto registration	Verify EI is re-registered automatically at least 2 different intervals
AS003	Register with switch, intentionally foul the user authentication password	Verify the EI does not register using Wireshark and or switch admin
AS004	Off hook	Verify audible dial tone, transition to off hook alarm(fast busy) and appropriate display indicator that a line is in use.
AS005	On hook	Verify any line display indicators
AS006	Make call, near end hang up before answer Repeat for both SIP to SIP and SIP to TDM	Verify all audible indicators, dial tone, ringback and call clearing from near on hook. Verify PRACK with Wireshark.
AS007	Make call, near end hang up after answer Repeat for both SIP to SIP and SIP to TDM	Verify all audible indicators, dial tone, ringback and call clearing from near on hook. Verify PRACK with Wireshark.
AS008	Make call, near end holds the call Repeat for both SIP to SIP and SIP to TDM	Verify all audible indicators that the call is on hold
AS009	Make call, near end holds the call and retrieves the call Repeat for both SIP to SIP and SIP to TDM	Verify all audible indicators that the voice path has been re-established
AS010	Make call, far end holds the call Repeat for both SIP to SIP and SIP to TDM	Verify all audible indicators that the call is on hold
AS011	Make call, far end holds the call and retrieves the call Repeat for both SIP to SIP and SIP to TDM	Verify all audible indicators that the voice path has been re-established
AS012	Make call, near end holds the call, near end hangs up Repeat for both SIP to SIP and SIP to TDM	Verify that the held call is unaffected and retrievable

AS013	Make call, near end holds the call, far end hangs up Repeat for both SIP to SIP and SIP to TDM	Verify that the caller phone clears properly, and any visual line indicators are appropriate.
AS014	Make call, far end holds the call, near end hangs up Repeat for both SIP to SIP and SIP to TDM	Verify proper line clearing.
AS015	Make call, far end hangs up after answer Repeat for both SIP to SIP and SIP to TDM	Verify proper line clearing.
AS016	Receive a call no answer Repeat with both SIP and TDM callers	Verify audible ringback and visual line activity indicator
AS017	Receive a call and answer Repeat with both SIP and TDM callers	Verify audio path
AS018	Receive a call and answer, near end on hold Repeat with both SIP and TDM callers	Verify audio path
AS019	Receive a call and answer, near end on hold the retrieve the call Repeat with both SIP and TDM callers	Verify audio path
AS020	Receive a call and answer, far end on hold Repeat with both SIP and TDM callers	Verify audio path
AS021	Receive a call and answer, far end on hold and retrieve the call Repeat with both SIP and TDM callers	Verify audio path
AS022	Make a call and place it on hold/off hold Rapidly at least 6 times finishing in the off hold state	Verify audio path
AS023	Make a call and place it on hold/off hold Rapidly at least 6 times finishing in the on hold state	Verify audio path
AS024	Receive a call and answer, far end on hook Repeat with both SIP and TDM callers	Verify Audio path
AS025	Receive a call and answer, near end on hook Repeat with both SIP and TDM callers	Verify Audio path

AS026	Blind transfer a call from a SIP phone to a SIP phone: SIP station A calls B, B blind transfers A to C. Repeat test hanging up A first, then C first.	Verify phones A&C are connected with a good audio path. Verify station B is no longer connected to either call and now has dial tone if still off hook. Re-verify A&C after B goes on hook.
AS027	Blind transfer a call from a TDM phone to a SIP phone: TDM station A calls SIP station B, B blind transfers A to C. Repeat test hanging up A first, then C first.	Verify phones A&C are connected with a good audio path. Verify station B is no longer connected to either call and now has dial tone if still off hook. Re-verify A&C after B goes on hook.
AS028	Blind transfer a call from a SIP phone to a TDM phone: SIP station A calls TDM station B, B blind transfers A to C. Repeat test hanging up A first, then C first.	Verify phones A&C are connected with a good audio path. Verify station B is no longer connected to either call and now has dial tone if still off hook. Re-verify A&C after B goes on hook.
AS029	Attended transfer a call from a SIP phone to another SIP phone: SIP station A calls SIP station B, B places A on hold and calls C. B takes A off hold.	Verify all 3 parties are now connected with good audio paths.
AS030	Attended transfer a call from a SIP phone to another SIP phone: SIP station A calls SIP station B, B places A on hold and calls C. B takes A off hold. A hangs up first.	Verify audio paths of all 3 parties are correct.
AS031	Attended transfer a call from a SIP phone to another SIP phone: SIP station A calls SIP station B, B places A on hold and calls C. B takes A off hold. B hangs up first.	Verify audio paths of all 3 parties are correct.
AS032	Attended transfer a call from a SIP phone to another SIP phone: SIP station A calls SIP station B, B places A on hold and calls C. B takes A off hold. C hangs up first.	Verify audio paths of all 3 parties are correct.

AS033	Attended transfer a call from a TDM phone to a SIP phone: TDM station A calls SIP station B, B places A on hold and calls C. B takes A off hold.	Verify all 3 parties are now connected with good audio paths.
AS034	Attended transfer a call from a TDM phone to a SIP phone: TDM station A calls SIP station B, B places A on hold and calls C. B takes A off hold. A hangs up first.	Verify audio paths of all 3 parties are correct.
AS035	Attended transfer a call from a TDM phone to a SIP phone: TDM station A calls SIP station B, B places A on hold and calls C. B takes A off hold. B hangs up first.	Verify audio paths of all 3 parties are correct.
AS036	Attended transfer a call from a TDM phone to a SIP phone: TDM station A calls SIP station B, B places A on hold and calls C. B takes A off hold. C hangs up first.	Verify audio paths of all 3 parties are correct.
AS037	Attended transfer a call from a SIP phone to a TDM phone: SIP station A calls TDM station B, B places A on hold and calls C. B takes A off hold.	Verify all 3 parties are now connected with good audio paths.
AS038	Attended transfer a call from a SIP phone to a TDM phone: SIP station A calls TDM station B, B places A on hold and calls C. B takes A off hold. A hangs up first.	Verify audio paths of all 3 parties are correct.
AS039	Attended transfer a call from a SIP phone to a TDM phone: SIP station A calls TDM station B, B places A on hold and calls C. B takes A off hold. B hangs up first.	Verify audio paths of all 3 parties are correct.
AS040	Attended transfer a call from a SIP phone to a TDM phone: SIP station A calls TDM station B, B places A on hold and calls C. B takes A off hold. C hangs up first.	Verify audio paths of all 3 parties are correct.

AS041	Preemption test: set station A's priority to "Routine" via the softkey pad on the phone. Make a single call from station A to station B.	Verify that receiving station (B) is displaying the text version of the incoming priority call: "Routine". No preemption takes place.
AS042	Preemption test: set station A's priority to "Priority" via the softkey pad on the phone. Make a single call from station A to station B.	Verify that receiving station (B) is displaying the text version of the incoming priority call: "Priority". No preemption takes place.
AS043	Preemption test: set station A's priority to "Immediate" via the softkey pad on the phone. Make a single call from station A to station B.	Verify that receiving station (B) is displaying the text version of the incoming priority call: "Immediate". No preemption takes place.
AS044	Preemption test: set station A's priority to "FLASH" via the softkey pad on the phone. Make a single call from station A to station B.	Verify that receiving station (B) is displaying the text version of the incoming priority call: "FLASH". No preemption takes place.
AS045	Preemption test: set station A's priority to "FLASH OVERRIDE" via the softkey pad on the phone. Make a single call from station A to station B.	Verify that receiving station (B) is displaying the text version of the incoming priority call: "FLASH OVERRIDE". No preemption takes place.
AS046	Preemption test: Station A makes at "Routine" priority call to station B by prefixing the dialed number with the digits: '94'. Regardless of what the stations priority is set to in the GUI, set the stations priority level to something different than what the dial prefix indicates.	Verify that receiving station (B) is displaying the text version of the incoming priority call: "Routine". No preemption takes place.
AS047	Preemption test: Station A makes at "Priority" priority call to station B by prefixing the dialed number with the digits: '93'. Regardless of what the stations priority is set to in the GUI, set the stations priority level to something different than what the dial prefix indicates.	Verify that receiving station (B) is displaying the text version of the incoming priority call: "Priority". No preemption takes place.

AS048	Preemption test: Station A makes at "Immediate" priority call to station B by prefixing the dialed number with the digits: '92'. Regardless of what the stations priority is set to in the GUI, set the stations priority level to something different than what the dial prefix indicates.	Verify that receiving station (B) is displaying the text version of the incoming priority call: "Immediate". No preemption takes place.
AS049	Preemption test: Station A makes at "FLASH" priority call to station B by prefixing the dialed number with the digits: '91'. Regardless of what the stations priority is set to in the GUI, set the stations priority level to something different than what the dial prefix indicates.	Verify that receiving station (B) is displaying the text version of the incoming priority call: "FLASH". No preemption takes place.
AS050	Preemption test: Station A makes at "FLASH OVERRIDE" priority call to station B by prefixing the dialed number with the digits: '90'. Regardless of what the stations priority is set to in the GUI, set the stations priority level to something different than what the dial prefix indicates.	Verify that receiving station (B) is displaying the text version of the incoming priority call: "FLASH OVERRIDE". No preemption takes place.
AS051	Preemption test: No preemption should occur when the priority domains of calls differ. The resource priority indicator of a call is stored as 3 separate components within the phones software: Example: dsn-000000.6 "DSN" is the network domain. "000000" is the priority domain. "6" is the priority indicator (FLASH) Place 2 Routine priority calls to station A using the default priority domain of "000000". Place a 3rd call to station A with a priority domain of "000001" and a the highest priority: FLASH OVERRIDE	Verify the priority domains using Wireshark or any similar network snooping tool. Verify the 3rd priority call with a priority domain of "000001" and a priority of "FLASH OVERRIDE" was rejected with a busy here and no preemption occurred

AS052	Preemption test: Station A makes a “routine” priority call to station D. Station B makes a “priority” priority call to station D. Station C makes a “FLASH” priority call to station D. All calls have the same priority domain.	Verify that the lowest priority call (station A) receives a continuous preemption tone until the on-hook is received. Station A should also display an indication of the preemption, we display “Preempted”. The called party, station D should also hear the preemption tone for a minimum of 3 seconds and after going on-hook should hear the precedence ring tone and be connected to the preempting caller (station C) Verify, using Wireshark or similar network snooping software, that the preempted caller (station A) also received a “Reason” header on the BYE method set to the following value: Reason: preemption ;cause=1 ;text=“UA Preemption”
AS053	Preemption test: Station B makes a “Routine” priority call to station A and A answers the call. Station A still has one line available. Station C makes a priority call higher than station B to station A.	No preemption should occur here. However, the called party station A should still produce the precedence ring tone. It is entirely up to the called party(A)on how to handle the higher precedence call. As always, the precedence of the incoming call should be displayed on the screen.
AS054	Preemption test: repeat the steps in test AS052 to produce a preemption scenario but make sure that the call to be preempted, the lowest priority call, is currently on hold.	In addition to the checks and verifications of test AS052, verify the held call received the required preemption tone.

AS055	Preemption test: Station B calls station A. Station A answers the call so that it now has one line in use. Station A places station B on hold. Station A now attempts to place an outbound call to station C on the 2nd line. Station C does not answer the call. Station D makes a priority call (any) to station.	Station A must terminate the incomplete outbound call. Send a BYE or CANCLE with a Reason header of: Reason: preemption ;cause=1 ;text="UA Preemption". There is no reason to play the 3 second preemption tone as the call has not yet been established.
AS056	Call forwarding: station A manually enables call forward "Unconditional" by prefixing the forwarding number with "*72". Example: *725553001. After dialing this forwarding sequence, Use another station to dial station A.	Verify that the GUI forwarding status was updated by the manual setting of call forward to unconditional. Verify that the station used to dial station A was forwarded to the desired station, in this example "3001". No preemption should ever take place in unconditional mode.
AS057	Call forwarding: repeat test AS056. Now turn call forward "Unconditional" off by manually dialing "*73". Dial station A from another station.	Verify that the GUI forwarding status was updated by the manual disable of call forward unconditional. GUI screen should now indicate that call forwarding is "OFF". Verify that the station used to dial station A now reaches station A and rings the phone.
AS058	Call forwarding: station A manually enables call forward "No Answer" by prefixing the forwarding number with "*92". Example: *925553001. After dialing this forwarding sequence, Use another station to dial station A	Verify that the GUI forwarding status was updated by the manual setting of call forward to no answer. Verify that the station used to dial station A was forwarded to the desired station, in this example "3001" after a user configurable number of rings(5)
AS059	Call forwarding: repeat test AS058. Now turn call forward "No Answer" off by manually dialing "*93". Dial station A from another station.	Verify that the GUI forwarding status was updated by the manual disable of call forward no answer. GUI screen should now indicate that call forwarding is "OFF". Verify that the station used to dial station A now reaches station A and rings the phone.

AS060	Call forwarding: station A manually enables call forward “Busy” by prefixing the forwarding number with “*90”. Example: *905553001. After dialing this forwarding sequence, Use another station to dial station A	Verify that the GUI forwarding status was updated by the manual setting of call forward to busy. Verify that the station used to dial station A was forwarded to the desired station, in this example “3001”
AS061	Call forwarding: repeat test AS060. Now turn call forward “Busy” off by manually dialing “*91”. Dial station A from another station.	Verify that the GUI forwarding status was updated by the manual disable of call forward busy. GUI screen should now indicate that call forwarding is “OFF”. Verify that the station used to dial station A now reaches station A and rings the phone.
	Call forwarding: repeat tests AS060, AS058 and AS056 except use the GUI interface softket to cycle through all 4 call forwarding states: OFF, BUST, NO ANSWER, UNCONDITIONAL.	

Acronyms and Abbreviations

AS-SIP	Assured Services Session Initiation Protocol as defined by the department or defense Unified Capabilities Requirements document.
FCL	Fusion Common Library
GNU	Recursive acronym for "GNU's Not Unix!"
GPL	General Public License
IETF	Internet Engineering Task Force
IPSEC	Internet Protocol Security
IPV4	Internet Protocol Version 4
IPV6	Internet Protocol Version 6
JTAG	Joint Test Action Group
LGPL	Lesser General Public License
LSC	Local Session Controller
PRACK	Provisional Response Acknowledgement (RFC 3262)
RFC	Request For Comments
RTOS	Real Time Operating System
RTP	Real Time Protocol
SIP	Session Initiation Protocol
SRTP	Secure Real Time Protocol
SSL	Secure Sockets Layer
TDM	Time Division Multiplexing

TLS	Transport Layer Security
UCR	Unified Capabilities Requirements