SENSORY®

SoundSource™ Technology

DESIGN NOTE

SoundSource™ (SS) is a technology in Sensory's FluentChip™ library which creates the ability for an application to track the position of a user's voice at very low cost. This tracking information can be used to drive a mechanical system (such as a swiveling head), to indicate position (such as on a display) or in other applications.

SS is designed to work in conjunction with Sensory's Text-to-Speaker-Independent (T2SI™) recognition technology, as well as a stand alone technology. This design note describes an SS implementation in a T2SI environment driving a swiveling head.

Please note: this document assumes the user is already familiar with programming RSC-4x applications, especially T2SI recognition technology. Code examples in this document are shown in C. Refer also to the assembly and C samples in the FluentChip™ (FC) library release.

1. Hardware discussion:

SS technology is a combination of technology software and a small 2 channel external circuit. The following schematic will be used in describing a typical swiveling head motor based application:

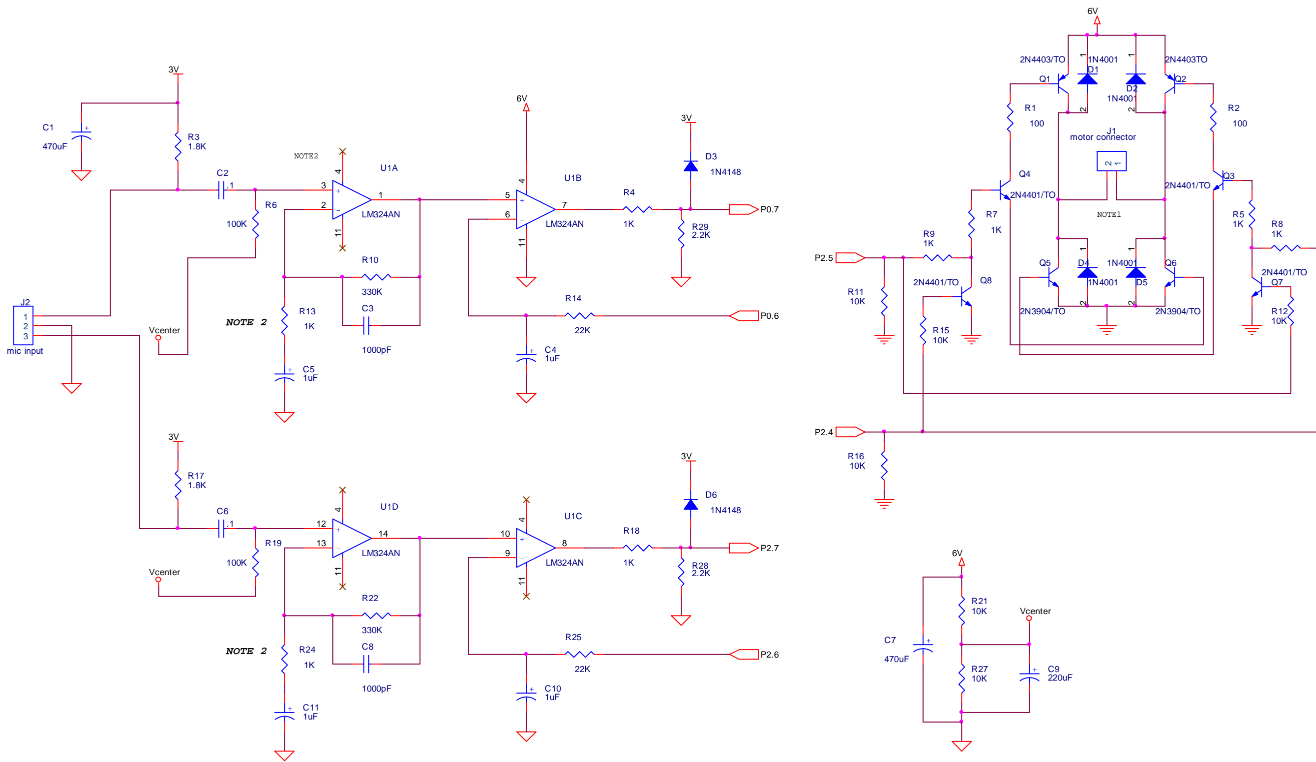

Fig. 1

NOTE: This specific schematic is designed for a nominal 6V power supply (for example, 4 cell battery) and motor application. It also assumes the RSC-4x is running at $3V_{dd}$. If significantly lower voltage operation (<4.8V) is required, the U1 opamp may need to be a low voltage, rail-to-rail out device (for example, National Semiconductor's LMV324 Operational Amplifier.) If the motor system design runs at less than the RSC-4x V_{dd}, further circuit design may be necessary. That case is not discussed in this document.

Also, note that this schematic shows a discrete transistor H-bridge. This particular design may not be optimal for all applications.

SS requires two microphones and amplifier circuits. Both microphones should be the same make and model omnidirectional elements to ensure similar frequency response and sensitivity.

Typically the microphones will be electret elements, connected to J2 in the above schematic, and mounted on a rotating platform. This is the design addressed in this document.
Important Note: One of the microphones may also be used as the recognition input microphone, IF the following design constraints are properly addressed:

1. The mechanical system should not transmit noise to the microphone as it rotates or reaches a mechanical limit.
2. The microphone should not rotate to a position where it is blocked from the user's voice.
3. The power supply must be "stiff" enough to prevent voltage spikes (which may be generated when the motor starts/reverses) from being introduced into the microphone signals.
4. The recognition microphone is separately capacitor coupled into the RSC-4x. A .1uF coupling cap is acceptable.

If mechanical or power supply limitations don't allow meeting the above constraints, the recognition microphone should be a separate unit. See the RSC-4128 Data Sheet (80-0206) for examples of typical recognition microphone circuit design.

SS requires four dedicated port pins on two RSC-4x I/O ports. Each amplifier channel must use Pn.7 and Pn.6. For example, the schematic in Fig. 1 uses P0.7 and P0.6 for one channel and P2.7 and P2.6 for the other channel. The port lines must be defined in the project's "config.mca" file. See Software discussion. Two more port pins are required to drive the motor. This particular application uses P2.4 and P2.5, but any available pin could be used.

SS also requires that the distance between the two microphones be fixed. The current release supports two distance ranges: approx. 2.5" to 3.5" and approx. 1.5" to 2.5". The desired range is set in the project's "config.mca" file.

2. Software discussion:

IMPORTANT NOTES:
1. The software example below was targeted at a mechanical system using a ~76:1 gear ratio between the motor and the moving head. Depending on your mechanical requirements, you may require significant variation in motor drive circuitry and code, defined constants, and lookup table. This code is for reference only.

2. While SoundSource can be run independently, this example code was written to run concurrently with Sensory's T2SI. For fastest T2SI response time, it is recommended that (on RSC-4x platforms that support 0 wait-states) the code be built with 0 wait-states.

3. If SS is used in a design that also uses one of the RSC-4x low-power sleep modes, be sure to reconfigure P0.7 and P2.7 as Hi-Z inputs to avoid creating a current path from Vdd through the internal pullup resistors and R28 & R29 to Ground.

4. The power to the external circuitry should also be removed before entering a low power sleep mode. The purpose is to prevent current drain through the microphones, current in the active external amplifiers, and to prevent the comparator outputs from driving R28 & R29. This design detail is not addressed in this note.

The following sections describe the code in various files necessary to perform SoundSource concurrently with T2SI. The example code comes from the C language "t2siss" sample code included in Sensory's FluentChip™ release. The release also includes RSC assembly code to perform the same functions.

 © 2007 Sensory Inc.

Configuring SS (config.mca):

The following is a config.mca fragment compatible with the hardware schematic in Fig 1.

```
SOUND_SOURCE .EQU 1              ; Special setup for pins, cmpCtl
                                 ; SEEPROM, SERFLASH

SS_MIC_DISTANCE .EQU 1           ; 1 = 3" between mics, 0 = 1.5"
          .PUBLIC SS_MIC_DISTANCE

; NOTE: ACHAN_IN and BCHAN_IN must both use pin 7 in some IO port.
; ACHAN_OUT and BCHAN_OUT must both use pin 6 in the ports used for ACHAN_IN and BCHAN_IN.
; To avoid conflicts with other devices, this version of config
; has many standard Demo/Eval devices disabled.

DefPort ACHAN_IN, p0, 7, INPUT, NONE
DefPort ACHAN_OUT, p0, 6, OUTPUT, 0
DefPort BCHAN_IN, p2, 7, INPUT, NONE
DefPort BCHAN_OUT, p2, 6, OUTPUT, 0
DefPort ADIR, p2, 5, OUTPUT, 0
DefPort BDIR, p2, 4, OUTPUT, 0
```

Invoking SS from the application program (t2siss.c):

SS is typically started immediately before a T2SI sequence is invoked and stopped immediately after the T2SI sequence returns. For example:

```
       _SoundSourceStart();
       error= _T2SI((long)&ACOUSTIC_MODEL,(long)&SRCH_TRIGGER,2,60,T2SI_DEFAULT_TRAILING, &results);
       _SoundSourceStop();
```

NOTE: The core of the SS operation takes place in a T3 irq handler and in the T2SI callout handler. `_SoundSourceStart` configures Timer3 for 24KHz and enables its interrupts.

SS sample gathering and motor control (timer3isr.mca):

The T3 interrupt handler runs at 24KHz, and is responsible for gathering data samples from both channels. This is done by calling the _SoundSource() function.

In our example, the T3 handler also generates the PWM used to drive the motor. The T2SI callout handler calls SoundSourceAction() to determine the PWM pulse width which it assigns to "chopper". The motor drive pulse time is determined by the comparison of chopCnt and chopper in the T3 handler.

```
// SoundSource global variables used by T3 irq handler and the T2SI callout handler
uchar chopper = 0;                // PWM value
uchar chopCnt = 0;                // PWM counter
uchar direct  = 0;                // NO_GO = 0, GO_A = 1, GO_B = 2
uchar stretch = 0;                // counts # of readings above threshold
```

Example T3 IRQ handler

NOTE: Because Timer3 needs to run at 24KHz it is written in assembly (timer3isr.mca). This C code just illustrates what the assembly code does.

```
void timer3_isr(void) {
    _SoundSource();        // take a SoundSource reading
    if (++chopCnt) {
       if (chopCnt >= chopper)) {
          StopADIR;        // allow a 1-255 count pulse width
          StopBDIR;        // then stop all movement
       }
    }
    else  {                // when chopCnt wraps, restore direction bits
       StopADIR;           // clear both bits
       StopBDIR;
       if (direct==GO_A) // then possibly set one
          GoADIR;
       else if (direct==GO_B)
          GoBDIR;
    }
    irq = ~T3_MASK;        // clear the interrupt (T3_MASK=0x10)
}
```

Converting SS data into motor control data (hmm_hc.c):
The SS position data is obtained at a 36Hz rate and processed by a SS-specific T2SI callout handler (hmm_hc.c). BfeHandler differs from the standard handler because it calls the application function `SoundSourceAction()`, described below. In our sample, BfeHandler doesn't check for user aborts, and always return 0 so that T2SI will not abort.

```
//--------------------------------------------------------------------------
// _BfeHandler - called at block rate (27.4 msec, 36Hz)
//   Calls SoundSourceAction for user-defined action
//   NO check for user abort; ALWAYS returns 0, meaning T2SI should not abort

char _BfeHandler(void)
{
  SoundSourceAction();
  return 0;
}
```

SoundSourceAction function (t2siss.c):
This function uses SS results to generate application specific motor control data.

First, `_SoundSourceReady(uint)` is called to check if enough SS data samples have been obtained to make position decisions. In this example, the value of this argument is the constant SS_TICKS. When enough samples have been obtained, `_SoundSourceReady()` stops the T3 interrupt to allow time for position data to be obtained and motor control variables to be updated without T3 interrupts. The position data is returned by `_SoundSourceResults()`. The result value (`int results` in this example) is a signed value. It contains both the direction of the source's detected position by sign, and how far off normal the microphones' axis is to the source. A higher value indicates that the angle is further from normal. This is the SS data that the application uses to update the motor control variables. Once they are updated, `_SoundSourceResume()` is called to reenable T3 interrupts and start accumulating a new SS result.

In our example, designed for our specific hardware and response requirements, we desire a non-linear response to curve the position data to our specific mechanical design's mass, inertia, gear ratio, and damping. To accomplish that, we can use a lookup table. The lookup table's value is then passed to the PWM generator (in the T3IRQ handler) via the variable "chopper".

This discussion includes a sample table. This table follows a square law and can never exceed 50% PWM duty cycle. This provides a relatively high drive value for strong _SoundSourceReturn() values, and exponentially weaker drive as we approach a neutral position. We do this to reduce the inertial overshoot due to the mass of the rotating head. For your specific mechanical design, you may find that a different table, or no table, is preferred.

We also use several "knobs" to optimize the system response. These may also be different in your design:

SS_TICKS: Since this determines how many samples are accumulated, it affects the sensitivity of the system. In effect, it is a gain control and jitter reducer. Large values also slow perceptible response time. This can also be useful to reduce response to impulse noise like claps or door slams.

SS_THRESHOLD provides noise immunity, helping to reduce hunting due to ambient noise and short impulse noise.

SS_STRETCH ensures that very short impulses don't skew the results. This helps reduce jitter.

SS_BIAS maps the response to a different part of the lookup table. Depending on the table, this can be used to control response speed and overshoot.

```
#define SS_TICKS     500 // # of calls to SS before results are returned; more means slower response
#define SS_THRESHOLD 10 // abs(SSresult) needed to move; higher means fewer small movements
#define SS_BIAS      60 // add to pre-curve abs(SSresult) before doing table lookup
#define SS_STRETCH    2 // number of post-curve results above SS_THRESHOLD needed to move
```

 © 2007 Sensory Inc.

<u>Example SoundSourceAction function:</u>

```
//---------------------------------------------------------------------------
// SoundSourceAction  ; NOTE: Called from hmm_hc callout(~36Hz rate)
//
// out: if SS results are available:
//       ADIR, BDIR bits set/cleared based on SS results (NOTE: Only 1 may be set)
//       direct reflects ADIR, BDIR
//       stretch, chopper, chopCnt updated
// calls: _SoundSourceReady, _SoundSourceResults, _SoundSourceResume
// uses:  stretch
//
void SoundSourceAction(void)
{
  int SSresult;
  uint temp;

    // If a complete reading is available
    if (!_SoundSourceReady(SS_TICKS))
    {
        SSresult = _SoundSourceResults();
        temp = abs(SSresult);
        if (temp > SS_THRESHOLD) {  // if enough difference between channels
            temp = temp + SS_BIAS;  // add a bias for table lookup
            if (temp > PWMCOUNT)
               temp = PWMCOUNT;     // clamp maximum value
            temp = pwm_tbl[temp];   // get PWM value from a lookup table
        }

        if (temp > SS_THRESHOLD) {        // if PWM above threshold
            if(++stretch >= SS_STRETCH) { // and has been above for a while

                // set the direction (always stop first)
               if (SSresult >0) {
                    StopBDIR;
                    GoADIR;
                    direct = GO_A;
                }
                else {
                    StopADIR;
                    GoBDIR;
                    direct = GO_B;
                }
                // set the PWM duty cycle
                chopper = (uchar)temp;
                chopCnt = 0;
            }
        }
        // below threshold, so stop
        else {
            stretch = 0;
            StopADIR;
            StopBDIR;
            direct  = NO_GO;
            chopper = 0;
            chopCnt = 0;
        }
    }
    // Start accumulating the next reading
    _SoundSourceResume();
}

// table used in example code is curve of y=(x^2)/127

uchar cdata pwm_tbl[128] = {0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,2,2,2,2,3,3,3,4,4,4,
                        5,5,6,6,7,7,8,8,9,9,10,10,11,11,12,13,13,14,15,15,16,
                        17,18,18,19,20,21,22,22,23,24,25,26,27,28,29,30,31,32,
                        33,34,35,36,37,38,39,40,41,43,44,45,46,47,49,50,51,52,
                        54,55,56,58,59,60,62,63,65,66,68,69,71,72,74,75,77,78,
                        80,81,83,85,86,88,90,91,93,95,97,98,100,102,104,105,107,
                        109,111,113,115,117,119,121,123,125,127};

#define PWMCOUNT    127 // maximum value of table index (can not be >255)
```

© 2007 Sensory Inc.

The Interactive Speech™ Product Line

The Interactive Speech line of ICs and software was developed to "bring life to products" through advanced speech recognition and audio technologies. It is designed for cost-sensitive consumer-electronic applications such as home electronics, home automation, toys, and personal communication. The line includes the award-winning RSC-4x family of mixed signal processors and tools, the *VR Stamp™* 40-pin DIP module and tools, and the SC series of speech and music synthesis microcontrollers. It also includes our suite of software development kits, which are designed to run on non-Sensory processors and DSPs and support most popular operating systems.

RSC Microcontrollers and Tools

The RSC product family contains low-cost 8-bit speech-optimized microcontrollers designed for use in consumer electronics. All members of the RSC family are fully integrated and include A/D, pre-amplifier, D/A, ROM, and RAM circuitry. The RSC family can perform a full range of speech/audio functions including speech recognition, speaker verification, speech and music synthesis, and voice recording/playback. The family is supported by a complete suite of evaluation and development toolkits.

Speech Recognition Modules and Tools

The VR Stamp™ is a complete speech recognition module based on the RSC-4x and is ideal for fast design and easy production. A low-noise audio channel and standardized 40-pin DIP footprint allow rapid prototyping, less debugging, and shorter time to market. The *VR Stamp Toolkit* includes everything needed to get started today, including VR Stamps, Module Programming Board, sample applications, and a complete set of development tools featuring the Phyton IDE and limited-life C compiler, QuickSynthesis™ 4 and Quick T2SI-Lite™ speech tools.

SC Microcontrollers and Tools

The SC-6x product family features the highest quality speech synthesis ICs at the lowest data rate in the industry. The line includes a 12.32 MIPS processor for high-quality, low data-rate speech compression and MIDI music synthesis, with plenty of power left over for other processing and control functions. Members of the SC-6x line can store as much as 37 minutes of speech on-chip and include as many as 64 I/O pins for external interfacing. Integrating this broad range of features into a single chip enables developers to create products with high quality, long duration speech at very competitive price points.

FluentSoft™ Technology

FluentSoft™ Recognizer is the engine powering the FluentSoft™ SDK. It provides a noise-robust, large-vocabulary, speaker-independent solution with continuous digit recognition and word-spotting capabilities. This small-footprint software recognizes up to 5,000 words; runs on non-Sensory processors including Intel XScale, TI OMAP, and ARM9 platforms; and supports operating systems such as MS Windows, Linux, and Symbian.

3Dmsg™ Technology

3Dmsg's (www.3Dmsg.com) Animated Speech technology offers animated avatars with advanced speech recognition and synthesis capabilities for use in smartphones, language trainers, and kiosk applications. Facial expressions can be configured to show emotions and lip synchronization can be automatically driven from voice or text data.

Important notices:

Sensory Incorporated (Sensory, Inc.) reserves the right to make changes, without notice, including circuits, standard cells, and/or software, described or contained herein in order to improve design and/or performance. Sensory, Inc. assumes no responsibility or liability for the use of any of these products, conveys no license or title under any patent, copyright, or mask work right to these products, and makes no representations or warranties that these products are free from patent, copyright, or mask work right infringement, unless otherwise specified. Applications that are described herein for any of these products are for illustrative purposes only. Sensory, Inc. makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Safety Policy:

Sensory, Inc. products are not designed for use in any systems where malfunction of a Sensory, Inc. product can reasonably be expected to result in a personal injury, including but not limited to life support appliances and devices. Sensory, Inc. customers using or selling Sensory Incorporated products for use in such applications do so at their own risk and agree to fully indemnify Sensory, Inc. for any damages resulting from such improper use or sale.

SENSORY®

575 N. Pastoria Ave., Sunnyvale, CA 94085
Tel: (408) 625-3300 Fax: (408) 625-3350

© 2007 SENSORY, INC. ALL RIGHTS RESERVED.
Sensory is registered by the U.S. Patent and Trademark Office.
All other trademarks or registered trademarks are the property of their respective owners.