# Sontaran Status Update 1

SECRET//NOFORN

## Date

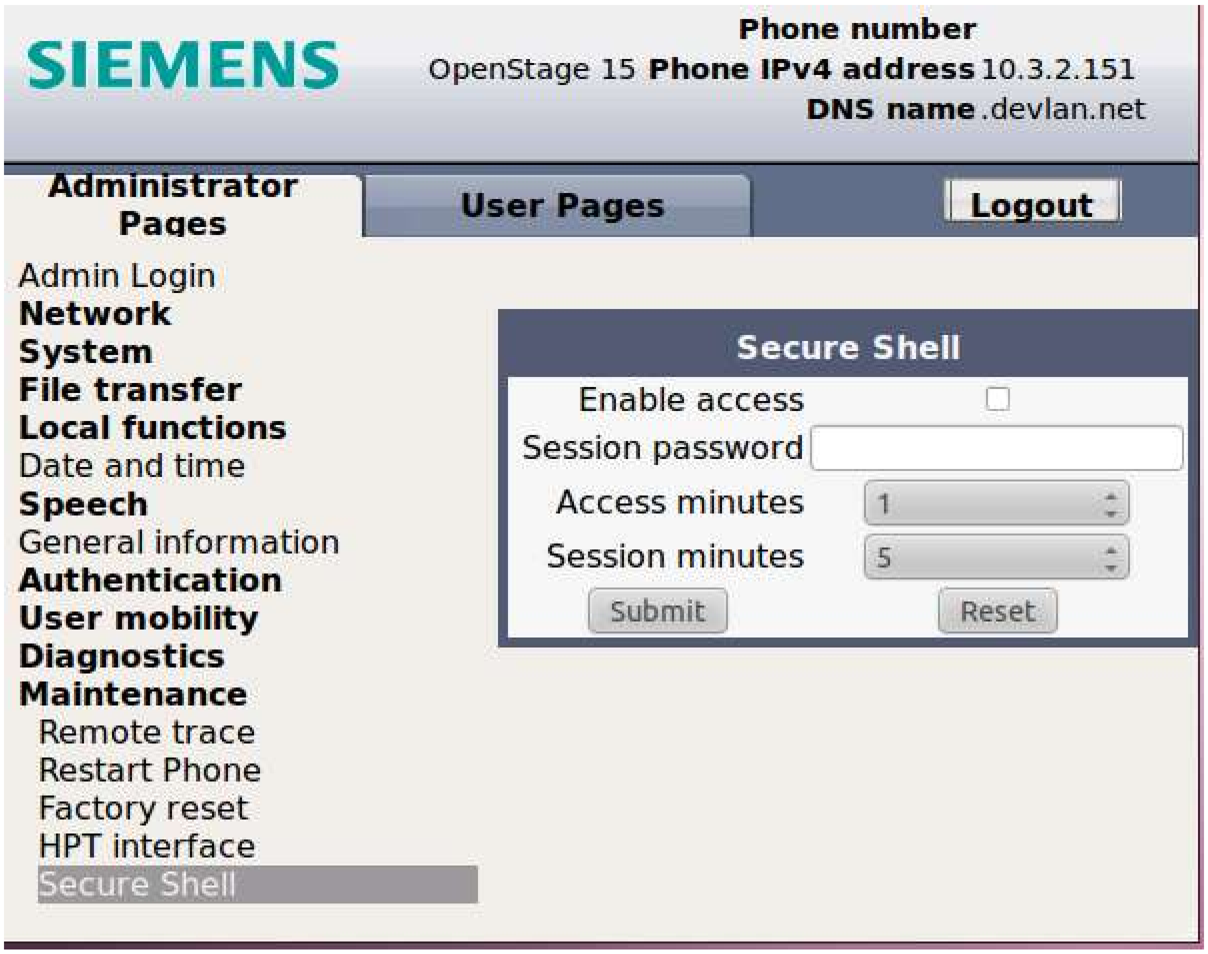
**07 June 2013**

## 1 - Status Overview:

- SDK unpacked and built
- SDK toolchain and my toolchain both build user and kernel code
- Bill P. provided time saving scripts and knowledge – persistent root access with tiny shell
- System and SDK familiarization
- I speculate we don't have all the documentation need for the ifx_mps driver – this driver is present on the system.
- I can build and load the TAPI drivers (not there by default), but have yet to get them to respond with useful information.
- I do not have any insight into how the Siemen's application(s) interact with the voice co-processor
- Limited "survey" data can be pulled from the proc filesystem
- Obvious (to me) approaches have been nearly exhausted
- Access to a call manager is becoming critical

## 2 - Establishing Initial & Recurring Access:

- enable SSH through web interface (web interface uses HTTPS) – login with credentials configured in the web interface



OpenStage 15 **Phone number**
**Phone IPv4 address** 10.3.2.151
**DNS name** .devlan.net

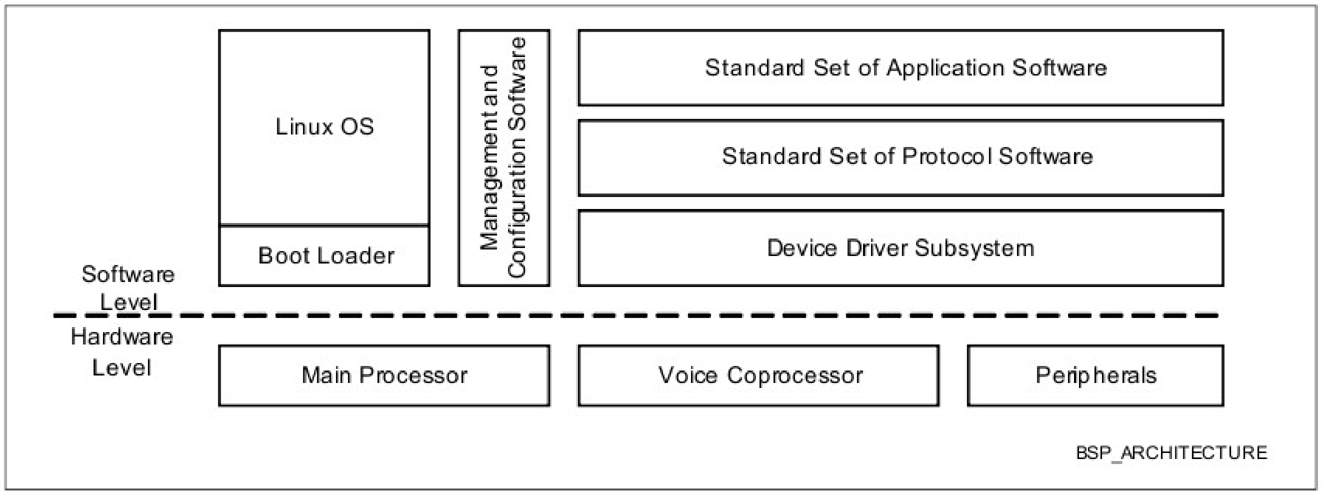
**Secure Shell**

| | |
|---|---|
| Enable access | ☐ |
| Session password | |
| Access minutes | 1 |
| Session minutes | 5 |

- use TFTP to upload tshd (tiny shell)
  - write tshd to writable directory (/usr/sbin)
  - add tshd.cmd to webroot -- un-authenticated execute trick found by Bill P.
  - add tshd to start-up script
- persistent root shell access!

## 3 - General Design of Siemens OpenStage 15 HFA VoIP Phone:

- Two processors -- 1 dedicated to phone functionality (voice coprocessor), 1 running Linux and providing interface to User (main processor)



- Communication between two processors through Linux OS device drivers.  There is a "command" channel with multiple "data" channels (4-8+)
- Linux device drivers expose kernel functions; or ioctls to user-space apps once /dev/device is opened
- Several drivers provide access to voice coprocessor:  ifx_mps and drv_tapi
  - ifx_mps - present on the Siemens phone and provides a low-level interface
  - drv_tapi - NOT present on the Siemens phone.  Provides a higher-level interface that is better documented.

## 3a - ifx_mps (multi-processor system):

- Example API functionality:
  - MPS event registration
  - MPS mailbox read and write (most important -- although commands to pass are not documented – does such a document exist?)
  - MPS get status
  - MPS get command history
  - MPS reset, restart
  - MPS download [firmware]

- SDK provides limited example user-space applications.  The provided stream application works where the tone generation is the easiest to demonstrate:

```
stream application (May 28 2013, 12:19:17)



Usage: stream [OPTIONS]
Example: stream -f firmware.bin -T -a -L



Options:
-f <file>      Firmware file download
-e <codec>     Codec select
-a             Activate Streaming
-L             High level loop
-l             Local narrowband loop
-w             Local wideband loop
-p <dclfreq>   Local PCM loop
-t             Tonegenerator
-v             Print Firmware Version
-R             Restart VCPU
-r             Reset VCPU
-b             Print command history buffer
-u <IP>:<port> Start UDP Connection
-c <IP>:<port> Start TCP Client
-s <IP>:<port> Start TCP Server
```

- ifx_mps interface is cryptic.  In the stream application, for example, I don't understand where it gets the values it passes to the ifx_mps driver.  These values are hard-coded.  They are not defined in a header file, nor can I find them in the driver source code.  They appear to be passed directly to the voice processor without further interpretation.

```
static u32 dsp_write_cmderr_ack_message[] = { 0x0600e000 };
static u32 dsp_pcm_ctrl_message[]  = { 0x06000004, 0x80600000 };
static u32 dsp_pcm_chan_message[]  = { 0x0600010C, 0x80020000, 0x20002000, 0x00000000
};
```

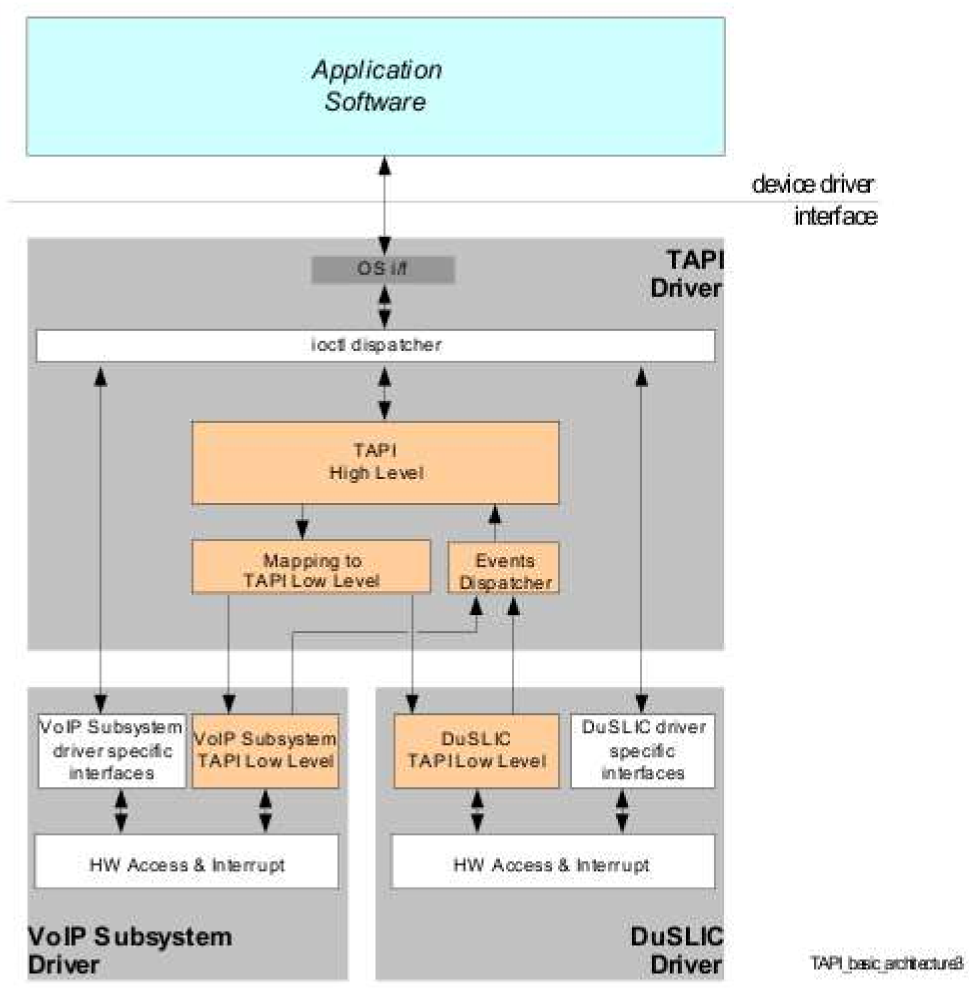
```
static u32 dsp_write_local_loop_message[] =
{
        0x06008118, 0x81040000, 0x20002000, 0x20002000, 0x00000000, 0x00000000, 0x00000000
};

static u32 dsp_enable_alm_message[] =
{
        0x06008118, 0x80080000, 0x20002000, 0x20000100, 0x00000000, 0x00001808, 0x00000000
};

static u32 dsp_write_TG1_message[] = { 0x06008304, 0x80000000 };
static u32 dsp_write_TG3_message[] = { 0x06008304, 0x00000000 };
static u32 dsp_write_TG2_message[] =
{
        0x0600D130, 0x47FB4000, 0x0000199A, 0xFFA31F3F, 0x20004000, 0x6E98E4E4, 0xE4E40400, 0x10800400, 0x20400400, 0x30200400, 0x40100400, 0x51800800, 0x00001000
};

static u32 dsp_read_version_register_message[] = { 0x8600E604 };
static u32 dsp_read_jb_stat_message[] = { 0x86007438 };
static u32 dsp_read_last_error_message[] = { 0x8600E408 };
static UINT32 dsp_write_audio_start_message[24][7] =
{
        /* Audio Channel Message */
        {0x06008118, 0x81040166, 0x20002000, 0x00000100, 0x00000000, 0x00001800, 0x00000000}, /*G711_A-Law*/
        {0x06008118, 0x81040166, 0x20002000, 0x00000100, 0x00000000, 0x00001800, 0x00000000}, /*G711_u-Law*/
        {},{},{},{},
        {},{},{},{},{},{},{},{},
        {},
        {0x06008118, 0x81040166, 0x20002000, 0x00000100, 0x00000000, 0x00001800, 0x00000000}, /*G729_AB*/
        {},
        {0x06008118, 0xC1040166, 0x20002000, 0x00000100, 0x00000000, 0x00001800, 0x0
0000000}, /*G722*/
        {},{},
        {},{},
        {},{}
};

static UINT32 dsp_write_rtp_up_message[] =
{
        /* RTP Upstream*/
        0x06007124, 0x00000000, 0x00000800, 0x04050607, 0x08090a0b, 0x0c0d0e0f, 0x10
111213, 0x14151617, 0x18191a1b, 0x1c1d1e1f
};

static UINT32 dsp_write_rtp_down_message[] =
{
        /* RTP Downstream*/
        0x06007920, 0x08000405, 0x06070809, 0x0a0b0c0d, 0x0e0f1011, 0x12131415, 0x16
171819, 0x1a1b1c1d, 0x1e1f0000
};

static UINT32 dsp_write_coder_start_message[24][3] =
{
        {0x06006108, 0x84020802, 0x20002000},     /*Coder Channel Speech Compression*
/
        {0x06006108, 0x84020803, 0x20002000},     /*Coder Channel Speech Compression*
/
        {},{},{},{},
        {},{},{},{},{},{},{},{},
        {},
        {0x06006108, 0x84020812, 0x20002000},     /*Coder Channel Speech Compression*
/
        {},
        {0x06006108, 0xC4020814, 0x20002000},     /*Coder Channel Speech Compression*
/
```

```
        {},{},
        {},{},
        {},{}
};


/* Message for PCM channel configuration and activation */
static u32 dsp_pcm_alm_message[] =
{
        0x06008114, 0x80080000, 0x20002000, 0x20000100, 0x00000000, 0x00000800, 0x00
000000
};


static u32 dsp_pcm_interface_message[] = { 0x06000004, 0x80000000 };
static u32 dsp_pcm_channel_message[] = { 0x0600010C, 0x80020000, 0x20002000, 0x000000
00 };
```

## 3b - drv_tapi Example API:

- In a sample application, I attempt to use the TAPI interface, but have yet to achieve any useful functionality. Opening the device files are successful and return valid file descriptors.  After opening the devices (command and data channels), I send commands that would:
  - initialize channels (claims success, but success seems to be the default return value -- also, not sure initialization is needed) -- Sec 2.9
  - register for events (like on/off-hook) -- Sec 2.7
  - query for hook status -- Sec 4.1.9.9
  - play tones -- Sec 3.5.2
- Could try other functionality, possibly other prerequisite configuration needed that I'm not doing or not doing properly, such as:
  - channel initialization
  - attaching / associating data channels to resources
  - setting up audio modes
- Additional functionality one could try:
  - detect room noise -- Sec 3.2.3
  - configure hook timing thresholds -- Sec 2.8.1
  - attempt to ring phone -- Sec 3.6

- This interface is NOT present in the Siemens.  Requires 2+ modules (drv_tapi, drv_vmmc).  Used SDK to build drv_tapi, drv_vmmc, and hapi.o.  drv_tapi and drv_vmmc load without problems on the phone.  hapi.o fails to load and reports error.



- According to documentation, hapi.o is a parrallel interface -- supposedly higher level, but of less interest.  The fact that it doesn't load seems to be unimportant.  After drv_vmmc is loaded, the appropriate device files are created under /dev.

**EASY 21653 INCA-IP2**
**VoIP IPP Subsystem**

**CONFIDENTIAL** **Introduction**

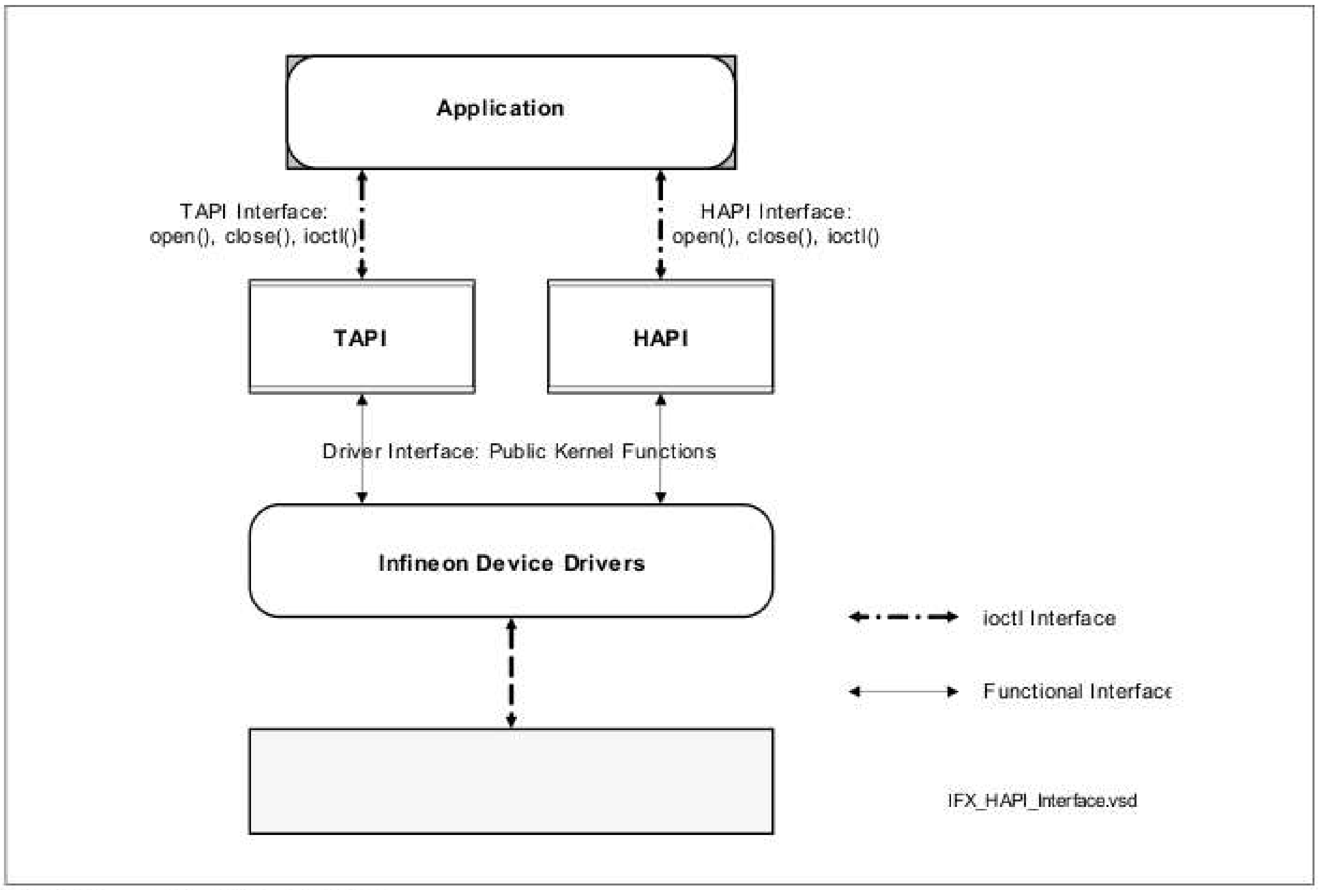

Application
TAPI Interface: open(), close(), ioctl()
HAPI Interface: open(), close(), ioctl()
TAPI
HAPI
Driver Interface: Public Kernel Functions
Infineon Device Drivers
ioctl Interface
Functional Interface
IFX_HAPI_Interface.vsd

**Figure 1 IFX_HAPI Interface**

# 4 - Next Steps?

- Acquire call manager. Phone can't find call manager and reports "Telephony is down (HO2)". Could this impact functions available? Are some modules loaded only when connected to the call manager?
- Is training available and an option?
- Dynamic analysis – who opens the /dev files (grep, lsof, /proc/pid/fd); hook & trace libc library calls to open, close, read, write, ioctl w/ LD_PRELOAD
- Diagnose HAPI load failure – try to load HAPI without TAPI....

- I know little about the Phone Application (PA) produced by Siemens (appears to be called "Opera") and I could begin reversing these.  These predominantely take the form of:

```
○  (93 processes) SvcConfig services.conf -startLogDaemon -logAll V2 R0.92.0      HFA  1
   20822

○  (58 processes) PhoneletLauncher messageshfa.phd V2 R0.92.0      HFA  120822 WP1 Sieme
   ns HFA GB en DD.MM.YYYY 24HR 0
```

- Google searches on Opera and ifx_mps have yielded no additional insight; however, a fresh look with what we now know may put open source information into better context.
- Try to load my drivers at boot time, first
- Continue to study at source – specifically, drv_vmmc and drv_tapi
- ~~Check older? phones for presence of /dev/vmmc1X~~
- Continue to study the sample applications that use the ifx_mps interface
- Continue trying to get an application using the TAPI interface to work
- Reverse voice processor firmware (strings, binwalk provide no additional insight)