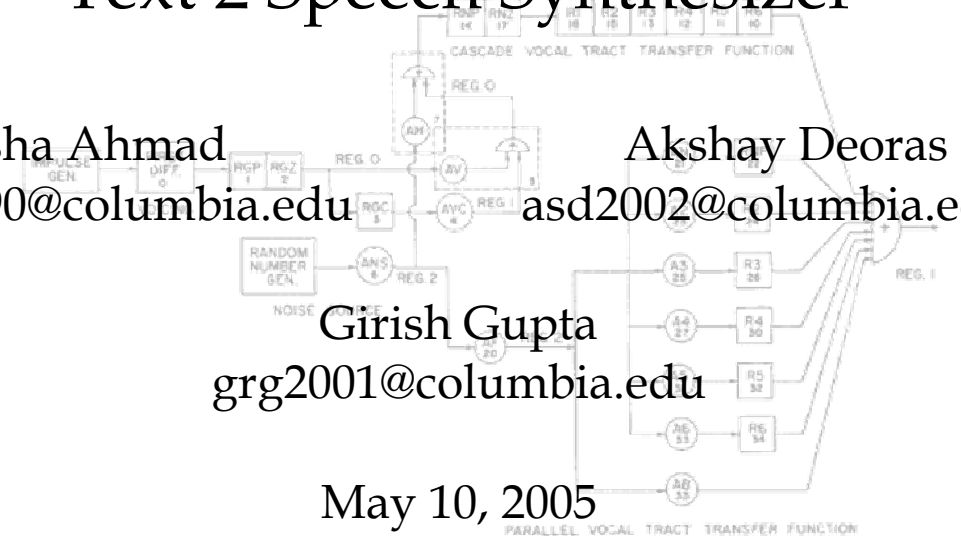
# Text 2 Speech Synthesizer

Aisha Ahmad
aa2190@columbia.edu

Akshay Deoras
asd2002@columbia.edu

Girish Gupta
grg2001@columbia.edu

May 10, 2005



# Contents

I. Project Proposal
- Overview and Implementation Challenges

II. Research
- Speech synthesis technology
- Text to Phoneme Translation

III. Project Design & Implementation
- Vocal Tract Model
- SRAM
- Audio Codec

IV. Miscellaneous
- Who did what?
- Lessons Learned
- Advice for the Future
- What went wrong?

V. References

VI. Source Code

# I. Project Proposal

**Overview**

We propose to build a text-to-speech synthesizer using the FPGA on the XESS XSB-300E. This project will require the use of the audio CODEC for analog output as well as extra memory space provided by the SRAM. The generation of speech and phonetic sounds can be implemented in a number of ways, three basic variations of which we are exploring are:

1. Concatenative speech consisting of fundamental phonetic sounds arranged according to input and retained in a phonetic library residing in on-board memory
2. Hard-wired vocal tract model implemented in VHDL, creating voiced and unvoiced speech sounds through a process of filtering and coefficient modification
3. Program code implementation of vocal tract model, shown in figure 1, with speech synthesis and input manipulation done in program compile

**Implementation Challenges**

Each implementation method involved a set of challenges we will have to face.

For method 1:

- Memory concerns, the storage of fundamental phonetic sounds in a library would take up much more memory than is available on the board
- Quality and robustness of synthesis, with only a predefined set of phonemes the output range will be limited, possibly leading to poor quality word synthesis

For method 2:

- Complexity, numerous algorithms for vocal tract modeling exist and have been researched but involve the implementation of complex filters making porting of such to VHDL difficult and leaving much room for error
- Compatibility, while much research has been done, locating a full top-down synthesizable hardware model has been rare and piecing together different models may result in performance issues

For method 3:

- Code size, synthesis requires the use of a variety of complex functions and large libraries creating issue of where to store the code
- Compatibility, same as for VHDL models

## II. Research

**Speech Synthesis Technology**

Research in the area of speech synthesis has been going on for decades. As we found out with our research, numerous models and theories exist for the best way implementing a speech synthesis system. Although the models seemed intuitive from a high level perspective (see figure 1), they quickly grew in complexity as we got closer to implementation. Notice in figure 2 the large number of filter coefficients needed.

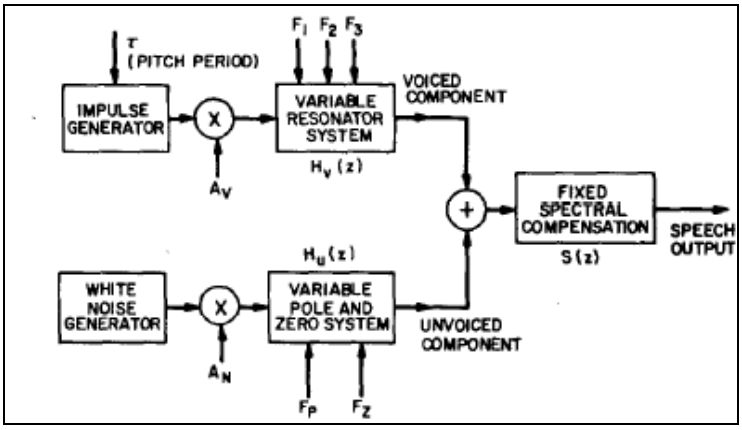



Figure 1: High level vocal tract model

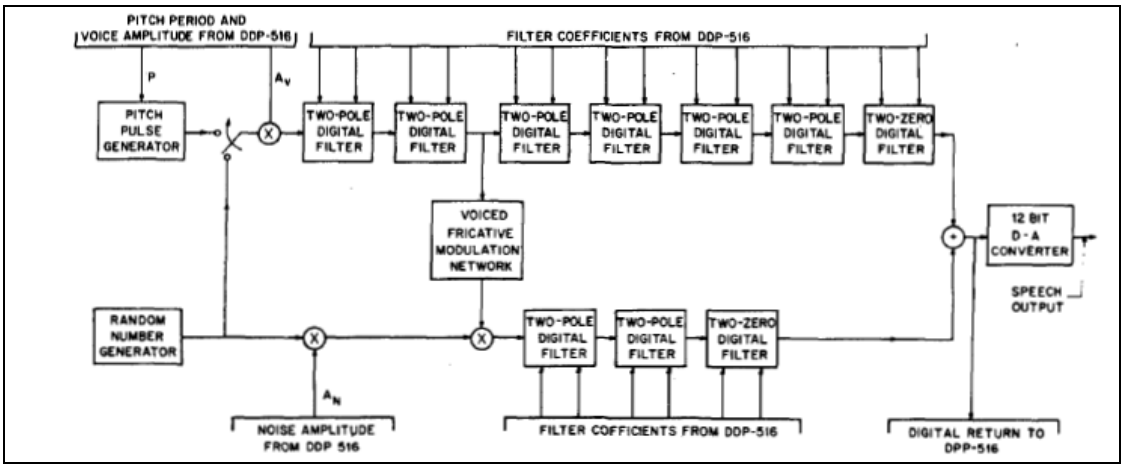



Figure 2: Corresponding digital hardware implementation

The variety of research in the topic lead us to observe that there exits three levels of Text to Speech technology: high, middle and low:

*High*: that is the initial text processing, finding the sentences (or whatever sized chunks you wish), dealing with punctuation, fonts, section titles etc.

*Mid*: translating words to phonemes, assigning duration, intonation tunes and prosodic phrasing
*Low*: synthesizing the waveform itself.

Our project focuses on the Mid and Low levels of the technology.
The middle level, has a very active following, papers on letter to sound rules, intonation, phrasing appear in many of the major computational linguistic conferences and journals.
For the low level the most popular and simplest methods of synthesis we found were:
*Formant Synthesis (as in the Klatt synthesizer )* - with the proper specification of parameters it is possible to make distinguishable human speech using filters
*Concatenative synthesis* - where sections of natural speech are concatenated to form utterances. A major difficulty here is to join them seamlessly.

While the research exists for these two areas, it is for the most part segmented, finding a complete working model for full text to speech synthesis was rare. One reason a model for synthesis of speech waveform that included the corresponding rules for coefficient generation was rare was that such model are only commercially available and are highly protected.

## III. Project Design

Our implementation of the speech synthesizer on XESS board is shown in figure 3. The diagram is a simple depiction of the overall structure of our projected goals. The proposition was that the FPGA would execute instructions downloaded SRAM. The program then outputs a bit stream of audio data to the audio codec in order to output the appropriate sound. After much deliberation spent on other complex ideas, we finally decided that this diagram was the best choice for this project. Many incomplete protected research endeavors bogged us down. Things just got confusing and complex because we tried to take different elements from various researchers. It was in the best interest of our group to stick to this simple diagram, which implemented our project, with the given limitations of the board, optimally.

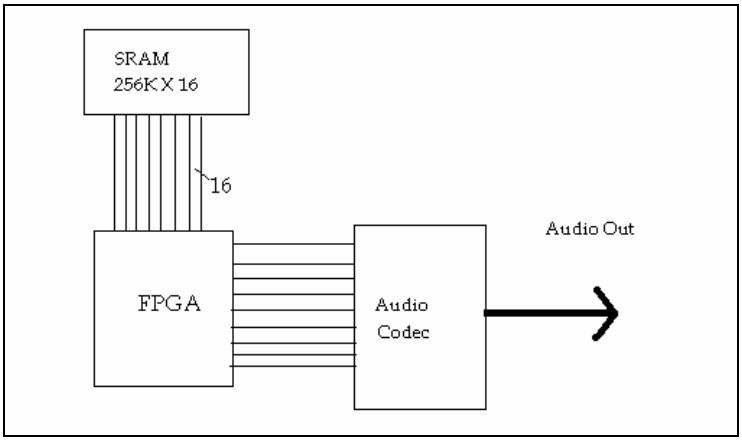



Figure 3: Block Diagram of XESS system

**Vocal Tract Model**

Our attempts to create a speech synthesizer include three attempts at a fitting vocal tract model. In generation one a simple concatenative speech system would be used to pull pre-recorded phonemes from memory and play them in a way such that full words could be generated. Due to limitation of memory on the board we could not fit enough recorded data onto the board to make a useful model.

In the second generation a Klatt synthesizer hardwired in VHDL would represent our vocal tract. Figure 4 shows a block diagram implementation of the

filter system to be created. This model though initially a viable option for hardware synthesis also

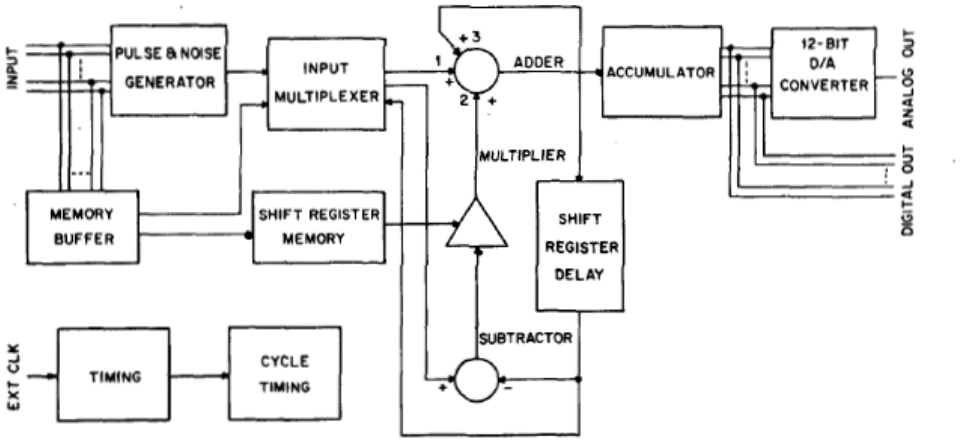



Figure 4

In the final project we decided to implement the synthesizer in C code. Given the complex signal processing nature of speech synthesizers, we realized that it would be best to use code already written as a basis for our project. After consulting with professors here at Columbia University and professors we knew at Carnegie Mellon University, we decided to use an implementation of the Klatt formant synthesizer found at (1).

This program was excellent in that its speech creation algorithm was very similar to ones we found in our research. It was also much smaller than other programs we had found. To actually implement it in our model, we still needed to gut it and only keep functions and code that were completely necessary. We had a working model that generated speech in the workstation, ready for the board.

The text-to-parameter conversion was not implemented in the above c file and it took some more research to find information about that. This information, actually, seems to be the more private and confidential of the two parts of our model. Eventually, using the models we had looked at, the Klatt c code, and the Klatt implementation found in (11), we were able to generate the parameters for the main speech synthesizer.

**SRAM**

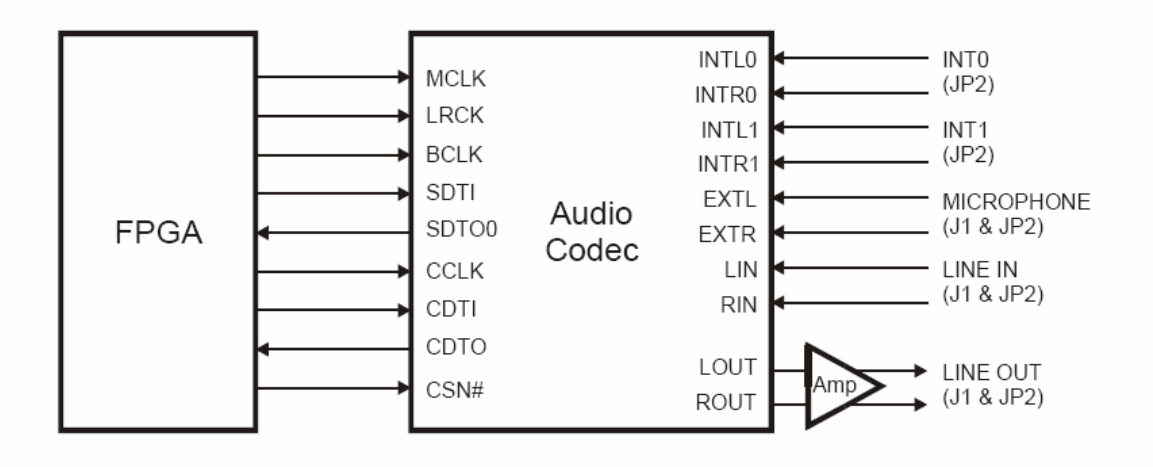
To fit our program code and project files onto the board we required the use of the SRAM. This turned out to be a complex task including the execution of instructions from the SRAM, bridging the architecture of the OPB to the

architecture of the SRAM, and the allocation of specific code segments to memory.

We based our SRAM peripheral on the one we wrote in Lab 6 of this year. Because we had used this controller in Lab 6, we were sure that it worked and had already tested its functionality.

The key to enabling this peripheral in our project was cracking the link script that would put part of code onto this SRAM and put the rest on the BRAM. Although we were able to get parts of this working, this inevitably was the part of the project that we could not complete fully and this is why our final project does not completely work.

**AKM AK4565 Audio Codec**




The audio codec was a challenging component to the project. Initially, the challenge was trying to figure out what files were needed, what pins to include – in other words the technical skeleton of the codec.

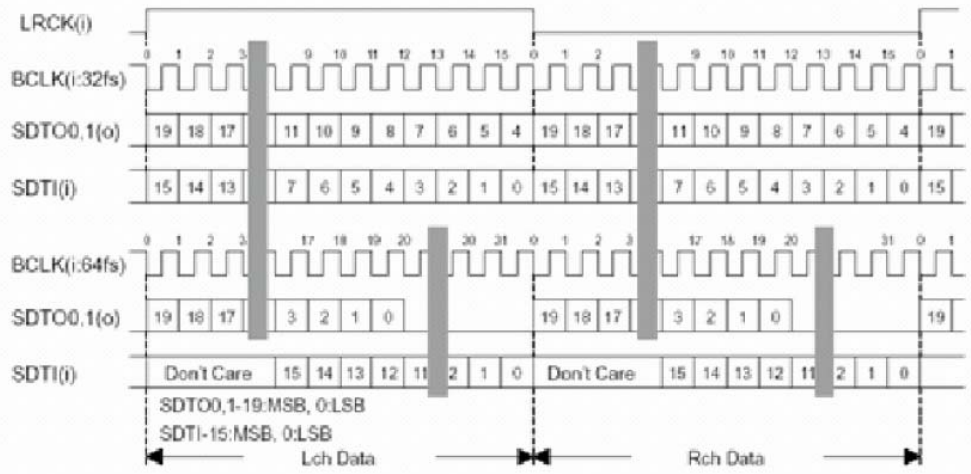



Figure 9. Audio Data Timing (No.0)

We looked at the timing diagram in order to determine exactly how to implement the codec. We looked at the code of successful past projects that also used the audio codec. This was very helpful since we could just modify their code to our own needs.

## IV. Miscellaneous

**Debugging**
Writing the entire project at once and then trying to debug for errors is pretty useless, so our debugging started early on in the project. After we had written the audio files, we originally wrote a small test script that would just test the audio output by sending out random tones. That file compiled, downloaded, and ran, so we knew the audio configuration was correct.

After that, however, the major problem we first encountered had to do with the sheer size of the program code (main.c, namely). Because we decided to implement the vocal tract model in c as opposed to in vhdl, our compiled system.elf program was too large to fit on the BRAM.

With complete audio and SRAM peripherals our program code was downloaded to the board and compiled with no errors, but it did not output sound at all. In this first stage of debugging we chose to first look at the audio peripheral.

To make sure our audio codec was working we created a new version of the project which did not include any of the speech synthesis program code and instead contained only a program that output text to the minicom window and a single tone to the audio CODEC. After numerous iterations through interrupt routine, driver, and address allocation code we discovered the lack of output was resulting from a typo in the memory address of the audio codec.

Once corrected, the tone output from the test file and our audio codec worked perfectly. Unfortunately once replaced in the real project files, the output still did not work. We figured then that there the problem might have had something to do with the way we were allocating our code to the memory. Looking into the SRAM peripheral, we worked more with the linker file (mylinkscript). The rest of our work concentrated on getting the code to properly download into the BRAM and SRAM.

**Who did what**
All group members basically worked collaboratively on the project. Each member had loosely assigned parts to work on. It was beneficial to the sanity of the group to trade off assigned sections. Not only did it increase learning amongst individual group members, but also it created an overall higher efficiency.

To begin with research had to be done in order to determine what kind of model we wanted to use given the memory restrictions of the FPGA. We chose the vocal tract model because it produces waveforms instead of having to store .wav files. The model reduces the amount of memory we needed to use. All group members actively researched different models in order to determine an optimal model.

**Aisha**
I worked on the audio codec with Akshay. The initial challenge was learning the technical skeleton of the audio codec. The XESS XSB-300E user's manual with the audio codec and ak4565 manuals were utilized. Projects from the previous year that also implemented the audio codec were helpful. Additionally, the labs that we did over the course of the semester helped with learning the fundamental communication between the fpga and its peripherals. In particular, lab 6 was helpful for the audio codec and for memory allocation. The lab neatly organized the files that we needed.

I learned a lot about the VHDL language. It was important to realize that the language isn't a 'coding' language. It's a language that codes FOR hardware. At times, it's easy to forget this and insert C language syntax. It was also interesting to parallel the C language with VHDL. For example, a loop in VHDL would be a finite state machine.

**Akshay**
I focused a lot on the vocal tract model we implemented in c. I originally wrote a vocal tract model using the hardware block diagrams above in vhdl, but we decided to settle on a program in software because the models we found were either incomplete or poorly documented. I also did much of the debugging that had to do with the audio codec and the memory problems.

**Girish**
I did a lot of research in the beginning in order to determine what kind of model we really wanted. It was important to determine what model we wanted to use given the limitations of our hardware. I also wrote the code to convert text to phonemes and subsequently the speech parameters.

**Lessons learned**
Probably the most important lesson that we learnt was to choose a project that is feasible. There's only a certain amount of time that we were alloted to finish this project. So, we wanted to choose a project that was interesting to each group member and also realistic given our time frame.

Every project is challenging because of the many new things to learn in a short period of time. It's easy to be ambitious about a project and later realize that things aren't always as easy as they may seem. Our project was definitely doable, but there were many things we didn't think about when proposing the topic. Some unforeseen problems arose during the planning portion of the project. Additionally, we weren't accustomed to CS-type projects. And so the scope of the project wasn't completely clear to us since we were all EE majors.

**Advice to future projects**

The 'make clean' command can really make your life difficult. At times, our programs, in lab and for the project, were correct but we just didn't 'make clean'! It certainly unnecessarily prolonged the amount of time we debugged and at times made us change parts of our program that were correct.

Small typos can really get annoying. And maybe they can't really be avoided. We spent 2 days trying to debug our audio codec, just to discover that the memory assignment didn't match in our VHDL and C parts of the project by just ONE memory address. Perhaps it'd be useful to try to be as consistent as possible. Additionally, sometimes walking away from the program can bring light to small errors.

It's a good idea to start your project as soon as your group settles on a topic. Starting early is key to the success of the project. Our group did start early however we had to spend a lot of time researching before we could hardcode.

# V. References


1. Klatt, D. (1980). Software for a cascade / parallel formant synthesizer. *Journal of the Acoustical Society of America 67(3).* 971-995

2. S. Parthasarathy and C. H. Coker. "Phoneme-level parameterization of speech using an articulatory model" <http://ieeexplore.ieee.org/iel5/132/3385/00115672.pdf?isnumber=&arnumber=1156>.

3. C.H. Coker, N. Umeda, and C.P. Browman. "Automatic Synthesis from Ordinary English Text" <http://ieeexplore.ieee.org/iel6/8337/26098/01162458.pdf?isnumber=&arnumber=1162458>.

4. "A Hardware Realization of a Digital Formant Speech Synthesizer" <http://ieeexplore.ieee.org/iel5/8159/23758/01090781.pdf?isnumber=&arnumber=109781>.

5. "A Polynomial Vocal Tract Model for Speech Synthesis" <http://ieeexplore.ieee.org/iel6/8370/26352/01171904.pdf?isnumber=&arnumber=1171904>.

6. Michael H. O'Malley. "Text-To-Speech Conversión Technology" <http://ieeexplore.ieee.org/iel1/2/2059/00056867.pdf?isnumber=&arnumber=56867>.

7. Robert Vich, Jiff Pfibil and Zdenek Smkkal. "New Cepstral Zero-Pole Vocal Tract Models for TTS Synthesis" <http://ieeexplore.ieee.org/iel5/7466/20308/00938161.pdf>.

8. Vesa Vulimiiki, Matti Karjalainen, and Titno Kuisma. "Articulatory Speech Synthesis Based on Fractional Delay Waveguide Filters" <http://ieeexplore.ieee.org/iel2/3104/8833/00389226.pdf?isnumber=&arnumber=389226>.

9. Colin C. Goodyear and Dongbing Wei. "Articulatory Copy Synthesis Using a Nine-Parameter Vocal Tract Model" <http://ieeexplore.ieee.org/iel3/3856/11264/00541113.pdf?isnumber=&arnumber=541113>.

10. Dan Ellis, Columbia University. "Sinewave Speech Analysis/Síntesis in Matlab" <http://www.ee.columbia.edu/~dpwe/resources/matlab/sws/>.

11. Sean McLennan, Indiana University. "Klatt Synthesizer in Simulink" <http://www.shaav.com/professional/linguistics/klatt.pdf>.

# VI. Source Code

## Code:

main.c
utt.c
isr.c
main.h
parwave.h
proto.h
opb_bram.vhd
opb_bram_v2_1_0.mpd
opb_bram_v2_1_0.pao
opb_xsb300_ak4565.vhd
audio_ak4565.vhd
opb_xsb300_v2_1_0.mpd
opb_xsb300_v2_1_0.pao
Makefile
mylinkscript
system.mhs
system.mss
system.ucf

**main.c**

```
#include "xparameters.h"
#include "xbasic_types.h"
#include "xio.h"
#include "xintc_l.h"
#include "math.h"
#include "proto.h"
#include "parwave.h"
#include "main.h"

// defined in isr.c
extern void audio_handler(void *callback);
extern Xuint32 audio_dac_buf[1024];
extern int audio_dac_bcnt_w, audio_dac_bcnt_r;
extern Xuint32 audiodatain;

// audio output functions
void sound_init();
void sound_out(Xuint32 word);

// defined in utt.c
extern int param[ROWS][COLS];
extern void uttparam();
```

```
/* functions for speech parameter-wave generation */
static void flutter PROTO((klatt_global_ptr,klatt_frame_ptr));
static float sampled_source PROTO((klatt_global_ptr));
static float impulsive_source PROTO((klatt_global_ptr));
static float natural_source PROTO((klatt_global_ptr));
static void pitch_synch_par_reset
PROTO((klatt_global_ptr,klatt_frame_ptr));
static float gen_noise PROTO((float,klatt_global_ptr));
static float DBtoLIN PROTO((long));
static void frame_init PROTO((klatt_global_ptr,klatt_frame_ptr));
static float resonator PROTO((resonator_ptr, float));
static float antiresonator PROTO((resonator_ptr, float));
static void setabc PROTO((long,long,resonator_ptr,klatt_global_ptr));
static void setzeroabc
PROTO((long,long,resonator_ptr,klatt_global_ptr));


int main()
{
  int i, j;
  char k, m;
  char c;
  int result;
  flag done_flag;
  long value;
  int *iwave;
  int isam;
  int icount;
  int par_count;
  int nmspf_def;
  klatt_global_ptr globals;
  klatt_frame_ptr frame;
  long *frame_ptr;
  unsigned char high_byte;
  unsigned char low_byte;
  flag raw_flag;
  flag raw_type;
  int loop;

  int addr = XPAR_AK4565_BASEADDR;
  Xuint32 word;

  static int natural_samples[NUMBER_OF_SAMPLES]=
  {
    -310,-400,530,356,224,89,23,-10,-58,-16,461,599,536,701,770,
    605,497,461,560,404,110,224,131,104,-97,155,278,-154,-1165,
    -598,737,125,-592,41,11,-247,-10,65,92,80,-304,71,167,-1,122,
    233,161,-43,278,479,485,407,266,650,134,80,236,68,260,269,179,
    53,140,275,293,296,104,257,152,311,182,263,245,125,314,140,44,
    203,230,-235,-286,23,107,92,-91,38,464,443,176,98,-784,-2449,
    -1891,-1045,-1600,-1462,-1384,-1261,-949,-730
  };

  globals->quiet_flag = TRUE;
  globals->synthesis_model = ALL_PARALLEL;
  globals->samrate = 16000;
  globals->glsource = NATURAL;
```

```
globals->natural_samples = natural_samples;
globals->num_samples = NUMBER_OF_SAMPLES;
globals->sample_factor = (float) SAMPLE_FACTOR;
nmspf_def = 5;
globals->nfcascade = 0;
globals->outsl = 0;
globals->f0_flutter = 0;
raw_flag = TRUE;
raw_type = 2;

globals->nspfr = (globals->samrate * nmspf_def) / 1000;

microblaze_enable_icache();

setup_interrupts();

sound_init();

icount=0;
done_flag = FALSE;
parwave_init(globals);
frame_ptr = (long*) frame;

while(done_flag == FALSE)
{
  for (i = 0; i < 359; i++)
  {
  for (par_count = 0; par_count < NPAR; ++par_count)
  {
    value = 0;

    value = param[i][par_count];

    frame_ptr[par_count] = value;
  }

  print("Hello World\r\n");

  if(i == 358)
  {
    done_flag = TRUE;
  }
  else
  {
    parwave(globals,frame,iwave);

    for (isam = 0; isam < globals->nspfr; ++ isam)
    {
    low_byte = iwave[isam] & 0xff;
    high_byte = iwave[isam] >> 8;
    word = high_byte;
      word = (word << 8) | low_byte;
    sound_out(word);
    }
    icount++;
  }
  }
```

```
  }

  return 0;
}

void setup_interrupts()
{
  XIntc_mMasterDisable(XPAR_INTC_SINGLE_BASEADDR);
  XIntc_mEnableIntr(XPAR_INTC_SINGLE_BASEADDR,0);
  XIntc_mAckIntr(XPAR_INTC_SINGLE_BASEADDR, 0xffffffff);


XIntc_InterruptVectorTable[XPAR_INTC_AK4565_INTERRUPT_INTR].Handler=aud
io_handler;
  audio_dac_bcnt_w=audio_dac_bcnt_r=0;

  microblaze_enable_interrupts();
  XIntc_mMasterEnable(XPAR_INTC_SINGLE_BASEADDR);
  XIntc_mEnableIntr(XPAR_INTC_SINGLE_BASEADDR,
XPAR_AK4565_INTERRUPT_MASK);
}

void sound_init()
{
  int i;

  for (i=0;i<1024;i++)
  {
    audio_dac_buf[i]=0;
  }

}


void sound_out(Xuint32 word)
{
  int i;

  while(audio_dac_bcnt_w - audio_dac_bcnt_r > 512);

  for (i=0; i<512; i++)
  {
  audio_dac_buf[audio_dac_bcnt_w++ & 0x3FF] =  word;
  }

}


/*
function RESONATOR
*/

static float resonator(r, input)

resonator_ptr r;
float input;
```

```
{
 float x;

 x = (float) (r->a * input + r->b * r->p1 + r->c * r->p2);
 r->p2 = r->p1;
 r->p1 = x;

 return x;
}



/*
function ANTIRESONATOR
*/

static float antiresonator(r, input)

resonator_ptr r;
float input;

{
 register float x = r->a * input + r->b * r->p1 + r->c * r->p2;
 r->p2 = r->p1;
 r->p1 = input;
 return x;
}



/*
function FLUTTER

This function adds F0 flutter, as specified in:

"Analysis, synthesis and perception of voice quality variations among
female and male talkers" D.H. Klatt and L.C. Klatt JASA 87(2) February
1990.

Flutter is added by applying a quasi-random element constructed from
three
slowly varying sine waves.
*/


static void flutter(globals,frame)

klatt_global_ptr globals;
klatt_frame_ptr frame;
{
  static int time_count;
  double delta_f0;
  double fla,flb,flc,fld,fle;

  fla = (double) globals->f0_flutter / 50;
  flb = (double) globals->original_f0 / 100;
  flc = sin(2*PI*12.7*time_count);
```

```
  fld = sin(2*PI*7.1*time_count);
  fle = sin(2*PI*4.7*time_count);
  delta_f0 =  fla * flb * (flc + fld + fle) * 10;
  frame->F0hz10 = frame->F0hz10 + (long) delta_f0;
  time_count++;
}


/*
function SAMPLED_SOURCE

Allows the use of a glottal excitation waveform sampled from a real
voice.
*/

static float sampled_source(globals)

klatt_global_ptr globals;

{
  int itemp;
  float ftemp;
  float result;
  float diff_value;
  int current_value;
  int next_value;
  float temp_diff;

  if(globals->T0!=0)
  {
    ftemp = (float) globals->nper;
    ftemp = ftemp / globals->T0;
    ftemp = ftemp * globals->num_samples;
    itemp = (int) ftemp;

    temp_diff = ftemp - (float) itemp;

    current_value = globals->natural_samples[itemp];
    next_value = globals->natural_samples[itemp+1];

    diff_value = (float) next_value - (float) current_value;
    diff_value = diff_value * temp_diff;

    result = globals->natural_samples[itemp] + diff_value;
    result = result * globals->sample_factor;
  }
  else
  {
    result = 0;
  }
  return(result);
}




/*
```

```
function PARWAVE

Converts synthesis parameters to a waveform.
*/


void parwave(globals,frame,output)

klatt_global_ptr globals;
klatt_frame_ptr frame;
int *output;

{
  float temp;
  float outbypas;
  float out;
  long n4;
  float frics;
  float glotout;
  float aspiration;
  float casc_next_in;
  float par_glotout;
  static float noise;
  static float voice;
  static float vlast;
  static float glotlast;
  static float sourc;

  frame_init(globals,frame);  /* get parameters for next frame of
speech */


  flutter(globals,frame);  /* add f0 flutter */


  /* MAIN LOOP, for each output sample of current frame: */

  for (globals->ns=0;globals->ns<globals->nspfr;globals->ns++)
  {

    /* Get low-passed random number for aspiration and frication noise
*/

    noise = gen_noise(noise,globals);

    /*
      Amplitude modulate noise (reduce noise amplitude during
      second half of glottal period) if voicing simultaneously present.
    */

    if (globals->nper > globals->nmod)
    {
      noise *= (float) 0.5;
    }

    /* Compute frication noise */
```

```
    frics = globals->amp_frica * noise;

    /*
      Compute voicing waveform. Run glottal source simulation at 4
      times normal sample rate to minimize quantization noise in
      period of female voice.
    */

    for (n4=0; n4<4; n4++)
    {
      switch(globals->glsource)
      {
      case IMPULSIVE:
      voice = impulsive_source(globals);
      break;
      case NATURAL:
      voice = natural_source(globals);
      break;
      case SAMPLED:
      voice = sampled_source(globals);
      break;
      }

      /* Reset period when counter 'nper' reaches T0 */

      if (globals->nper >= globals->T0)
      {
      globals->nper = 0;
      pitch_synch_par_reset(globals,frame);
      }

      /*
      Low-pass filter voicing waveform before downsampling from
4*samrate
      to samrate samples/sec.  Resonator f=.09*samrate, bw=.06*samrate
      */

      voice = resonator(&(globals->rlp),voice);

      /* Increment counter that keeps track of 4*samrate samples per
sec */

      globals->nper++;
    }

    /*
      Tilt spectrum of voicing source down by soft low-pass filtering,
amount
      of tilt determined by TLTdb
    */


    voice = (voice * globals->onemd) + (vlast * globals->decay);
    vlast = voice;


    /*
```

```
  Add breathiness during glottal open phase. Amount of breathiness
  determined by parameter Aturb Use nrand rather than noise because
  noise is low-passed.
*/


if (globals->nper < globals->nopen)
{
  voice += globals->amp_breth * globals->nrand;
}


/* Set amplitude of voicing */

glotout = globals->amp_voice * voice;
par_glotout = globals->par_amp_voice * voice;


/* Compute aspiration amplitude and add to voicing source */


aspiration = globals->amp_aspir * noise;
glotout += aspiration;


par_glotout += aspiration;


/*
  Cascade vocal tract, excited by laryngeal sources.
  Nasal antiresonator, then formants FNP, F5, F4, F3, F2, F1
*/

if(globals->synthesis_model != ALL_PARALLEL)
{
  casc_next_in = antiresonator(&(globals->rnz),glotout);
  casc_next_in = resonator(&(globals->rnpc),casc_next_in);

  /* Do not use unless sample rate >= 16000 */

  if (globals->nfcascade >= 8)
  {
  casc_next_in= resonator(&(globals->r8c),casc_next_in);
  }

  /* Do not use unless sample rate >= 16000 */

  if (globals->nfcascade >= 7)
  {
  casc_next_in = resonator(&(globals->r7c),casc_next_in);
  }

  /* Do not use unless long vocal tract or sample rate increased */

  if (globals->nfcascade >= 6)
  {
  casc_next_in = resonator(&(globals->r6c),casc_next_in);
```

```
  }

  if (globals->nfcascade >= 5)
  {
  casc_next_in = resonator(&(globals->r5c),casc_next_in);
  }

  if (globals->nfcascade >= 4)
  {
  casc_next_in = resonator(&(globals->r4c),casc_next_in);
  }

  if (globals->nfcascade >= 3)
  {
  casc_next_in = resonator(&(globals->r3c),casc_next_in);
  }

  if (globals->nfcascade >= 2)
  {
  casc_next_in = resonator(&(globals->r2c),casc_next_in);
  }

  if (globals->nfcascade >= 1)
  {
  out = resonator(&(globals->r1c),casc_next_in);
  }
}
else
{
  /* we are not using the cascade tract, set out to zero */
  out = 0;
}


/* Excite parallel F1 and FNP by voicing waveform */

sourc = par_glotout;        /* Source is voicing plus aspiration */



/*
  Standard parallel vocal tract Formants F6,F5,F4,F3,F2,
  outputs added with alternating sign. Sound sourc for other
  parallel resonators is frication plus first difference of
  voicing waveform.
*/

out += resonator(&(globals->r1p),sourc);
out += resonator(&(globals->rnpp),sourc);

sourc = frics + par_glotout - glotlast;
glotlast = par_glotout;

out = resonator(&(globals->r6p),sourc) - out;
out = resonator(&(globals->r5p),sourc) - out;
out = resonator(&(globals->r4p),sourc) - out;
out = resonator(&(globals->r3p),sourc) - out;
```

```
    out = resonator(&(globals->r2p),sourc) - out;

    outbypas = globals->amp_bypas * sourc;
    out = outbypas - out;


    if (globals->outsl != 0)
    {
      switch(globals->outsl)
      {
      case 1:
      out = voice;
      break;
      case 2:
      out = aspiration;
      break;
      case 3:
      out = frics;
      break;
      case 4:
      out = glotout;
      break;
      case 5:
      out = par_glotout;
      break;
      case 6:
      out = outbypas;
      break;
      case 7:
      out = sourc;
      break;
      }
    }

    out = resonator(&(globals->rout),out);

    temp = out * globals->amp_gain0;  /* Convert back to integer */


    if (temp < -32768.0)
    {
      temp = -32768.0;
    }

    if (temp >    32767.0)
    {
      temp =  32767.0;
    }

    *output++ = (int) temp;
  }
}



/*
```

```
function PARWAVE_INIT

Initialises all parameters used in parwave, sets resonator internal
memory
to zero.
*/

void parwave_init(globals)

klatt_global_ptr globals;

{
  globals->FLPhz = (950 * globals->samrate) / 10000;
  globals->BLPhz = (630 * globals->samrate) / 10000;
  globals->minus_pi_t = -PI / globals->samrate;
  globals->two_pi_t = -2.0 * globals->minus_pi_t;
  setabc(globals->FLPhz,globals->BLPhz,&(globals->rlp),globals);
  globals->nper = 0;
  globals->T0 = 0;
  globals->nopen = 0;
  globals->nmod = 0;

  globals->rnpp.p1=0;
  globals->r1p.p1=0;
  globals->r2p.p1=0;
  globals->r3p.p1=0;
  globals->r4p.p1=0;
  globals->r5p.p1=0;
  globals->r6p.p1=0;
  globals->r1c.p1=0;
  globals->r2c.p1=0;
  globals->r3c.p1=0;
  globals->r4c.p1=0;
  globals->r5c.p1=0;
  globals->r6c.p1=0;
  globals->r7c.p1=0;
  globals->r8c.p1=0;
  globals->rnpc.p1=0;
  globals->rnz.p1=0;
  globals->rgl.p1=0;
  globals->rlp.p1=0;
  globals->rout.p1=0;

  globals->rnpp.p2=0;
  globals->r1p.p2=0;
  globals->r2p.p2=0;
  globals->r3p.p2=0;
  globals->r4p.p2=0;
  globals->r5p.p2=0;
  globals->r6p.p2=0;
  globals->r1c.p2=0;
  globals->r2c.p2=0;
  globals->r3c.p2=0;
  globals->r4c.p2=0;
  globals->r5c.p2=0;
  globals->r6c.p2=0;
  globals->r7c.p2=0;
```

```
  globals->r8c.p2=0;
  globals->rnpc.p2=0;
  globals->rnz.p2=0;
  globals->rgl.p2=0;
  globals->rlp.p2=0;
  globals->rout.p2=0;
}


/*
function FRAME_INIT

Use parameters from the input frame to set up resonator coefficients.
*/

static void frame_init(globals,frame)

klatt_global_ptr globals;
klatt_frame_ptr frame;

{
  float amp_parF1;
  float amp_parFNP;
  float amp_parF2;
  float amp_parF3;
  float amp_parF4;
  float amp_parF5;
  float amp_parF6;

  globals->original_f0 = frame->F0hz10 / 10;

  frame->AVdb  = frame->AVdb - 7;
  if (frame->AVdb < 0)
  {
    frame->AVdb = 0;
  }

  globals->amp_aspir = DBtoLIN(frame->ASP) * 0.05;
  globals->amp_frica = DBtoLIN(frame->AF) * 0.25;
  globals->par_amp_voice = DBtoLIN(frame->AVpdb);
  amp_parF1 = DBtoLIN(frame->A1) * 0.4;
  amp_parF2 = DBtoLIN(frame->A2) * 0.15;
  amp_parF3 = DBtoLIN(frame->A3) * 0.06;
  amp_parF4 = DBtoLIN(frame->A4) * 0.04;
  amp_parF5 = DBtoLIN(frame->A5) * 0.022;
  amp_parF6 = DBtoLIN(frame->A6) * 0.03;
  amp_parFNP = DBtoLIN(frame->ANP) * 0.6;
  globals->amp_bypas = DBtoLIN(frame->AB) * 0.05;
  frame->Gain0 = frame->Gain0 - 3;
  if (frame->Gain0 <= 0)
  {
    frame->Gain0 = 57;
  }
  globals->amp_gain0 = DBtoLIN(frame->Gain0);

  /* Set coefficients of variable cascade resonators */
```

```
  if (globals->nfcascade >= 8)
  {
    setabc(7500,600,&(globals->r8c),globals);
  }
  if (globals->nfcascade >= 7)
  {
    setabc(6500,500,&(globals->r7c),globals);
  }
  if (globals->nfcascade >= 6)
  {
    setabc(frame->F6hz,frame->B6hz,&(globals->r6c),globals);
  }
  if (globals->nfcascade >= 5)
  {
    setabc(frame->F5hz,frame->B5hz,&(globals->r5c),globals);
  }
  setabc(frame->F4hz,frame->B4hz,&(globals->r4c),globals);
  setabc(frame->F3hz,frame->B3hz,&(globals->r3c),globals);
  setabc(frame->F2hz,frame->B2hz,&(globals->r2c),globals);
  setabc(frame->F1hz,frame->B1hz,&(globals->r1c),globals);

  /* Set coeficients of nasal resonator and zero antiresonator */

  setabc(frame->FNPhz,frame->BNPhz,&(globals->rnpc),globals);
  setzeroabc(frame->FNZhz,frame->BNZhz,&(globals->rnz),globals);

  /* Set coefficients of parallel resonators, and amplitude of outputs
*/

  setabc(frame->F1hz,frame->B1phz,&(globals->r1p),globals);
  globals->r1p.a *= amp_parF1;
  setabc(frame->FNPhz,frame->BNPhz,&(globals->rnpp),globals);
  globals->rnpp.a *= amp_parFNP;
  setabc(frame->F2hz,frame->B2phz,&(globals->r2p),globals);
  globals->r2p.a *= amp_parF2;
  setabc(frame->F3hz,frame->B3phz,&(globals->r3p),globals);
  globals->r3p.a *= amp_parF3;
  setabc(frame->F4hz,frame->B4phz,&(globals->r4p),globals);
  globals->r4p.a *= amp_parF4;
  setabc(frame->F5hz,frame->B5phz,&(globals->r5p),globals);
  globals->r5p.a *= amp_parF5;
  setabc(frame->F6hz,frame->B6phz,&(globals->r6p),globals);
  globals->r6p.a *= amp_parF6;

  /* output low-pass filter */

  setabc((long)0.0,(long)(globals->samrate/2),&(globals-
>rout),globals);
}



/*
function IMPULSIVE_SOURCE
*/
```

```
static float impulsive_source(globals)

klatt_global_ptr globals;

{
  static float doublet[] = {0.0,13000000.0,-13000000.0};
  static float vwave;

  if (globals->nper < 3)
  {
    vwave = doublet[globals->nper];
  }
  else
  {
    vwave = 0.0;
  }

  return(resonator(&(globals->rgl),vwave));
}



/*
function NATURAL_SOURCE

Vwave is the differentiated glottal flow waveform, there is a weak
spectral zero around 800 Hz, magic constants a,b reset pitch
synchronously.
*/

static float natural_source(globals)

klatt_global_ptr globals;

{
  float lgtemp;
  static float vwave;

  if (globals->nper < globals->nopen)
  {
    globals->pulse_shape_a -= globals->pulse_shape_b;
    vwave += globals->pulse_shape_a;
    lgtemp=vwave * 0.028;

    return(lgtemp);
  }
  else
  {
    vwave = 0.0;
    return(0.0);
  }
}

/*
function PITCH_SYNC_PAR_RESET
*/
```

```
static void pitch_synch_par_reset(globals,frame)

klatt_global_ptr globals;
klatt_frame_ptr frame;

{
  long temp;
  float temp1;
  static long skew;
  static short B0[224] =
  {
    1200,1142,1088,1038, 991, 948, 907, 869, 833, 799, 768, 738, 710,
683, 658,
    634, 612, 590, 570, 551, 533, 515, 499, 483, 468, 454, 440, 427,
415, 403,
    391, 380, 370, 360, 350, 341, 332, 323, 315, 307, 300, 292, 285,
278, 272,
    265, 259, 253, 247, 242, 237, 231, 226, 221, 217, 212, 208, 204,
199, 195,
    192, 188, 184, 180, 177, 174, 170, 167, 164, 161, 158, 155, 153,
150, 147,
    145, 142, 140, 137, 135, 133, 131, 128, 126, 124, 122, 120, 119,
117, 115,
    113,111, 110, 108, 106, 105, 103, 102, 100, 99, 97, 96, 95, 93, 92,
91, 90,
    88, 87, 86, 85, 84, 83, 82, 80, 79, 78, 77, 76, 75, 75, 74, 73, 72,
71,
    70, 69, 68, 68, 67, 66, 65, 64, 64, 63, 62, 61, 61, 60, 59, 59, 58,
57,
    57, 56, 56, 55, 55, 54, 54, 53, 53, 52, 52, 51, 51, 50, 50, 49, 49,
48, 48,
    47, 47, 46, 46, 45, 45, 44, 44, 43, 43, 42, 42, 41, 41, 41, 41, 40,
40,
    39, 39, 38, 38, 38, 38, 37, 37, 36, 36, 36, 36, 35, 35, 35, 35, 34,
34,33,
    33, 33, 33, 32, 32, 32, 32, 31, 31, 31, 31, 30, 30, 30, 30, 29, 29,
29, 29,
    28, 28, 28, 28, 27, 27
  };

  if (frame->F0hz10 > 0)
  {
    /* T0 is 4* the number of samples in one pitch period */

    globals->T0 = (40 * globals->samrate) / frame->F0hz10;


    globals->amp_voice = DBtoLIN(frame->AVdb);

    /* Duration of period before amplitude modulation */

    globals->nmod = globals->T0;
    if (frame->AVdb > 0)
    {
      globals->nmod >>= 1;
    }
```

```
    /* Breathiness of voicing waveform */

    globals->amp_breth = DBtoLIN(frame->Aturb) * 0.1;

    /* Set open phase of glottal period where  40 <= open phase <= 263
*/

    globals->nopen = 4 * frame->Kopen;

    if ((globals->glsource == IMPULSIVE) && (globals->nopen > 263))
    {
      globals->nopen = 263;
    }

    if (globals->nopen >= (globals->T0-1))
    {
      globals->nopen = globals->T0 - 2;
      if(globals->quiet_flag == FALSE)
      {
      print("Warning: glottal open period cannot exceed T0,
truncated\n");
      }
    }

    if (globals->nopen < 40)
    {
      /* F0 max = 1000 Hz */
      globals->nopen = 40;
      if(globals->quiet_flag == FALSE)
      {
      print("Warning: minimum glottal open period is 10 samples.\n");
      print("truncated, nopen = ");
      putnum((int) globals->nopen);
      print("\r\n");
      }
    }


    /* Reset a & b, which determine shape of "natural" glottal waveform
*/

    globals->pulse_shape_b = B0[globals->nopen-40];
    globals->pulse_shape_a = (globals->pulse_shape_b * globals->nopen)
* 0.333;

    /* Reset width of "impulsive" glottal pulse */

    temp = globals->samrate / globals->nopen;

    setabc((long)0,temp,&(globals->rgl),globals);

    /* Make gain at F1 about constant */

    temp1 = globals->nopen *.00833;
    globals->rgl.a *= temp1 * temp1;

    /*
```

```
    Truncate skewness so as not to exceed duration of closed phase
    of glottal period.
  */


  temp = globals->T0 - globals->nopen;
  if (frame->Kskew > temp)
  {
    if(globals->quiet_flag == FALSE)
    {
    print("Kskew duration = ");
    putnum((int)frame->Kskew);
    print("> glottal closed period= ");
    putnum((int)(globals->T0 - globals->nopen));
    print("truncate\r\n");
    }
    frame->Kskew = temp;
  }
  if (skew >= 0)
  {
    skew = frame->Kskew;
  }
  else
  {
    skew = - frame->Kskew;
  }

  /* Add skewness to closed portion of voicing period */


  globals->T0 = globals->T0 + skew;
  skew = - skew;

}

else
{
  globals->T0 = 4;                      /* Default for f0 undefined */
  globals->amp_voice = 0.0;
  globals->nmod = globals->T0;
  globals->amp_breth = 0.0;
  globals->pulse_shape_a = 0.0;
  globals->pulse_shape_b = 0.0;
}

/* Reset these pars pitch synchronously or at update rate if f0=0 */

if ((globals->T0 != 4) || (globals->ns == 0))
{
  /* Set one-pole low-pass filter that tilts glottal source */

  globals->decay = (0.033 * frame->TLTdb);

  if (globals->decay > 0.0)
  {
    globals->onemd = 1.0 - globals->decay;
  }
```

```
    else
    {
      globals->onemd = 1.0;
    }
  }
}



/*
function SETABC

Convert formant freqencies and bandwidth into resonator difference
equation constants.
*/


static void setabc(f, bw, rp, globals)

long int f;          /* Frequency of resonator in Hz         */
long int bw;        /* Bandwidth of resonator in Hz         */
resonator_ptr rp;
klatt_global_ptr globals;

{
 float r;
 double arg;

/* Let r  =  exp(-pi bw t) */

 arg = globals->minus_pi_t * bw;
 r = exp(arg);

/* Let c  =  -r**2 */

 rp->c = -(r * r);

/* Let b = r * 2*cos(2 pi f t) */

 arg = globals->two_pi_t * f;
 rp->b = r * cos(arg) * 2.0;

/* Let a = 1.0 - b - c */

 rp->a = 1.0 - rp->b - rp->c;
}


/*
function SETZEROABC

Convert formant freqencies and bandwidth into anti-resonator difference
equation constants.
*/

static void setzeroabc(f, bw, rp, globals)
```

```
long int f;          /* Frequency of resonator in Hz         */
long int bw;        /* Bandwidth of resonator in Hz         */
resonator_ptr rp;
klatt_global_ptr globals;

{
 float r;
 double arg;

 f = -f;

 if(f>=0)
 {
   f = -1;
 }

/* First compute ordinary resonator coefficients */
/* Let r  =  exp(-pi bw t) */

 arg = globals->minus_pi_t * bw;
 r = exp(arg);

/* Let c  =  -r**2 */

 rp->c = -(r * r);

/* Let b = r * 2*cos(2 pi f t) */

 arg = globals->two_pi_t * f;
 rp->b = r * cos(arg) * 2.;

/* Let a = 1.0 - b - c */

 rp->a = 1.0 - rp->b - rp->c;

/* Now convert to antiresonator coefficients (a'=1/a, b'=b/a, c'=c/a)
*/

 rp->a = 1.0 / rp->a;
 rp->c *= -rp->a;
 rp->b *= -rp->a;
}


/*
function GEN_NOISE

Random number generator (return a number between -8191 and +8191)
Noise spectrum is tilted down by soft low-pass filter having a pole
near
the origin in the z-plane, i.e. output = input + (0.75 * lastoutput)
*/


static float gen_noise(noise,globals)

float noise;
```

```
klatt_global_ptr globals;

{
  long temp;
  static float nlast;

  temp = (long) getrandom(-8191,8191);
  globals->nrand = (long) temp;

  noise = globals->nrand + (0.75 * nlast);
  nlast = noise;

  return(noise);
}


/*
function DBTOLIN
*/


static float DBtoLIN(dB)

long dB;

{
  float lgtemp;
  static float amptable[88] =
  {
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
6.0, 7.0,
    8.0, 9.0, 10.0, 11.0, 13.0, 14.0, 16.0, 18.0, 20.0, 22.0, 25.0,
28.0, 32.0,
    35.0, 40.0, 45.0, 51.0, 57.0, 64.0, 71.0, 80.0, 90.0, 101.0, 114.0,
128.0,
    142.0, 159.0, 179.0, 202.0, 227.0, 256.0, 284.0, 318.0, 359.0,
405.0,
    455.0, 512.0, 568.0, 638.0, 719.0, 811.0, 911.0, 1024.0, 1137.0,
1276.0,
    1438.0, 1622.0, 1823.0, 2048.0, 2273.0, 2552.0, 2875.0, 3244.0,
3645.0,
    4096.0, 4547.0, 5104.0, 5751.0, 6488.0, 7291.0, 8192.0, 9093.0,
10207.0,
    11502.0, 12976.0, 14582.0, 16384.0, 18350.0, 20644.0, 23429.0,
    26214.0, 29491.0, 32767
  };

  if ((dB < 0) || (dB > 87))
  {
      return(0);
  }

  lgtemp=amptable[dB] * .001;
  return(lgtemp);
}
```

**utt.c**

```
#include "main.h"

int param[ROWS][COLS];

void uttparam();

void uttparam()
{

//f0 av f1 b1 f2 b2 f3 b3 f4 b4 f5 b5 f6 b6 fnz bnz fnp bnp asp kopen
aturb tilt af skew a1 b1p a2 b2p a3 b3p a4 b4p a5 b5p a6 b6p anp ab avp
gain

int utt[58][40]= {
{1000,50,290,0,610,0,2150,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20,
0,0,0,0,0,80,0,60,0,250,0,80,0,80,0,0,0,70},  // /w/
{1000,50,260,0,2070,0,3020,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,0,0,0,52,0,250,0,500,0,250,0,80,0,80,0,0,0,70},  // /y/
{1000,50,310,0,1060,0,1380,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,0,0,0,52,0,100,0,120,0,250,0,80,0,80,0,0,0,70},  // /r/
{1000,50,310,0,1050,0,2880,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,0,0,0,52,0,100,0,280,0,250,0,80,0,80,0,0,0,70},  // /l/
{1000,40,480,0,1270,0,2130,0,3300,0,3750,0,4900,0,0,0,270,100,0,40,0,20
,0,0,0,52,0,200,0,200,0,250,0,80,0,80,0,0,50,70},  // /m/
{1000,40,480,0,1340,0,2470,0,3300,0,3750,0,4900,0,0,0,270,100,0,40,0,20
,0,0,0,52,0,300,0,300,0,250,0,80,0,80,0,0,50,70},  // /n/
{1000,40,480,0,2000,0,2900,0,3300,0,3750,0,4900,0,0,0,270,100,0,40,0,20
,0,0,0,52,0,300,0,300,0,250,0,80,0,80,0,0,50,70},  // /N/
{1000,0,340,0,1100,0,2080,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20,
20,0,0,52,0,120,0,150,0,250,0,80,0,80,0,57,0,70},  // /f/
{1000,47,220,0,1100,0,2080,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,20,0,0,52,0,90,0,120,0,250,0,80,0,80,0,57,47,70},  // /v/
{1000,0,320,0,1290,0,2540,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20,
20,0,0,52,0,90,0,200,0,250,0,80,28,80,0,38,0,70},  // /T/
{1000,47,270,0,1290,0,2540,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,20,0,0,52,0,80,0,170,0,250,0,80,28,80,0,38,47,70},  // /D/
{1000,0,320,0,1390,0,2530,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20,
20,0,0,52,0,80,0,200,0,250,0,80,52,80,0,0,0,70},  // /s/
{1000,47,240,0,1390,0,2530,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,20,0,0,52,0,60,0,180,0,250,0,80,52,80,0,0,47,70},  // /z/
{1000,0,300,0,1840,0,2750,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20,
20,0,0,52,0,100,28,300,24,250,24,80,23,80,0,0,0,70},  // /S/
{1000,47,220,0,1840,0,2750,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,20,0,0,52,0,60,28,280,24,250,24,80,23,80,0,0,47,70},  // /Z/
{1000,0,350,0,1800,0,2820,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20,
10,0,0,52,0,90,22,300,30,250,26,80,26,80,0,0,0,70},  // /c/
{1000,37,260,0,1800,0,2820,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,10,0,0,52,0,80,22,270,30,250,26,80,26,80,0,0,37,70},  // /j/
{1000,20,200,0,1100,0,2150,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,10,0,0,52,0,110,0,130,0,250,0,80,0,80,0,63,20,70},  // /b/
{1000,20,200,0,1600,0,2600,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,10,0,0,52,0,100,23,170,30,250,31,80,30,80,0,0,20,70},  // /d/
{1000,20,200,0,1990,0,2850,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,10,0,0,52,30,150,27,280,22,250,23,80,23,80,0,0,20,70},  // /g/
```

```
{1000,0,400,0,1100,0,2150,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20,
20,0,0,52,0,150,0,220,0,250,0,80,0,80,0,63,0,70},  // /p/
{1000,0,400,0,1600,0,2600,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20,
20,0,0,52,0,120,15,250,23,250,28,80,32,80,0,0,0,70},  // /t/
{1000,0,300,0,1990,0,2850,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20,
20,0,0,52,30,160,26,330,22,250,23,80,23,80,0,0,0,70},  // /k/
{1000,60,310,0,2020,0,2960,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,0,0,0,52,0,200,0,400,0,250,0,80,0,80,0,0,0,70},  // /i/
{1000,60,290,0,2070,0,2960,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,0,0,0,52,0,200,0,400,0,250,0,80,0,80,0,0,0,70},  //
{1000,60,400,0,1800,0,2570,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,0,0,0,52,0,100,0,140,0,250,0,80,0,80,0,0,0,70},  // /I/
{1000,60,470,0,1600,0,2600,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,0,0,0,52,0,100,0,140,0,250,0,80,0,80,0,0,0,70},  //
{1000,60,480,0,1720,0,2520,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,0,0,0,52,0,100,0,200,0,250,0,80,0,80,0,0,0,70},  // /e/
{1000,60,330,0,2020,0,2600,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,0,0,0,52,0,100,0,200,0,250,0,80,0,80,0,0,0,70},  //
{1000,60,530,0,1680,0,2500,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,0,0,0,52,0,90,0,200,0,250,0,80,0,80,0,0,0,70},  // /E/
{1000,60,620,0,1530,0,2530,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,0,0,0,52,0,90,0,200,0,250,0,80,0,80,0,0,0,70},  //
{1000,60,620,0,1660,0,2430,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,0,0,0,52,0,150,0,320,0,250,0,80,0,80,0,0,0,70},  // /A/
{1000,60,650,0,1490,0,2470,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,0,0,0,52,0,100,0,320,0,250,0,80,0,80,0,0,0,70},  //
{1000,60,700,0,1220,0,2600,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,0,0,0,52,0,70,0,160,0,250,0,80,0,80,0,0,0,70},  // /a/
{1000,60,700,0,1220,0,2600,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,0,0,0,52,0,70,0,160,0,250,0,80,0,80,0,0,0,70},  //
{1000,60,600,0,990,0,2570,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20,
0,0,0,52,0,100,0,80,0,250,0,80,0,80,0,0,0,70},  // />/
{1000,60,630,0,1040,0,2600,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,0,0,0,52,0,100,0,80,0,250,0,80,0,80,0,0,0,70},  //
{1000,60,620,0,1220,0,2550,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,0,0,0,52,0,50,0,140,0,250,0,80,0,80,0,0,0,70},  // /^/
{1000,60,620,0,1220,0,2550,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,0,0,0,52,0,50,0,140,0,250,0,80,0,80,0,0,0,70},  //
{1000,60,540,0,1100,0,2300,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,0,0,0,52,0,70,0,70,0,250,0,80,0,80,0,0,0,70},  // /o/
{1000,60,450,0,900,0,2300,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20,
0,0,0,52,0,70,0,70,0,250,0,80,0,80,0,0,0,70},  //
{1000,60,450,0,1100,0,2350,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,0,0,0,52,0,100,0,80,0,250,0,80,0,80,0,0,0,70},  // /U/
{1000,60,500,0,1180,0,2390,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,0,0,0,52,0,100,0,80,0,250,0,80,0,80,0,0,0,70},  //
{1000,60,350,0,1250,0,2200,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,0,0,0,52,0,110,0,140,0,250,0,80,0,80,0,0,0,70},  // /u/
{1000,60,320,0,900,0,2200,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20,
0,0,0,52,0,110,0,140,0,250,0,80,0,80,0,0,0,70},  //
{1000,50,500,0,1400,0,2300,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,0,0,0,52,0,60,0,110,0,250,0,80,0,80,0,0,0,70},  // / * /
{1000,50,500,0,1400,0,2300,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,0,0,0,52,0,60,0,110,0,250,0,80,0,80,0,0,0,70},  //
{1000,60,660,0,1200,0,2550,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,0,0,0,52,0,70,0,200,0,250,0,80,0,80,0,0,0,70},  // /@/
```

```
{1000,60,400,0,1880,0,2500,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,0,0,0,52,0,100,0,200,0,250,0,80,0,80,0,0,0,70},  //
{1000,60,640,0,1230,0,2550,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,0,0,0,52,0,70,0,140,0,250,0,80,0,80,0,0,0,70},  // /&/
{1000,60,420,0,940,0,2350,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20,
0,0,0,52,0,70,0,80,0,250,0,80,0,80,0,0,0,70},  //
{1000,60,550,0,960,0,2400,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20,
0,0,0,52,0,50,0,130,0,250,0,80,0,80,0,0,0,70},  ///!/
{1000,60,360,0,1820,0,2450,0,3300,0,3750,0,4900,0,0,0,250,100,0,40,0,20
,0,0,0,52,0,50,0,160,0,250,0,80,0,80,0,0,0,70} };


/*
IPA                  Representation    ASCII             Example
/H/                        "                           (aspiration)
/'upsidedown e'/           *               *           (neutral vowel)
      --
/a/                        AA              a           fAther
      --
/ae/                       AE              A           fAt
      --
/Ã/                        AH              ^           bUt
      --
/'upsidedow c/             AO              >           lAWn
      --
/'upsidedow c'j/           AU              !           cAUght
      --
/aw/                       AW              &           hOW
      --
/b/                        B               b           Back
      --
/t'integral sign'/         CH              c           CHar
      --
/d/                        D               d           Dime
      --
/D/                        DH              D           eiTHer
      --
/E/                        EH              E           gEt
      --
/e/                        EY              e           gAte
      --
/f/                        F               f           Fault
      --
/g/                        G               g           Goat
      --
/h/                        HH              h           How
/I/                        IH              I           bIt
      --
/i/                        IY              i           bEEt
      --
/d3/                         JH              j           Jar
      --
/k/                        K               k           Coat
      --
/l/                        L               l           Laugh
      --
```

```
/m/                     M           m           suM
      --
/n/                     N           n           suN
      --
/N/                     NX          N           suNG
      --
/o/                     OW          o           lOne
      --
/p/                     P           p           Pack
      --
/R/                     R           r           Rate
      --
/s/                     S           s           Sue
      --
/'Integral sign'/       SH          S           leaSH
      --
/t/                     T           t           Time
      --
/'theta'/               TH          T           eTHer
      --
/U/                     UH          U           fUll
      --
/u/                     UW          u           fOOl
      --
/v/                     V           v           Vault
/w/                     W           w           Wear
/j/                     Y           y           Young
/z/                     Z           z           Zoo
/Z/                     ZH          Z           leiSure
/aj/                     @

*/


//Enter input string using ASCII representation above
char input[20] = "ons ^p>n a tim";        //Length cannot be greater
than 20 for space considerations

// Above text says "Once upon a time"

int length, i, j, row;
int q, q2;

int index = 0;

length = 20;
row =1;

//Initialize parameter array to 0
for (i=0; i<ROWS; i++)
{
  for (j=0; j<COLS; j++)
  {
    param[i][j]=0;
  }
}
```

```
//i = 1;
for (i=0; i<length; i++){
  char c = input[i];
  switch (c)
  {
    case 'w':
      row = 0;
      break;
    case 'y':
      row = 1;
      break;
    case 'r':
      row = 2;
      break;
    case 'l':
      row = 3;
      break;
    case 'm':
      row = 4;
         break;
         case 'n':
           row = 5;
         break;
       case 'N':
               row = 6;
         break;
         case 'f':
           row = 7;
         break;
       case 'v':
               row = 8;
         break;
         case 'T':
           row = 9;
         break;
       case 'D':
               row = 10;
         break;
         case 's':
           row = 11;
         break;
       case 'z':
               row = 12;
         break;
         case 'S':
           row = 13 ;
         break;
       case 'Z':
               row = 14;
         break;
         case 'c':
           row = 15;
         break;
       case 'j':
               row = 16;
         break;
         case 'b':
```

```
      row = 17;
    break;
  case 'd':
          row = 18;
    break;
    case 'g':
      row = 19;
    break;
  case 'p':
          row = 20;
    break;
    case 't':
      row = 21;
    break;
  case 'k':
          row = 22;
    break;
    case 'i':
      row = 23;
    break;
  case 'I':
          row = 25;
    break;
    case 'e':
      row = 27;
    break;
  case 'E':
          row = 29;
    break;
    case 'A':
      row = 31;
    break;
  case 'a':
          row = 33;
    break;
    case '>':
      row = 35;
    break;
  case '^':
          row = 37;
    break;
    case 'o':
      row = 39;
    break;
case 'U':
      row = 41;
    break;
case 'u':
      row = 43;
    break;
case '*':
      row = 45;
    break;
case '@':
      row = 47;
    break;
case '&':
```

```
           row = 49;
         break;
      case '!':
           row = 51;
         break;
      default:
        row = 52;
      break;
      }

      q = 0;
      while (q<20)
      {
        for (q2=0; q2<40; q2++)
        {
          param[index+q][q2] = utt[row][q2];
        }
      q++;
      }
      index+=20;

  }

  /* Uncomment to see parameter array */

  /*
  printf("{");

  for (i=0; i<ROWS; i++)
  {
    for (j=0; j<COLS; j++)
    {
      printf("%d,", param[i][j]);
    }
    printf("},\n{");
  }
  */

}
```

**isr.c**

```
#include "xbasic_types.h"
#include "xio.h"
#include "xparameters.h"
#include "xintc_l.h"
#include "main.h"

Xuint32 audio_dac_buf[1024];
int audio_dac_bcnt_w, audio_dac_bcnt_r;
Xuint32 audiodatain;

void audio_handler(void *callback)
{
  int  i;
  Xuint32 dacaddr, *rx_buf_addr;
```

```
  // copy 128 samples in the PASSIVE half

  dacaddr=XPAR_AK4565_BASEADDR;

  // note that 1024 is multiple of 128
  rx_buf_addr = audio_dac_buf + (audio_dac_bcnt_r & 0x3FF);

  // unroll the loop, as the time is critical
  for(i=0;i<64;i+=4){
    XIo_Out32(dacaddr+  0, rx_buf_addr[0]);
    XIo_Out32(dacaddr+  4, rx_buf_addr[1]);
    XIo_Out32(dacaddr+  8, rx_buf_addr[2]);
    XIo_Out32(dacaddr+ 12, rx_buf_addr[3]);
    dacaddr += 16;
    rx_buf_addr += 4;
  }

  audio_dac_bcnt_r += 128;

}
```

### main.h

```
/*
 * Embedded Systems Lab CSEE 4840
 *
 * Speech Synthesizer Project - main.h
 *
 */

#ifndef _MAIN_H
#define _MAIN_H

#define XPAR_AK4565_BASEADDR 0xFEFE0000
#define XPAR_AK4565_HIGHADDR 0xFEFEFFFF

#define AUDIO_BUFF_LEN 1024

#define ROWS 400
#define COLS 40

#define NUMBER_OF_SAMPLES 100
#define SAMPLE_FACTOR 0.00001

#define getrandom(min,max) ((rand()%(long)(((max)+1)-(min)))+(min))

#endif /* _MAIN_H */
```

### parwave.h

```
/*
file: PARWAVE.H
date: 15/11/93
version: 3.0
```

```
Contains structure definitions used in parwave.c
*/


#define CASCADE_PARALLEL 1         /* Type of synthesis model */
#define ALL_PARALLEL     2
#define NPAR             40        /* Number of control parameters */
#define MAX_SAM          20000     /* Maximum sample rate */
#define TRUE             1
#define FALSE            0
#define IMPULSIVE        1         /* Type of voicing source */
#define NATURAL          2
#define SAMPLED          3
#define PI               3.1415927



/* typedef's that need to be exported */

typedef char flag;

/* Resonator Structure */

typedef struct
{
  float a;
  float b;
  float c;
  float p1;
  float p2;
} resonator_t, *resonator_ptr;

/* Structure for Klatt Globals */

typedef struct
{
  flag synthesis_model; /* cascade-parallel or all-parallel */
  flag outsl;       /* Output waveform selector                    */
  long samrate;     /* Number of output samples per second         */
  long FLPhz ;      /* Frequeny of glottal downsample low-pass filter
*/
  long BLPhz ;      /* Bandwidth of glottal downsample low-pass filter
*/
  long nfcascade;   /* Number of formants in cascade vocal tract    */
  flag glsource;    /* Type of glottal source */
  int f0_flutter;   /* Percentage of f0 flutter 0-100 */
  flag quiet_flag;  /* set to TRUE for error messages */
  long nspfr;       /* number of samples per frame */
  long nper;        /* Counter for number of samples in a pitch period
*/
  long ns;
  long T0;          /* Fundamental period in output samples times 4 */
  long nopen;       /* Number of samples in open phase of period    */
  long nmod;        /* Position in period to begin noise amp. modul */
  long nrand;       /* Varible used by random number generator      */
  float pulse_shape_a;  /* Makes waveshape of glottal pulse when open
*/
```

```
  float pulse_shape_b;  /* Makes waveshape of glottal pulse when open
*/
  float minus_pi_t;
  float two_pi_t;
  float onemd;
  float decay;
  float amp_bypas; /* AB converted to linear gain              */
  float amp_voice; /* AVdb converted to linear gain            */
  float par_amp_voice; /* AVpdb converted to linear gain       */
  float amp_aspir; /* AP converted to linear gain              */
  float amp_frica; /* AF converted to linear gain              */
  float amp_breth; /* ATURB converted to linear gain           */
  float amp_gain0; /* G0 converted to linear gain              */
  int num_samples; /* number of glottal samples */
  float sample_factor; /* multiplication factor for glottal samples */
  int *natural_samples; /* pointer to an array of glottal samples */
  long original_f0; /* original value of f0 not modified by flutter */

  resonator_t rnpp; /* internal storage for resonators */
  resonator_t r1p;
  resonator_t r2p;
  resonator_t r3p;
  resonator_t r4p;
  resonator_t r5p;
  resonator_t r6p;
  resonator_t r1c;
  resonator_t r2c;
  resonator_t r3c;
  resonator_t r4c;
  resonator_t r5c;
  resonator_t r6c;
  resonator_t r7c;
  resonator_t r8c;
  resonator_t rnpc;
  resonator_t rnz;
  resonator_t rgl;
  resonator_t rlp;
  resonator_t rout;
} klatt_global_t, *klatt_global_ptr;

/* Structure for Klatt Parameters */

typedef struct
{
  long F0hz10; /* Voicing fund freq in Hz                          */
  long AVdb;   /* Amp of voicing in dB,           0 to   70       */
  long F1hz;   /* First formant freq in Hz,       200 to 1300     */
  long B1hz;   /* First formant bw in Hz,         40 to 1000      */
  long F2hz;   /* Second formant freq in Hz,      550 to 3000     */
  long B2hz;   /* Second formant bw in Hz,        40 to 1000      */
  long F3hz;   /* Third formant freq in Hz,       1200 to 4999    */
  long B3hz;   /* Third formant bw in Hz,         40 to 1000      */
  long F4hz;   /* Fourth formant freq in Hz,      1200 to 4999    */
  long B4hz;   /* Fourth formant bw in Hz,        40 to 1000      */
  long F5hz;   /* Fifth formant freq in Hz,       1200 to 4999    */
  long B5hz;   /* Fifth formant bw in Hz,         40 to 1000      */
  long F6hz;   /* Sixth formant freq in Hz,       1200 to 4999    */
```

```
  long B6hz;   /* Sixth formant bw in Hz,          40 to 2000      */
  long FNZhz;  /* Nasal zero freq in Hz,           248 to  528     */
  long BNZhz;  /* Nasal zero bw in Hz,             40 to 1000      */
  long FNPhz;  /* Nasal pole freq in Hz,           248 to  528     */
  long BNPhz;  /* Nasal pole bw in Hz,             40 to 1000      */
  long ASP;    /* Amp of aspiration in dB,         0 to   70       */
  long Kopen;  /* # of samples in open period,     10 to   65      */
  long Aturb;  /* Breathiness in voicing,          0 to   80       */
  long TLTdb;  /* Voicing spectral tilt in dB,     0 to   24       */
  long AF;     /* Amp of frication in dB,          0 to   80       */
  long Kskew;  /* Skewness of alternate periods,   0 to   40 in
sample#/2  */
  long A1;     /* Amp of par 1st formant in dB,    0 to   80       */
  long B1phz;  /* Par. 1st formant bw in Hz,       40 to 1000      */
  long A2;     /* Amp of F2 frication in dB,       0 to   80       */
  long B2phz;  /* Par. 2nd formant bw in Hz,       40 to 1000      */
  long A3;     /* Amp of F3 frication in dB,       0 to   80       */
  long B3phz;  /* Par. 3rd formant bw in Hz,       40 to 1000      */
  long A4;     /* Amp of F4 frication in dB,       0 to   80       */
  long B4phz;  /* Par. 4th formant bw in Hz,       40 to 1000      */
  long A5;     /* Amp of F5 frication in dB,       0 to   80       */
  long B5phz;  /* Par. 5th formant bw in Hz,       40 to 1000      */
  long A6;     /* Amp of F6 (same as r6pa),        0 to   80       */
  long B6phz;  /* Par. 6th formant bw in Hz,       40 to 2000      */
  long ANP;    /* Amp of par nasal pole in dB,     0 to   80       */
  long AB;     /* Amp of bypass fric. in dB,       0 to   80       */
  long AVpdb;  /* Amp of voicing,  par in dB,      0 to   70       */
  long Gain0;  /* Overall gain, 60 dB is unity,    0 to   60       */
 } klatt_frame_t, *klatt_frame_ptr;



/* function prototypes that need to be exported */

void parwave PROTO((klatt_global_ptr,klatt_frame_ptr,int*));
void parwave_init PROTO((klatt_global_ptr));
```

### proto.h

```
#ifndef PROTO
#if defined (USE_PROTOTYPES) ? USE_PROTOTYPES : defined (__STDC__)
#define PROTO(ARGS) ARGS
#else
#define PROTO(ARGS) ()
#endif
#ifndef __STDC__
#define const
#endif
#ifndef __GNUC__
#define inline
#endif
#endif
```

### opb_bram.vhd  (named opb_bram, but actually is sram controller)

```
--------------------------------------------------------------------------
--------
--
-- SRAM controller
--
-- Aisha Ahmad
-- Akshay Deoras
-- Girish Gupta
-- CSEE W4840 Embedded Systems
-- Speech Synthesizer
--
--------------------------------------------------------------------------
--------
library ieee;
use ieee.std_logic_1164.all;

entity opb_bram is

  generic (
    C_OPB_AWIDTH : integer                   := 32;
    C_OPB_DWIDTH : integer                   := 32;
    --C_BASEADDR   : std_logic_vector(0 to 31) := X"0180_0000";
    --C_HIGHADDR   : std_logic_vector(0 to 31) := X"0180_3FFF");
    C_BASEADDR : std_logic_vector(0 to 31) := X"0086_0000";
    C_HIGHADDR : std_logic_vector(0 to 31) := X"0087_FFFF");

  port (
    OPB_Clk     : in  std_logic;
    OPB_Rst     : in  std_logic;
    OPB_ABus    : in  std_logic_vector(0 to C_OPB_AWIDTH-1);
    OPB_BE      : in  std_logic_vector(0 to C_OPB_DWIDTH/8-1);
    OPB_DBus    : in  std_logic_vector(0 to C_OPB_DWIDTH-1);
    OPB_RNW     : in  std_logic;
    OPB_select  : in  std_logic;
    OPB_seqAddr : in  std_logic;        -- Sequential Address
    Sln_DBus    : out std_logic_vector(0 to C_OPB_DWIDTH-1);
    Sln_errAck  : out std_logic;        -- (unused)
    Sln_retry   : out std_logic;        -- (unused)
    Sln_toutSup : out std_logic;        -- Timeout suppress
    Sln_xferAck : out std_logic;       -- Transfer acknowledge

    PB_D        : inout std_logic_vector(0 to 15);
    PB_A        : out std_logic_vector(0 to 17);
    PB_WE       : out std_logic;
    PB_OE       : out std_logic;
    PB_LB       : out std_logic;
    PB_UB       : out std_logic;
    PB_CE       : out std_logic
    );

end opb_bram;

architecture Behavioral of opb_bram is

  constant RAM_AWIDTH : integer := 18;  -- Number of address lines on
the RAM
```

```
  constant RAM_DWIDTH : integer := 16;  -- Number of data lines on the
RAM

  --component memoryctrl
    --port (
      --rst : in std_logic;
      --clk : in std_logic;
      --cs : in std_logic;
      --select0 : in std_logic;
      --rnw : in std_logic;
      --ce0 : out std_logic;
      --ce1 : out std_logic;
      --rres : out std_logic);
  --end component;

  --Modified code
  component OBUF_F_24
    port (
      O : out std_ulogic;
      I : in  std_ulogic);
  end component;

  component IOBUF_F_24
    port (
      O  : out   std_ulogic;
      IO : inout std_ulogic;
      I  : in    std_ulogic;
      T  : in    std_ulogic);
  end component;
  --End of modified code

  signal RNW : std_logic;
  signal RAM_DI, RAM_DO : std_logic_vector(0 to RAM_DWIDTH-1);
  signal ABus : std_logic_vector(0 to RAM_AWIDTH-1);
  signal chip_select : std_logic;
  signal output_enable : std_logic;
  signal WE, RST, CE, OE : std_logic;
  signal LB, UB : std_logic;
  signal tristate : std_logic;
  signal onecycle : std_logic;
  signal rce0, rce1, rreset : std_logic;

  -- Critical: Sln_xferAck is generated directly from state bit 0!
  constant STATE_BITS : integer := 3;
  constant Idle     : std_logic_vector(0 to STATE_BITS-1) := "000";
  constant Selected : std_logic_vector(0 to STATE_BITS-1) := "001";
  constant Read     : std_logic_vector(0 to STATE_BITS-1) := "011";
  constant Xfer     : std_logic_vector(0 to STATE_BITS-1) := "111";

  signal present_state, next_state : std_logic_vector(0 to STATE_BITS-
1);

begin

  --modified port map
  addressbus: for i in 0 to 17 generate
    addresspad : OBUF_F_24 port map(
```

```
    O => PB_A(i),
    I => ABus(i)
  );
end generate;

databus: for j in 0 to 15 generate
  datapad : IOBUF_F_24 port map (
  O  => RAM_DO(j),
  IO => PB_D(j),
  I  => RAM_DI(j),
  T  => tristate);
end generate;

write_enable: OBUF_F_24 port map(
  O => PB_WE,
  I => WE
);

chip_enable: OBUF_F_24 port map(
  O => PB_CE,
  I => CE
);

o_enable: OBUF_F_24 port map(
  O => PB_OE,
  I => OE
);

lb_enable: OBUF_F_24 port map(
  O => PB_LB,
  I => LB
);

ub_enable: OBUF_F_24 port map(
  O => PB_UB,
  I => UB
);

-- Memory control/arbitration state machine

--memoryctrl1 : memoryctrl port map (
  --rst => OPB_Rst,
  --clk => OPB_Clk,
  --cs => chip_select,
  --select0 => OPB_select,
  --rnw => RNW,
  --ce0 => rce0,
  --ce1 => rce1,
  --rres => rreset);

 --end of modified port map

register_opb_inputs: process (OPB_Clk, OPB_Rst)
begin
  if OPB_Rst = '1' then
    RAM_DI <= (others => '0');
    --data
```

```
      ABus <= (others => '0');
      RNW <= '0';
    elsif OPB_Clk'event and OPB_Clk = '1' then
      RAM_DI <= OPB_DBus(0 to RAM_DWIDTH-1);
      --data
      ABus <= OPB_ABus(C_OPB_AWIDTH-3-(RAM_AWIDTH-1) to C_OPB_AWIDTH-
3);
      RNW <= OPB_RNW;
    end if;
  end process register_opb_inputs;

  register_opb_outputs: process (OPB_Clk, OPB_Rst)
  begin
    if OPB_Rst = '1' then
      Sln_DBus(0 to RAM_DWIDTH-1) <= (others => '0');
      --data
    elsif OPB_Clk'event and OPB_Clk = '1' then
     if output_enable = '1' then
        Sln_DBus(0 to RAM_DWIDTH-1) <= RAM_DO;
      --data
     else
       Sln_DBus(0 to RAM_DWIDTH-1) <= (others => '0');
            --data
     end if;
    end if;
  end process register_opb_outputs;

    -- Write the low two bytes if rce0 or rce1 is enabled

  --register_opb_outputs1: process (OPB_Clk, OPB_Rst)
  --begin
    --if OPB_Rst = '1' then
      --Sln_DBus(0 to 15) <= X"0000";
    --elsif OPB_Clk'event and OPB_Clk = '1' then
      --if rreset = '1' then
        --Sln_DBus(0 to 15) <= X"0000";
      --elsif (rce1 or rce0) = '1' then
        --Sln_DBus(0 to 15) <= RAM_DO(0 to 15);
      --end if;
    --end if;
  --end process;

  -- Write the high two bytes if rce0 is enabled

  --register_opb_outputs2: process (OPB_Clk, OPB_Rst)
 -- begin
    --if OPB_Rst = '1' then
      --Sln_DBus(16 to 31) <= X"0000";
    --elsif OPB_Clk'event and OPB_Clk = '1' then
      --if rreset = '1' then
        --Sln_DBus(16 to 31) <= X"0000";
      --elsif rce0 = '1' then
       -- -Sln_DBus(16 to 31) <= RAM_DO(0 to 15);
      --end if;
    --end if;
  --end process;
```

```
  -- Unused outputs
  Sln_errAck  <= '0';
  Sln_retry   <= '0';
  Sln_toutSup <= '0';
  Sln_DBus(RAM_DWIDTH to C_OPB_DWIDTH-1) <= (others => '0');
            --data

  --chip_select <=
    --'1' when OPB_select = '1' and
      --  OPB_ABus(0 to C_OPB_AWIDTH-3-RAM_AWIDTH) =
        --C_BASEADDR(0 to C_OPB_AWIDTH-3-RAM_AWIDTH) else
    --'0';

  chip_select <= OPB_select when OPB_ABus(31 downto 20) = X"008" else
'0';

  -- Sequential part of the FSM
  fsm_seq : process(OPB_Clk, OPB_Rst)
  begin
    if OPB_Rst = '1' then
      present_state <= Idle;
    elsif OPB_Clk'event and OPB_Clk = '1' then
      present_state <= next_state;
    end if;
  end process fsm_seq;

  -- Combinational part of the FSM
  fsm_comb : process(OPB_Rst, present_state, chip_select, OPB_Select,
RNW)
  begin
    RST <= '1';                         -- Default values
    WE <= '1';                          -- modified
    output_enable <= '0';
    tristate <= '1';                    -- modified
    CE <= '0';
    OE <= '0';
    LB <= '0';
    UB <= '0';
    if OPB_RST = '1' then
      next_state <= Idle;
    else
      case present_state is
        when Idle =>
          if chip_select = '1' then
            next_state <= Selected;
          else
            next_state <= Idle;
          end if;

        when Selected =>
          if OPB_Select = '1' then
            if RNW = '1' then
              RST <= '0';
              output_enable <= '1';     -- modified
              tristate <= '1';
              next_state <= Read;
            else
```

```
              WE <= '0';                -- modified
              tristate <= '0';
              next_state <= Xfer;
            end if;
          else
            next_state <= Idle;
          end if;

        when Read =>
          if OPB_Select = '1' then
            -- tristate <= '1';         -- modified
            output_enable <= '1';
            next_state <= Xfer;
          else
            next_state <= Idle;
          end if;

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

        when others =>
          next_state <= Idle;
      end case;
    end if;
  end process fsm_comb;

  Sln_xferAck <= present_state(0);

end Behavioral;

-- Local Variables:
-- compile-command: "ghdl -a opb_bram.vhd"
-- End:
```

## opb_bram_v2_1_0.mpd

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

BEGIN opb_bram, IPTYPE = PERIPHERAL, EDIF=TRUE

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

OPTION IMP_NETLIST=TRUE

## Generics for VHDL
##PARAMETER c_baseaddr     = 0x01800000, DT = std_logic_vector,
MIN_SIZE = 0xFF
```

```
##PARAMETER c_highaddr     = 0x01803FFF, DT = std_logic_vector

PARAMETER c_baseaddr = 0x00860000, DT = std_logic_vector, MIN_SIZE =
0xFF
PARAMETER c_highaddr = 0x0087FFFF, DT = std_logic_vector

PARAMETER c_opb_awidth   = 32,         DT = integer
PARAMETER c_opb_dwidth   = 32,         DT = integer

## Ports
PORT opb_abus    = OPB_ABus,    DIR = IN, VEC = [0:(c_opb_awidth-1)],
BUS = SOPB
PORT opb_be      = OPB_BE,      DIR = IN, VEC = [0:((c_opb_dwidth/8)-
1)], BUS = SOPB
PORT opb_clk     = "",          DIR = IN,                     BUS =
SOPB
PORT opb_dbus    = OPB_DBus,    DIR = IN, VEC = [0:(c_opb_dwidth-1)],
BUS = SOPB
PORT opb_rnw     = OPB_RNW,     DIR = IN,
BUS = SOPB
PORT opb_rst     = OPB_Rst,     DIR = IN,
BUS = SOPB
PORT opb_select  = OPB_select,  DIR = IN,
BUS = SOPB
PORT opb_seqaddr = OPB_seqAddr, DIR = IN,
BUS = SOPB
PORT sln_dbus    = Sl_DBus,     DIR = OUT, VEC = [0:(c_opb_dwidth-1)],
BUS = SOPB
PORT sln_errack  = Sl_errAck,   DIR = OUT,
BUS = SOPB
PORT sln_retry   = Sl_retry,    DIR = OUT,
BUS = SOPB
PORT sln_toutsup = Sl_toutSup,  DIR = OUT,
BUS = SOPB
PORT sln_xferack = Sl_xferAck,  DIR = OUT,
BUS = SOPB

PORT PB_D    = "",             DIR=INOUT,  VEC=[15:0],
      3STATE=FALSE, IOB_STATE=BUF
PORT PB_A    = "",             DIR=OUT,    VEC=[17:0],
      3STATE=FALSE, IOB_STATE=BUF
PORT PB_WE   = "",             DIR=OUT,
      3STATE=FALSE, IOB_STATE=BUF
PORT PB_OE   = "",             DIR=OUT,
      3STATE=FALSE, IOB_STATE=BUF
PORT PB_LB   = "",             DIR=OUT,
      3STATE=FALSE, IOB_STATE=BUF
PORT PB_UB   = "",             DIR=OUT,
      3STATE=FALSE, IOB_STATE=BUF
PORT PB_CE   = "",             DIR=OUT,
      3STATE=FALSE, IOB_STATE=BUF

END
```

### opb_bram_v2_1_0.pao

```
##############################################################
#
# opb_bram pao file
#
##############################################################

lib opb_bram_v1_00_a opb_bram
#lib opb_bram_v1_00_a memoryctrl
```

### opb_xsb300_ak4565.vhd

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

library UNISIM;
use UNISIM.VComponents.all;

entity opb_xsb300_ak4565 is
  generic (
    C_OPB_AWIDTH    : integer := 32;
    C_OPB_DWIDTH    : integer := 32;
    C_BASEADDR      : std_logic_vector(31 downto 0) := X"FEFE_0000";
    C_HIGHADDR      : std_logic_vector(31 downto 0) := X"FEFE_FFFF"
    );

  port (
    OPB_Clk : in std_logic;
    OPB_Rst : in std_logic;
    OPB_ABus : in std_logic_vector (31 downto 0);
    OPB_BE : in std_logic_vector (3 downto 0);
    OPB_DBus : in std_logic_vector (31 downto 0);
    OPB_RNW : in std_logic;
    OPB_select : in std_logic;
    OPB_seqAddr : in std_logic;
    UIO_DBus : out std_logic_vector (31 downto 0);
    UIO_errAck : out std_logic;
    UIO_retry : out std_logic;
    UIO_toutSup : out std_logic;
    UIO_xferAck : out std_logic;

    ak4565_intr : out std_logic;
    Interrupt : out std_logic;

    AU_CSN : out std_logic;
    AU_BCLK : out std_logic;
    AU_MCLK : out std_logic;
    AU_LRCK : out std_logic;
    AU_SDTI : out std_logic;
    AU_SDTO0 : in std_logic
  );

end opb_xsb300_ak4565;

architecture Behavioral of opb_xsb300_ak4565 is
```

```
component audio_ak4565
  port (
    CLK : in std_logic;
    RST : in std_logic;

    audio_bram_clk : out std_logic;
    audio_bram_addr_dac : out std_logic_vector(11 downto 0);
    audio_bram_addr_adc : out std_logic_vector(11 downto 0);

    audio_bram_dacdata : in std_logic;
    audio_bram_adcdata : out std_logic;

    interrupt : out std_logic;
    audio_fifohalf : out std_logic;

    AU_CSN : out std_logic;
    AU_BCLK : out std_logic;
    AU_MCLK : out std_logic;
    AU_LRCK : out std_logic;
    AU_SDTI : out std_logic;
    AU_SDTO0 : in std_logic
  );

end component;

signal cs, xfer, xfer_1, xfer_2 : std_logic;

signal rnw : std_logic;
signal addr : std_logic_vector (15 downto 0);
signal wdata : std_logic_vector (31 downto 0);

signal rdata : std_logic_vector (31 downto 0);
signal opb_ad_ce : std_logic;

signal we, a0, ce0, ce1 : std_logic;
signal bram_rdata, bram_wdata : std_logic_vector(15 downto 0); -- as
the CPU sees them
signal bram_rdata_rev, bram_wdata_rev : std_logic_vector(15 downto 0);
-- reversed !
signal bram_addr : std_logic_vector(7 downto 0);

signal audio_bram_clk : std_logic;
signal audio_bram_addr_dac, audio_bram_addr_adc : std_logic_vector(11
downto 0);
signal audio_bram_dacdata : std_logic;
signal audio_bram_adcdata : std_logic;
signal audio_bram_dacdata_v : std_logic_vector(0 downto 0);
signal audio_bram_adcdata_v : std_logic_vector(0 downto 0);

signal audio_fifohalf : std_logic;

begin

process(OPB_Rst, OPB_Clk)
begin

-- register adresses, data write, rnw
```

```
   if OPB_Rst = '1' then
          addr <=  X"0000";
          wdata <= X"0000_0000";
          rnw <= '0';
      elsif OPB_Clk'event and OPB_Clk ='1' then
          if opb_ad_ce = '1' then
                        wdata <= OPB_DBus;
                        addr <= OPB_ABus(15 downto 0);
             rnw <= OPB_RNW;
                   end if;
      end if;

-- register data read
   if OPB_Rst = '1' then
                  rdata <= X"0000_0000";
      elsif OPB_Clk'event and OPB_Clk ='1' then
                   if(ce0='1') then rdata(15 downto 0) <= bram_rdata;
end if;
                   if(ce1='1') then rdata(31 downto 16) <= bram_rdata;
end if;
      end if;

end process;

-- very important
-- TO DO
-- when writing, the read data can corrupe the DBus
      UIO_DBus <= rdata when xfer = '1' else X"0000_0000";


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

-- the 1st ff -- FDR
process(OPB_Clk)
begin
      if OPB_Clk'event and OPB_Clk ='1' then
            if (xfer or xfer_1) = '1' then xfer <='0'; else xfer <= cs;
end if;
      end if;
end process;

process (OPB_Rst, OPB_Clk)
begin
      if OPB_Rst = '1' then
       xfer_1 <= '0';
             xfer_2 <= '0';
   elsif OPB_Clk'event and OPB_Clk ='1' then
             xfer_1 <= xfer;
             xfer_2 <= xfer_1 and not rnw;
      end if;
end process;

-- combinational logic for BRAM
we <= (xfer or xfer_1) and not rnw;
ce0 <= xfer_1 and not rnw;
ce1 <= xfer_2;
a0 <= xfer_1;
```

```
opb_ad_ce <= not xfer;

bram_addr <= (not audio_fifohalf) & addr(7 downto 2) & a0;
bram_wdata <= wdata(31 downto 16) when a0='0' else wdata(15 downto 0);

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


-- instantiate the BRAMS
process (bram_rdata_rev) begin
      for i in 0 to 15 loop
            bram_rdata(i)<=bram_rdata_rev(15-i);
      end loop;
end process;

process(bram_wdata) begin
      for i in 0 to 15 loop
            bram_wdata_rev(i)<=bram_wdata(15-i);
      end loop;
end process;

audio_bram_dacdata <= audio_bram_dacdata_v(0);
dac_bram : RAMB4_S1_S16 port map (
              DIA    => "0" ,
        ENA    => '1',
              WEA    => '0',
        RSTA   => '0',
              CLKA   => audio_bram_clk,
              ADDRA  => audio_bram_addr_dac,
              DOA    => audio_bram_dacdata_v,

              DIB    => bram_wdata_rev,
        ENB    => '1',
        WEB    => we,
              RSTB   => '0',
        CLKB   => OPB_Clk,
        ADDRB  => bram_addr,
        DOB    => open
);

audio_bram_adcdata_v(0) <= audio_bram_adcdata ;
adc_bram : RAMB4_S1_S16 port map (
              DIA    => audio_bram_adcdata_v,
        ENA    => '1',
              WEA    => '1',
        RSTA   => '0',
              CLKA   => audio_bram_clk,
              ADDRA  => audio_bram_addr_adc,
              DOA    => open,

              DIB    => X"0000",
        ENB    => '1',
        WEB    => '0',
```

```
              RSTB   => '0',
        CLKB   => OPB_Clk,
        ADDRB  => bram_addr,
        DOB    => bram_rdata_rev
);



-- the sound stuff

audio : audio_ak4565 port map(

  CLK => OPB_Clk,
  RST => OPB_Rst,

  audio_bram_clk => audio_bram_clk,
  audio_bram_addr_dac => audio_bram_addr_dac,
  audio_bram_addr_adc => audio_bram_addr_adc,
  audio_bram_dacdata => audio_bram_dacdata,
  audio_bram_adcdata => audio_bram_adcdata,

  interrupt => interrupt,
  audio_fifohalf => audio_fifohalf,

  AU_CSN => AU_CSN,
  AU_BCLK => AU_BCLK,
  AU_MCLK => AU_MCLK,
  AU_LRCK => AU_LRCK,
  AU_SDTI => AU_SDTI,
  AU_SDTO0 => AU_SDTO0
);

end Behavioral;
```

### audio_ak4565.vhd

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

library UNISIM;
use UNISIM.VComponents.all;


entity audio_ak4565 is

port (
  CLK : in std_logic;
  RST : in std_logic;

  audio_bram_clk : out std_logic;
  audio_bram_addr_dac : out std_logic_vector(11 downto 0);
  audio_bram_addr_adc : out std_logic_vector(11 downto 0);
```

```
  audio_bram_dacdata : in std_logic;
  audio_bram_adcdata : out std_logic;

  interrupt : out std_logic;
  audio_fifohalf : out std_logic;

  AU_CSN : out std_logic;
  AU_BCLK : out std_logic;
  AU_MCLK : out std_logic;
  AU_LRCK : out std_logic;
  AU_SDTI : out std_logic;
  AU_SDTO0 : in std_logic
);

end audio_ak4565;


architecture Behavioral of audio_ak4565 is

component RAMB4_S1_S16
--
  generic (
       INIT_00 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_01 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_02 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_03 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000"
            );
--
  port (DIA    : in STD_LOGIC_VECTOR (0 downto 0);
        DIB    : in STD_LOGIC_VECTOR (15 downto 0);
        ENA    : in STD_logic;
        ENB    : in STD_logic;
        WEA    : in STD_logic;
        WEB    : in STD_logic;
        RSTA   : in STD_logic;
        RSTB   : in STD_logic;
        CLKA   : in STD_logic;
        CLKB   : in STD_logic;
        ADDRA  : in STD_LOGIC_VECTOR (11 downto 0);
        ADDRB  : in STD_LOGIC_VECTOR (7 downto 0);
        DOA    : out STD_LOGIC_VECTOR (0 downto 0);
        DOB    : out STD_LOGIC_VECTOR (15 downto 0));
end component;

signal clkcnt : std_logic_vector(4 downto 0);
signal audiocnt_dac, audiocnt_adc : std_logic_vector(11 downto 0);

signal audio_clk : std_logic;
signal lrck : std_logic;

begin

AU_CSN <= '1'; -- no chip select as we don't write the ctrl regs
```

```
-- generate the 3 clocks: master, serial, frame

process(CLK, RST)
begin
      if rst = '1' then
            clkcnt <= "00000";
      elsif clk'event and clk='1' then
            clkcnt <= clkcnt + 1;
      end if;
end process;
AU_MCLK <= clkcnt(1); -- master clock, 12.5 MHz
audio_clk <= clkcnt(4); -- this is the serial clock, 1.5625 Mhz
AU_BCLK <= not audio_clk; -- dont't ask but read AK4565 specs

process(audio_clk, RST)
begin
      if rst = '1' then
      audiocnt_dac <= "000000000000";
      audiocnt_adc <= "111111111111";
      lrck <= '0';
      elsif audio_clk'event and audio_clk='1' then
            audiocnt_dac <= audiocnt_dac + 1;
            audiocnt_adc <= audiocnt_adc + 1;
            lrck <= not audiocnt_dac(4);
      end if;
end process;
AU_LRCK <= lrck; -- audio clock, 48.828 kHz

interrupt <= not audiocnt_dac(10);
audio_fifohalf <= audiocnt_dac(11);

audio_bram_addr_dac <= audiocnt_dac;
audio_bram_addr_adc <= audiocnt_adc;
audio_bram_clk <= audio_clk;

AU_SDTI <= audio_bram_dacdata;
audio_bram_adcdata <= AU_SDTO0;

end architecture;
```

### opb_xsb300_ak4565_v2_1_0.mpd

```
##############################################################################
#########
##
## Copyright (c) 1995-2002 Xilinx, Inc.  All rights reserved.
##
## opb_xsb300_ak4565_v2_1_0.mpd
##
## Microprocessor Peripheral Definition
##
##############################################################################
#########
```

```
PARAMETER VERSION = 2.0.0

BEGIN opb_xsb300_ak4565, IPTYPE = PERIPHERAL
OPTION IMP_NETLIST = TRUE
OPTION HDL = VHDL
#OPTION CORE_STATE = DEVELOPMENT

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

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


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

Port Interrupt ="", DIR=OUT, EDGE=RISING, SIGIS=INTERRUPT

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

PORT AU_CSN = "", DIR=OUT
PORT AU_BCLK = "", DIR=OUT
PORT AU_MCLK = "", DIR=OUT
PORT AU_LRCK = "", DIR=OUT
PORT AU_SDTI = "", DIR=OUT
PORT AU_SDTO0 = "", DIR=IN

END
```

### opb_xsb300_ak4565_v2_1_0.pao

```
##############################################################################
#########
##
## Copyright (c) 1995-2002 Xilinx, Inc.  All rights reserved.  Xilinx,
Inc.
##
## MicroBlaze_Brd_ZBT_ClkGen_v2_0_0_a.pao
##
```

```
## Peripheral Analyze Order
##
############################################################################
#########


lib opb_xsb300_ak4565_v1_00_a opb_xsb300_ak4565
lib opb_xsb300_ak4565_v1_00_a audio_ak4565
```

### Makefile

```
#
# CSEE W4840 Embedded Systems
# Speech Synthesizer
#
# Makefile
#
# Aisha Ahmad
# Akshay Deoras
# Girish Gupta
#

SYSTEM = system

MICROBLAZE_OBJS = \
      c_source_files/main.o \
      c_source_files/isr.o \
      c_source_files/utt.o

LIBRARIES = mymicroblaze/lib/libxil.a

ELF_FILE = $(SYSTEM).elf

NETLIST = implementation/$(SYSTEM).ngc

# Bitstreams for the FPGA

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

# Files to be downloaded to the SRAM

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

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

FPGA_ARCH = spartan2e
DEVICE = xc2s300epq208-6

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

```
# Paths for programs

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

XILINX_EDK = /usr/cad/xilinx/edk6.2i
EDKBINDIR = $(XILINX_EDK)/bin/lin
EDKENVCMDS = LD_LIBRARY_PATH=$(ISEBINDIR):$(EDKBINDIR) XILINX=$(XILINX)
XILINX_EDK=$(XILINX_EDK) PATH=$(ISEBINDIR):$(EDKBINDIR):$(PATH)

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

# Executables

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

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

MICROBLAZE_CC = /usr/cad/xilinx/gnu/bin/mb-gcc
MICROBLAZE_CC_SIZE = /usr/cad/xilinx/gnu/bin/microblaze-size
MICROBLAZE_OBJCOPY = /usr/cad/xilinx/gnu/bin/microblaze-objcopy

#MICROBLAZE_CC = $(XILINX_EDK)/gnu/microblaze/lin/bin/mb-gcc
#MICROBLAZE_CC_SIZE = $(XILINX_EDK)/gnu/microblaze/lin/bin/mb-size
#MICROBLAZE_OBJCOPY = $(XILINX_EDK)/gnu/microblaze/lin/bin/mb-objcopy

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

# External Targets

all :
      @echo "Makefile to build a Microprocessor system :"
      @echo "Run make with any of the following targets"
      @echo "  make libs     : Configures the sw libraries for this
system"
      @echo "  make program  : Compiles the program sources for all the
processor instances"
      @echo "  make netlist  : Generates the netlist for this system
($(SYSTEM))"
      @echo "  make bits     : Runs Implementation tools to generate
the bitstream"
      @echo "  make init_bram: Initializes bitstream with BRAM data"
      @echo "  make download : Downloads the bitstream onto the board"
      @echo "  make netlistclean: Deletes netlist"
```

```
	@echo "  make hwclean  : Deletes implementation dir"
	@echo "  make libsclean: Deletes sw libraries"
	@echo "  make programclean: Deletes compiled ELF files"
	@echo "  make clean    : Deletes all generated files/directories"
	@echo " "
	@echo "  make <target> : (Default)"
	@echo "      Creates a Microprocessor system using default
initializations"
	@echo "      specified for each processor in MSS file"


bits : $(FPGA_BITFILE)

netlist : $(NETLIST)

libs : $(LIBRARIES)

program : $(ELF_FILE)

init_bram : $(MERGED_BITFILE)

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

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

swclean : libsclean programclean

oneclean :
	rm -rf $(NETLIST)
	@echo "Now rm other .ngc files in implementation/ and
implentation/cache"

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

libsclean :
	rm -rf mymicroblaze/lib

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

#
# Software rules
#

MICROBLAZE_MODE = executable

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

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

```
# Compilation

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

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

# Linking

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

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

#
# Hardware rules
#

# Hardware compilation : optimize the netlist, place and route

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

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

```
$(NETLIST) : $(MHSFILE)
	$(PLATGEN) $(PLATGEN_OPTIONS) -st xst $(MHSFILE)
	$(XST) -ifn synthesis/$(SYSTEM)_xst.scr

#
# Downloading
#

# Add software code to the FPGA bitfile

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

# Create a .hex file with data for the SRAM

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

# Download the files to the target board

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

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

### mylinkscript

```
/*
 * Instructions for the linker about where in memory to locate various
 * portions of the program.  See "info ld" for details.
 */

/* List of memory blocks */
MEMORY {
  BRAM : ORIGIN = 0x00000000, LENGTH = 0x01000 /* On the FPGA */
  SRAM : ORIGIN = 0x00860000, LENGTH = 0x20000 /* SRAM past the
framebuffer */
  /*SRAM : ORIGIN = 0x01800000, LENGTH = 0x03FFF*/
}

/* Symbol where the program will start executing */
ENTRY(_start)

/* Instructions on where to locate each segment of the program */
SECTIONS
{

  /*
```

```
   * Critical code to place on the FPGA: initialization, interrupts,
etc.
   */

  .bram_text : {
      /usr/cad/xilinx/gnu/lib/gcc-lib/microblaze/2.95.3-4/crt0.o(.text)
      /usr/cad/xilinx/gnu/lib/gcc-lib/microblaze/2.95.3-
4/crtinit.o(.text)
      ./mymicroblaze/lib/libxil.a(.text)
      ./mymicroblaze/lib//libxil.a(.text)
      c_source_files/isr.o(.text)
      /*c_source_files/speech.o(.text)*/
  } > BRAM

  /*
   * The stack
   */

  . = ALIGN(4);
  .stack : {
     _STACK_SIZE = 0x200;
     . += _STACK_SIZE;
     . = ALIGN(4);
  } > BRAM
  _stack = .;  /* It starts here and grows downward */

  /*
   * Code to be placed in SRAM: the rest of the program
   */

  . = ALIGN(4);
  .sram_text : {
    *(.text)
  } > SRAM

  /*
   * Small initialized read-only memory
   */

  . = ALIGN(8);
  _ss_small = .;
  .sdata2 . : {
    *(.sdata2)
  } > SRAM

  /*
   * Small initialized memory
   */

  . = ALIGN(8);
  .sdata : {
    *(.sdata)
  } > SRAM

  . = ALIGN(8);
  _es_small = .;
  _ssize_small = _es_small - _ss_small;
```

```
  _SDA2_BASE_ = _ss_small + (_ssize_small / 2 );

  /*
   * Initialized read-only memory
   */

  . = ALIGN(4);
  .rodata . : {
    *(.rodata)
  } > SRAM

  /*
   * Initialized memory
   */

  . = ALIGN(4);
  .data : {
    *(.data)
  } > SRAM

  /* SBSS and BSS */

  . = ALIGN(8);
  __sbss_start = .;
  .sbss : {
     *(.sbss)
  } > SRAM
  . = ALIGN(8);
  __sbss_end = .;

  __sbss_size = __sbss_end - __sbss_start;
  _SDA_BASE_ = __sbss_start + (__sbss_size / 2);

  /*
   * The heap
   */

  . = ALIGN(4);
  .sram_bss : {
    __bss_start = .;
    *(.bss)
    *(COMMON)
    . = ALIGN(4);
    __bss_end = . ;
    _heap = .;
  } > SRAM

  _end = .;

}
```

### system.mhs

```
#
# CSEE W4840 Embedded Systems
# Speech Synthesizer
```

```
#
# Aisha Ahmad
# Akshay Deoras
# Girish Gupta
#

# Parameters
PARAMETER VERSION = 2.1.0

# Global Ports

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

PORT PB_D = PB_D, DIR = INOUT, VEC=[15:0]
PORT PB_A = PB_A, DIR = OUT, VEC=[17:0]
PORT PB_WE = PB_WE, DIR = OUT
PORT PB_OE = PB_OE, DIR = OUT
PORT PB_LB = PB_LB, DIR = OUT
PORT PB_UB = PB_UB, DIR = OUT
PORT PB_CE = PB_CE, DIR = OUT

PORT AU_CSN = AU_CSN, DIR = OUT
PORT AU_MCLK = AU_MCLK, DIR = OUT
PORT AU_LRCK = AU_LRCK, DIR = OUT
PORT AU_BCLK = AU_BCLK, DIR = OUT
PORT AU_SDTI = AU_SDTI, DIR = OUT
PORT AU_SDTO0 = AU_SDTO0, DIR = IN

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

BEGIN opb_xsb300_ak4565
 PARAMETER INSTANCE = audio
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0xFEFE0000
 PARAMETER C_HIGHADDR = 0xFEFEFFFF
 PORT AU_CSN = AU_CSN
 PORT AU_BCLK =  AU_BCLK
 PORT AU_MCLK = AU_MCLK
 PORT AU_LRCK = AU_LRCK
 PORT AU_SDTI = AU_SDTI
 PORT AU_SDTO0 = AU_SDTO0
 PORT OPB_Clk = sys_clk
 PORT Interrupt = audio_intr
 BUS_INTERFACE SOPB = myopb_bus
END

BEGIN opb_intc
 PARAMETER INSTANCE = intc
 PARAMETER HW_VER = 1.00.c
 PARAMETER C_BASEADDR = 0xFFFF0000
 PARAMETER C_HIGHADDR = 0xFFFF00FF
 PORT OPB_Clk = sys_clk
 PORT Intr =  audio_intr
 PORT Irq =   intr
```

```
 BUS_INTERFACE SOPB = myopb_bus
END

BEGIN opb_bram
  PARAMETER INSTANCE = bram_peripheral
  PARAMETER HW_VER = 1.00.a
  #PARAMETER C_BASEADDR = 0x01800000
  #PARAMETER C_HIGHADDR = 0x01803FFF
  PARAMETER C_BASEADDR = 0x00860000
  PARAMETER C_HIGHADDR = 0x0087FFFF
  PORT OPB_Clk = sys_clk
  BUS_INTERFACE SOPB = myopb_bus
  PORT PB_D = PB_D
  PORT PB_A = PB_A
  PORT PB_WE = PB_WE
  PORT PB_OE = PB_OE
  PORT PB_LB = PB_LB
  PORT PB_UB = PB_UB
  PORT PB_CE = PB_CE
END

# The main processor core

BEGIN microblaze
 PARAMETER INSTANCE = mymicroblaze
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_USE_BARREL = 1
 #PARAMETER C_USE_ICACHE = 0
 PARAMETER C_USE_ICACHE = 1
 PARAMETER C_ADDR_TAG_BITS = 6
 PARAMETER C_CACHE_BYTE_SIZE = 2048
 PARAMETER C_ICACHE_BASEADDR = 0x00860000
 PARAMETER C_ICACHE_HIGHADDR = 0x0087FFFF
 PORT Clk = sys_clk
 PORT Reset = fpga_reset
 BUS_INTERFACE DLMB = d_lmb
 BUS_INTERFACE ILMB = i_lmb
 BUS_INTERFACE DOPB = myopb_bus
 BUS_INTERFACE IOPB = myopb_bus
END

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

# Data LMB bus

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

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = lmb_data_controller
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_BASEADDR = 0x00000000
```

```
 PARAMETER C_HIGHADDR = 0x00000FFF
 BUS_INTERFACE SLMB = d_lmb
 BUS_INTERFACE BRAM_PORT = conn_0
END

# Instruction LMB bus

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

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = lmb_instruction_controller
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00000FFF
 BUS_INTERFACE SLMB = i_lmb
 BUS_INTERFACE BRAM_PORT = conn_1
END

# The actual block memory

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

# Clock divider to make the whole thing run

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

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

BEGIN opb_v20
 PARAMETER INSTANCE = myopb_bus
 PARAMETER HW_VER = 1.10.a
 PARAMETER C_DYNAM_PRIORITY = 0
 PARAMETER C_REG_GRANTS = 0
 PARAMETER C_PARK = 0
 PARAMETER C_PROC_INTRFCE = 0
 PARAMETER C_DEV_BLK_ID = 0
 PARAMETER C_DEV_MIR_ENABLE = 0
 PARAMETER C_BASEADDR = 0x0fff1000
 PARAMETER C_HIGHADDR = 0x0fff10ff
```

```
 PORT SYS_Rst = fpga_reset
 PORT OPB_Clk = sys_clk
END

# UART: Serial port hardware

BEGIN opb_uartlite
 PARAMETER INSTANCE = myuart
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_CLK_FREQ = 50_000_000
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_BASEADDR = 0xFEFF0100
 PARAMETER C_HIGHADDR = 0xFEFF01FF
 PORT OPB_Clk = sys_clk
 BUS_INTERFACE SOPB = myopb_bus
 PORT RX=RS232_RD
 PORT TX=RS232_TD
END
```

### system.mss

```
#
# CSEE W4840 Embedded Systems
# Speech Synthesizer
#
# Aisha Ahmad
# Akshay Deoras
# Girish Gupta
#

PARAMETER VERSION = 2.2.0
PARAMETER HW_SPEC_FILE = system.mhs

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

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

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

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

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

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

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

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

BEGIN DRIVER
 PARAMETER HW_INSTANCE = intc
 PARAMETER DRIVER_NAME = intc
 PARAMETER DRIVER_VER = 1.00.b
 PARAMETER LEVEL = 0
END
```

### system.ucf

```
net sys_clk period = 18.000;

net FPGA_CLK1 loc="p77";

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

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

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

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

net PB_WE loc = "p123";
net PB_OE loc = "p125";
net PB_LB loc = "p140";
net PB_UB loc = "p146";
net PB_CE loc = "p147";
```