Detection of Breathing and Infant Sleep Apnea

By:
Brian Berg

Senior Project
Computer Engineering Department,
California Polytechnic State University,
San Luis Obispo
June 2011

Table of Contents

List of Figures

Acknowledgments

I would like to thank the students who worked with me on this project, David Bydalek and Ricky Hennessy and those who worked on this project in previous years.

I would also like to thank all on those who advised and supported the project: Dr. Lily Laiho for leading the project; Dr. John Jacobs and Aaron Stein, PE from Raytheon; Dr. Lynne Slivovsky for being my advisor and Dr. Wayne Pilkington for offering much needed technical assistance in the last weeks of the project.

Special thanks to Raytheon for supporting the project both in funding and with technical assistance.

Abstract

Sleep apnea is a condition where people pause while breathing in their sleep; this can be of great concern for infants and premature babies. Current monitoring systems either require physical attachment to a user or may be unreliable. This project is meant to develop a device that can accurately detect breathing through sound and issue appropriate warnings upon its cessation. The device produced is meant to be a standalone device and thus was developed as an embedded systems project on a Xilinx Spartan 6 FPGA.

I. Introduction

This project is a breath detection system; the particular aim of the project is to be able to detect the breathing of an infant. By being able to detect breathing you can notice when it stops and for how long, this is important due sleep apnea. Sleep apnea is a condition where people pause their breathing while sleeping. This can potentially be hazardous, especially for infant and premature babies where it is called apnea of prematurity if they are less than 37 weeks and apnea of infancy if they are older than 37 weeks[1]. Apnea events are classified as cessation of breathing at least 20 seconds or longer. There is also a possible link between sleep apnea and sudden infant death syndrome, though it is debated[3].

Various monitors already exist, some use attached electrical leads on the body to determine breathing and heart beats[2], while others are vibration sensors that detect movement of the baby. Monitors relying upon sensors attached to the body can be cumbersome and movement sensors are not always accurate. To this end a project was set out to build something that worked better without requiring direct contact with the body.

Breath analysis is a broad subject and is mostly beyond the simple detection of a single breath to include characterization. Of course these systems also use advanced techniques such as neural nets and genetic algorithms[4]. For obvious reasons such advanced systems are impractical for compact, portable systems and thus were somewhat limited in aiding the project.

II. Project Definition

Background

This project is a continuation of a project that has been ongoing in Cal Poly's BMED department with support from Raytheon. The initial phase was to research possible methods for breath detection, that resulted in a recommendation of audio detection and CO2 monitoring. Audio monitoring was pursued with emphasis on an application inside a hospital. Such a noisy environment would require special consideration and a method to use two microphones to detect breathing and cancel out background noise by differential audio. A solution utilizing a general purpose computer was also pursued.

In the current iteration an embedded system audio processing implementation is desired. The use of a single microphone with a parabolic dish to focus the sound and help eliminate was considered when this stage project started. In this stage microphones with parabolic dishes was not the focus but may be useful in future iterations to improve upon this work.

System Requirements

One of the crucial points of the device is that it be accurate, with extremely little chance of false negatives and few false positives while detecting apnea events. The device must be easy to operate and setup. No special training should be required to use or maintain the device and setup requirements should be minimal and unobtrusive. The device itself should be small in dimension not requiring large spaces and have few remote connections. The device should be powered from the wall with possible support for battery backup. To meet these needs some form of embedded system is desired.

III. Design

Overview

The overall design involves acquiring sound from a microphone, this sound is then processed to detect breathing and a timer counts how long between breaths. When an apnea event occurs, longer than 20 seconds without breath, an alarm is sounded.

Matlab Prototype

One of the first steps in the project's design was a Matlab model created by Ricky Hennessy. This served as a proof of concept and a much easier platform to debug than methods used than hardware can easily provide. The program took in audio data from a microphone and detected the peaks in the signal, as seen in Figure 1 below. From the Matlab code a Simulink model was derived, the filtering portion of this code was used to produce the VHDL filter that will be discussed shortly.

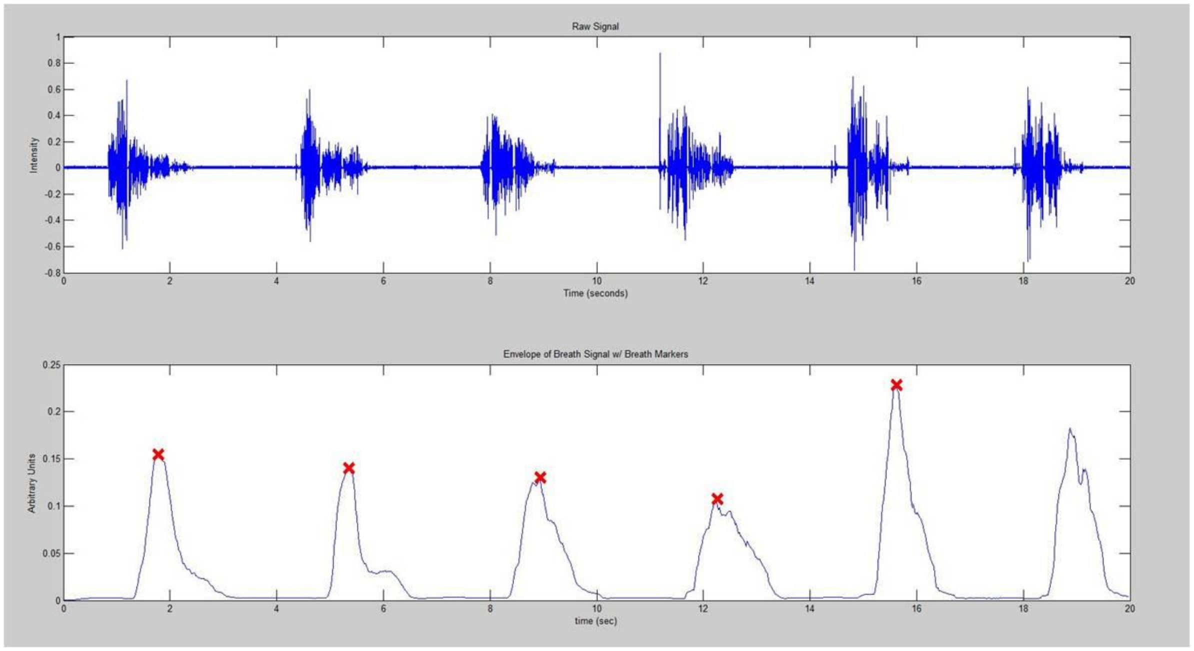

Figure 1 Matlab breath detector - Credit Ricky Hennessy

Hardware

The main hardware for the project is the ATLYS project board from Digilent, as seen in Figure 2. This board features a Xilinx Spartan 6 LX45 FPGA, of special use to the project was the chips DSP48A1 slices which allows for efficient implementation of digital filters. The ATLYS board also features an AC'97 codec, National Semiconductor LM4550, which ideally would be used to sample incoming audio. It has a USB port used to power the USB pre-amp for the microphone system. Primary power is provided by a 20 watt AC to DC converter. The board does get warm so ventilation needs to be provided for in the packaging.

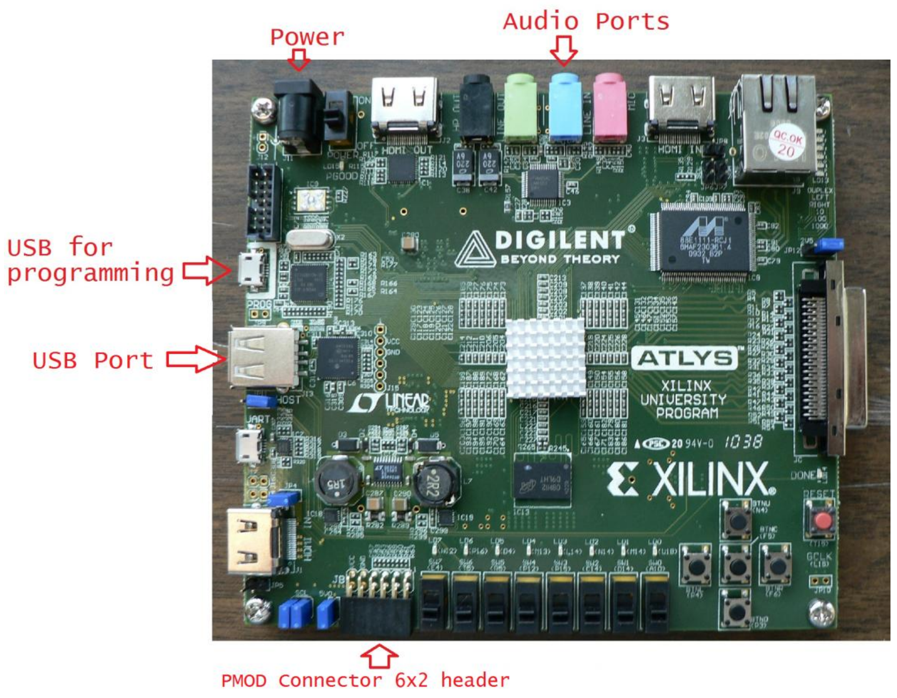

Figure 2 Digilent ATLYS board with ports marked

Due to difficulties in working with the AC'97 codec a separate analog to digital converter (ADC) was used, Digilent's PMOD AD1. This produces a 12 bit value representing the voltage read between 0 and 3.3 v (the voltage applied to the module). In addition the PMOD DA2 module, a 12 bit digital to analog converter (DAC), was used for testing and debugging purposes.

Both of these devices run off 0 to 3.3 v provided from the board, however sound from a microphone has both positive and negative voltages. A level shifter was needed to shift the microphone output to all positive voltages so it could be read by the ADC, the circuit used can be seen in Figure 3 below. This simple circuit shifts the incoming voltage up by the reference voltage, and provides a gain of 2 to amplify the signal. Because we have only positive supply voltage an op-amp capable of running without a negative voltage rail was required, an LM324 was used.

This circuit also works to protect the analog converter, which cannot accept voltages outside 0 to 3.3 volts, by limiting output voltage from 0 to about 2 volts.

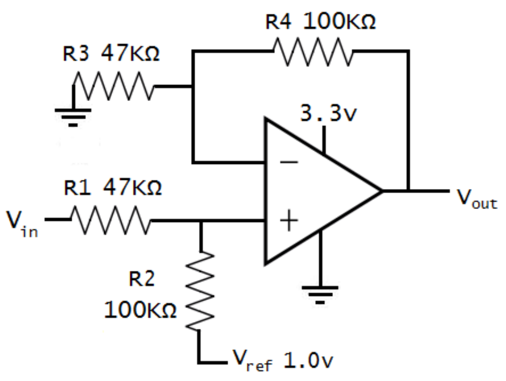

Figure 3 Level shift circuit for microphone

Microphone

Several microphones were used in the course of the project and any microphone should work given sufficient sensitivity and relatively low signal noise, though some calibration might be necessary. The final targeted microphone was a directional microphone from Audio Technica, the AT803 specifically, it does require a pre-amp and the ARTcessoires USB Dual Pre was used. In other testing a parabolic dish was used to help limit surrounding noise and may be helpful in nosier environments or to increase distance between patient and microphone.

VHDL Filter

A filter too complex for C to run in real time was desired but greater flexibility than a hardware filter was desired, as a compromise a VHDL filter was used. The first stage is a band pass filter for the 300 to 800 Hz range, as Figure 4 shows this is where there is clearly information sufficient to detect breathing but reduces the influence of external noise.

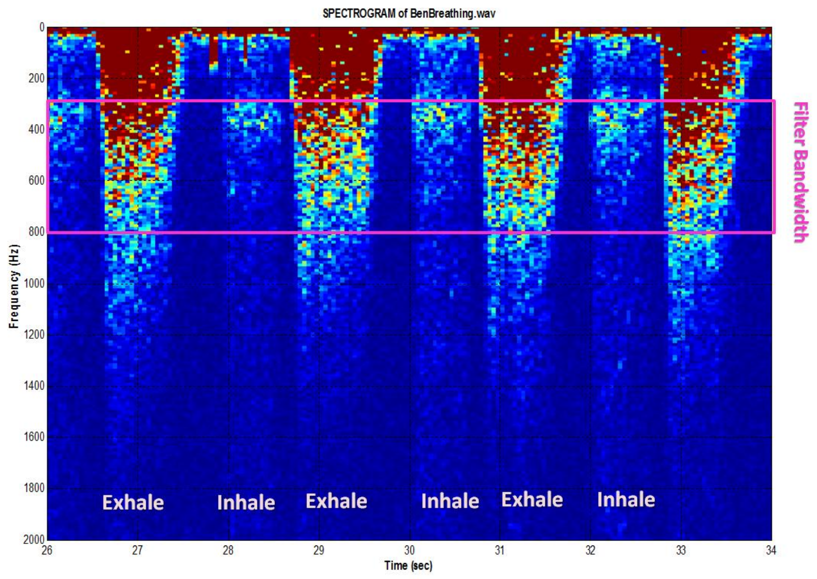

Figure 4 Spectrogram of breathing – Credit: Dr. Pilkington

The VHDL filter also provides the envelope of the signal rather than just a filtered waveform. This allows for simple peak detection to be done with software. The envelope of a modulated sine wave can be seen clearly in yellow in Figure 5.

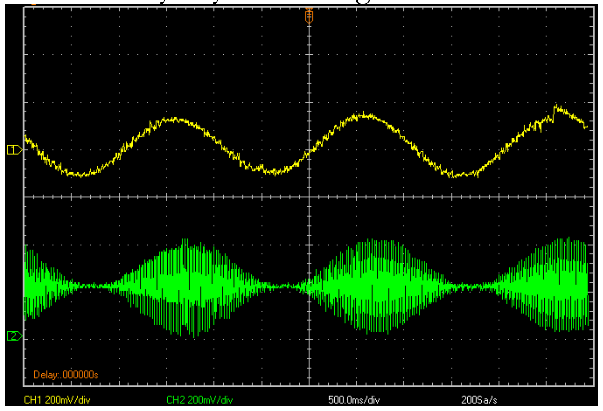

Figure 5 Envelope filtering: Green is raw signal, yellow is the resulting envelope

Software

The software part of the system is to be done using MicroBlaze. This is a microprocessor that is run on the FPGA and programmed using the Xilinx Embedded Develpment Kit. This is programmed using C with some slight modifications due to a more limited environment than a general computer system. MicroBlaze interacts with the hardware which is synthesized in the FPGA through a memory mapped interface.

Figure 6 provides an overview for the software of the system. Samples are read when an interrupt occurs, set at 8 kHz, which are processed through the VHDL filter to find the envelope, peaks are detected and a timer is updated. When the warning timer reaches the set limit for an apnea event an alarm is sounded.

The ADC uses SPI which is handled by a hardware module and controlled by a function in software. The interrupts are based off a timer that counts from the 66 MHz main clock cycle to provide an 8 kHz interrupt.

The filter outputs samples at a rate 500 times less than it is given samples. This means that we only need to deal with samples every 500th interrupt or 16 times per second. The peak detection is observed by having an array of samples, when the middle of that sample period is the same as the maximum of the array then a peak has been detected. There is calibration and averaging of these peaks to ensure that no peaks below the level of breathing are falsely detected as breaths.

A record of time between breaths is kept and as this reaches various lengths appropriate warnings are issued, with an audible alarm in the case of an apnea event.

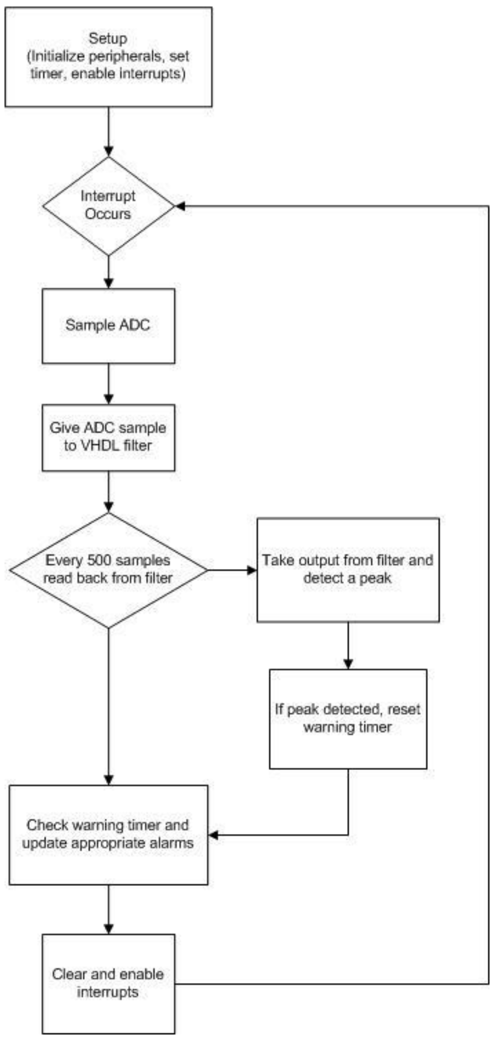
Setup
(Initialize peripherals, set timer, enable interrupts)
Interrupt Occurs
Sample ADC
Give ADC sample to VHDL filter
Every 500 samples read back from filter
Take output from filter and detect a peak
If peak detected, reset warning timer
Check warning timer and update appropriate alarms
Clear and enable interrupts

Figure 6 Software flowchart

User Interface

The user interface needs to be simple and self explanatory. An absolute minimum of features are a power light to indicate that the system has power, an audible warning and a reset button to silence the alarm once sounded. Other lights might be used such as a light indicating that the system is working properly. Another possible feature is a series of LEDs that show how long since the last breath that would start to flash as an apnea event is approached. A method to set the time between breaths considered an apnea event, typically this time is 20 seconds. A volume select to change the level of the audible warning might be another interface option.

IV. Implementation

VHDL Filter

The audio filter was developed in Simulink and exported as VHDL. This caused a number of problems during implementation. The first of which is that on a general purpose computer floating point numbers are perfectly reasonable when dealing with audio signals. However this is not supported in VHDL where floating point hardware requires impractically large circuits. Because of this an integer filter had to be created, which caused problems with overflow. Ricky sorted out the filter problems and produced a filter that would work with 32-bit integers, which were to be supplied by the AC'97 codec.

However the filter was expecting to run at the same frequency it was expecting data samples, 8 kHz. The main MircoBlaze clock runs at 66.666 MHz, a step down from the 100 MHz clock signal provided by the board, so a slower clock would need to be provided directly to the filter. The filter also required a wrapper module so that it could work with the memory mapped IO that MicroBlaze usually worked with.

Luckily the Xilinx development software provides a peripheral creation tool that can easily allow for custom coded VHDL to be attached to the MicroBlaze memory bus, these are called pcores. The system clock divider was setup to produce an 8 MHz signal, since the lowest possible clock speed it can produce is 1 MHz. An additional port line was added to the peripheral wrapper and a simple clock divider was added to the VHDL filter to produce the desired 8 kHz signal.

The extra clock line was added in the port definition for the memory bus wrapper, user_logic.vhd, as well as in the projects .mpd file where all external ports are defined for the system interface in the development software. The extra source VHDL files that were needed for the filter were added in the pcore vhdl folder and entries were made into the projects .pao and .prj files. These files definitions were added into the definition for the pcores library. Additionally the project needed to be modified to not halt upon finding a part of the hardware running at a significant fraction of the main system clock, our 8 kHz filter. This was achieved by selecting the project option for timing closure failures to no longer be considered errors.

When the AC'97 codec was dropped as an input method the precision of the incoming data was reduced to 12 bits. The filter was redesigned to take in 16 bit numbers and outputting 32 bit numbers that had been filtered for the desired frequency range and the envelope of the signal. The MicroBlaze system is a big endian microcontroller and thus the 16 bits for the filter input needed to be pulled from the top half of the incoming 32 bit register on the memory bus.

Initially this filter did not work, the only time it appeared to work was when the aliasing of the 8 kHz sample rate happened to create the appearance of proper envelope detection. This was due to input being in the all positive range for a filter designed for a bipolar filter. The solution was to subtract the known offset of the signal and feed the now bipolar signal into the filter.

Level Shift Circuit

The level shift circuit seen previously in Figure 3 was built on a breadboard and tested with lab equipment to ensure the desired operation. The only problem here was that an op-amp that allowed for only a ground rail and a positive rail was required. The next stage in testing was to make sure that the circuit could be powered from the voltages on the ATLYS board and didn’t cause any problem with the operation of the other circuits.

The reference voltage is supplied from the DAC PMOD, this gives us precise control over the offset and also the software decides this voltage which can be supplied to the filter so that the bipolar nature of the audio signal can be restored before filtering.

Signal Input and Output

Initially it was desired to utilize the AC'97 audio codec on the board. The first attempt at doing so was meant to pull an audio sample and write it back out, this worked with little effort. However once I attempted to manipulate this sample the output signal did not change. Unfortunately the loopback functionality is built into the AC'97 codec chip. Many attempts to control the chip in the desired way were unsuccessful, it was decided that alternative methods of input and output would be more efficient for time usage.

The PMOD connector on the ATLYS board works with a variety of add-ons called PMODs. Two of these add-ons are the PMOD AD1 and the PMOD DA2. The AD1 is a 12 bit analog to digital conversion chip and the DA2 is a 12 bit digital to analog converter. With these chips it was possible to sample an audio signal at the 8 kHz rate desired at 12 bits of precision. These 12 bits are the unsigned value above ground and max out at the positive voltage rail, 3.3 volts. Each of these chips has two channels for analog input or digital output respectively and communicate through the SPI protocol. The SPI protocol is supported in hardware with a Xilinx module and controlled in software through command registers.

The DA2 module provides two output channels, however this is achieved through the use of two separate DAC chips. Because of this the SPI connection is a bit odd, instead of having a common data channel, clock channel and two select lines it has two data lines and a common select channel. This requires two separate SPI modules to provide the two data channels to communicate with it, the OR signal of the two DAC SPI channels and the AND signal of their select lines (active low connections). Additionally because the select line is common they cannot be updated independently and must be updated as a pair, this is non-ideal for our purposes and requires an inelegant work around in the software. Each SPI core cannot be run at exactly the same time, the appropriate commands are given right after each other in the hope that there is not sufficient time differences to ruin either setting. The second channel of the DAC is used to provide the reference voltage of the level shift circuit, the constant updating produces additional noise to this line. This possibly adds extra interference to the audio signal that is being read into the system and some sort of filtering may become necessary.

MicroBlaze

Microblaze is what is called a soft core processor, unlike other micro controllers it is only implemented in VHDL. This is provided by Xilinx as part of their development software as well as many other hardware components. This project utilized the basic components of Microblaze, the processor, a data, instruction and peripheral busses and a memory block for the processor. Other peripherals included: three SPI cores used to talk to the ADC and DAC; a clock divider to create the desired clocks; a timer to generate the 8 kHz interrupts; and several general purpose IO modules for the LEDs, and switches on the board.

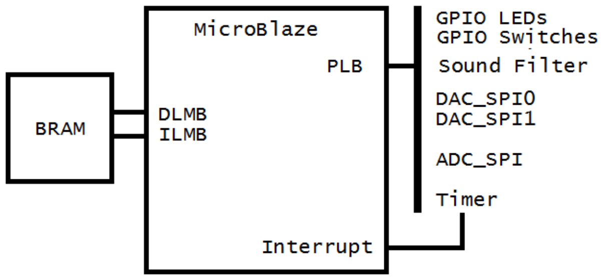

Figure 7 MicroBlaze system diagram

Software

The software is responsible for regulating the hardware components of the MicroBlaze system as well as performing some of the calculations. The code is not particularly complex, the main source file comes in under 240 lines and there aren't many tricky shortcuts. Most of the heavy lifting was implemented in the VHDL filter, which was the desired outcome of the project. Of course there were problems along the way, mostly dealing with improper control of the hardware components such as the problems with input and output modules. The software could allow for greater characterization of the breath signal and a finer tuned peak detection system than a VHDL approach. This would allow for greater information than simply time between breathing and a reduction of false positives, but nothing beyond simple peak detection has yet been implemented.

User Interface

The user interface consists of the LEDs, switches and audio codec. The LEDs provide a visual display of the timer which will start to fill as the time approaches an apnea event. Each LED represents and eighth of the time to apnea event and as the last 3 LEDs become lit they will start to blink. When an apnea event occurs the LEDs will continue to flash and an alarm will sound. While the audio codec was unhelpful in data acquisition various beeps can be sent through the line out, this could be hooked to an external speaker for the sound alarms. Switch 8 serves to silence the alarm and reset it, when this switch is on the LED above it is lit up to indicate that the system is in silent mode.

V. System integration and testing

The system was rather closely knit and thus integration was done over the course of development rather than entirely at the end. An example would be the input and output, these are related systems and can be used to verify each other. The input and output systems are also crucial to the development and testing of the VHDL filter since signals needed to be given as input and verified on an oscilloscope through the output device.

Once the input and output systems were in place the VHDL filter was inserted into the design, this also required the interrupt system to be in place to ensure the 8 kHz input. This was when the filter was discovered to require a bipolar signal instead of the offset input as well as an 8 kHz clock instead of the main system clock of 66.666 MHz.

The filter was tested using an AM modulated sine wave produced from a function generator. This produced a configurable, constant, and stronger signal than audio while trouble shooting and tuning the filter. The envelope detection could be visually verified as in Figure 5 as well as the filter roll off around the desired range. Once the filter was working acceptably the peak detection algorithm could be finished. Once peak detection of the envelope was working a counter was added so that the interval between breaths could be monitored for the user interface operation and the detection of apnea events. The LED output and switch behavior was refined and the user interface was mostly complete.

At the same time the audio signal was being worked on. First the microphone setup was tested to ensure operation as it was from a previous iteration of the project, then the shifter circuit was produced and tested. One of the more crucial stages for the audio system was ensuring that the required USB pre-amp and shifter circuit could be powered from the board to reduce the need for further external power sources.

After both the detection and audio systems were working system testing could begin. Unfortunately this was extremely late in the project and an insufficient amount of testing and calibration was possible. The system works with audio signals but a noticeable amount of false positives were present, this is probably a problem with the peak detection and can be fixed by averaging the past few maximums and not allowing a maximum significantly above or below this average. This means that once breaths are being detected the filter will become accustom to the breathing levels and minimize false positives and negatives.

Operation of the device requires setting up the microphone at a reasonable distance from the patient and connecting all the hardware components together. After the system is together the power switch can be turned on and after system initialization should begin detecting breaths. Switch 8 can be used to silence the system and the LEDs will provide information about the current operation of the device.

VI. Conclusion

I believe this project was rather successful, it was a difficult process and development was quite a bit slower than desired but the final product is rather close to the designed system. It could probably require a bit more fit and finish but the system is at least workable. An incoming signal is filtered for the desired range an envelope is created, peaks are detected and a timer is kept.

Mainly the problem is in that there has been no testing on sound from actual infants and that might require a change in the VHDL filter and audio setups. There is also a noticeable presence of false positives which is unacceptable for a final device. This is probably due to a lack of testing and calibration and could be remedied with more testing. The peak detection currently has a minimum level for peaks and this could be raised to reduce false positives but could also introduce a number of false positives and would vary depending upon audio input parameters. A maximum level could also be added to reduce the impact of sudden noises not currently being filtered out. Averaging a number of past maximums to rule out infrequent behavior would also be helpful.

The project could be simplified in two ways, first the proper usage of the AC'97 codec for data acquisition. This removes the need for the shifter circuit as well as some of the workarounds in the filter. If this cannot be done then a cheaper board, without the audio codec, is recommended. Initially size of the design on the FPGA was a concern, but as implemented is not a current issue. The speed of the system is also not a great concern in the current implementation which works at 8 kHz and doesn't require a great deal of computation. Of course if further analysis of the breathing is desired then these might become relevant concerns, though that would also require an overhaul of the filtering system or secondary processing of the initial samples.

Works Cited

[1] "Infant Sleep Apnea." Internet: http://www.stanford.edu/~dement/infantapnea.html, March 24, 1999 [April 12, 2011]

[2] "Healthdyne 970 Smart Apnea Monitor." Internet: http://www.dremed.com/irsrental/product_info.php/cPath/412_413/products_id/9336, April 12, 2011 [April 12, 2011]

[3] "Apnea, Sudden Infant Death Syndrome, and Home Monitoring" Internet: http://pediatrics.aappublications.org/content/111/4/914.full, April 1, 2003 [June 12, 2011]

[4] Sandra Reichert et al. "Analysis of Respiratory Sounds: State of the Art" Internet: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2990233/, May 16, 2008 [June 12, 2011]

VII. Appendices

Appendix A: Bill of Materials

- Digilent ATLYS development board
  - 20 watt AC to DC power supply
  - micro USB programming cable
- Digilent PMOD AD1 12 bit analog to digital converter
- Digilent PMOD DA2 12 bit digital to analog converter
- Audio Technica AT803
  - XLR microphone cable
- ARTcessoires USB Dual Pre - USB pre-amp
  - USB B cable for power
- Level Shift Circuit
  - LM324 Op-Amp
  - 2 47 kΩ resistor
  - 2 100 kΩ resistor

Appendix B: infant_apnea.c

```
/* Infant Sleep Apnea project
 * Author: Brian Berg
 *
 * This program takes in sound, filters it, detects it's peaks for breathing and
 * provides appropriate alarms when no peak is detected within a certain timeframe
 */

 /* libraries we need */
#include <stdio.h>
#include "xparameters.h"
#include "xgpio.h"
#include "xio.h"

/* perpherial defines to make addressing easier to read */
#define LEDS_OUT       XPAR_LEDS_8BITS_BASEADDR
#define SWITCHES_IN    XPAR_DIP_SWITCHES_8BITS_BASEADDR
#define DAC0_BASE      XPAR_SPI_DAC0_BASEADDR
#define DAC1_BASE      XPAR_SPI_DAC1_BASEADDR
#define ADC_BASE       XPAR_SPI_ADC_BASEADDR
// filter connections
#define FILTER         XPAR_CUSTOM_SOUND_FILTER_0_BASEADDR
#define FILTER_IN      FILTER
#define FILTER_OUT     FILTER + 0x4
#define FILTER_REF     FILTER + 0x8

// our 66,666,700Hz system clock
#define SYS_CLK_SPEED  XPAR_TIMER_CLOCK_FREQ_HZ
#define TIMER_BASE     XPAR_TIMER_BASEADDR

#define TARGET_FREQ      8000 //target rate for sampling
#define REDUCTION        500  //how many times we are down sampling
#define SAMPLES_PER_SEC  16   // TARGET_FREQ/REDUCTION (8K/500)=16
#define SAMPLE_TIME      10   // when in count we pull a sample from the filter
#define SAMPLE_WINDOW    16   // How many samples our window
#define APNEA_TIME       320  // SAMPLES_PER_SEC * 20 seconds
#define MIN_MAX_LEVEL    0xFFFFF //avoid false positives


/* define where each switch is in the 8 bit value */
#define SWITCH1 1
#define SWITCH2 2
#define SWITCH3 4
#define SWITCH4 8
#define SWITCH5 16
#define SWITCH6 32
#define SWITCH7 64
#define SWITCH8 128

/* Shifter reference voltage for offset of audio signal */
#define SHIFTER_REF 1250
```

```
/* function prototypes */
void isr() __attribute__ ((interrupt_handler));
void set_timer();
void frequency_check();
void dac_out(int value);
void breath_detect();
int find_max(int *array, int size);

int main(){
   /* setup phase, get everything set */
   // set LEDS to output mode
   XIo_Out32(LEDS_OUT + 0x04 , 0x00);
   // show something on the LEDS so we know we got this far
   XIo_Out32(LEDS_OUT, 0xFF);

   //set out shifter reference voltage and tell the filter what it is
   spi_out(DAC0_BASE, SHIFTER_REF);
   XIo_Out32(FILTER_REF, SHIFTER_REF);

   set_timer(); //get the interrupt timer ready
   XIo_Out32(LEDS_OUT, 0x0F); //new LED stage

   /* program starts now! */
   microblaze_enable_interrupts();
}

/* used to verify what frequency our interrupts are working at
 * will output a 50% duty cycle square wave between 0 and 3.3v
 * at half the frequency of the interrupts */
void frequency_check(){
   static int freq_check = 0, freq_check2 = 0;

   dac_out(freq_check);
   if(freq_check){
      freq_check = 0;
      freq_check2 = 0xFFF;
   }
   else{
      freq_check = 0xFFF;
      freq_check2 = 0;
   }
}

/* any time we update the main output channel we also need to send data
 * to the channel handling the shifter circuit reference voltage due to the
 * construction of the DAC
 * This is a handy abstraction */
void dac_out(int value){
   spi_out2(DAC0_BASE, value, DAC1_BASE, SHIFTER_REF);
}

/* load our timer with the appropriate count */
```

```
void set_timer(){
    //load timer's load register, subtract two from count for reload
    XIo_Out32(TIMER_BASE + 4, (SYS_CLK_SPEED/TARGET_FREQ) - 2);
    //copy load reg to timer reg
    XIo_Out32(TIMER_BASE, 0x20);
    /* config timer to count down, auto reload,
      * enable ints, gen mode, and enable timer*/
    XIo_Out32(TIMER_BASE, 0XD2);
}

/* main calculation function */
void breath_detect(){
    int sample, spi_sent, switches, filter_out, shift, leds;
    static int count = 0, breath_count = 0, blink;
    static int sample_spot = SAMPLE_WINDOW, mid_spot = (SAMPLE_WINDOW/2)-1;
    static int samples[SAMPLE_WINDOW];

    sample = spi_in(ADC_BASE); //obtain audio sample
    switches = XIo_In32(SWITCHES_IN);
    spi_sent = 0; //reset SPI lock, or if set to 1 disable SPI output from this function

    //allows to check for proper input
    if((switches & SWITCH1) && !spi_sent ){
            dac_out(sample);
            spi_sent = 1;
    }

    //hand the sample to the filter
    XIo_Out32(FILTER_IN, sample);

    //the filter downsamples, so count till it's good to go
    count = (count+1)%REDUCTION;

    if(count == SAMPLE_TIME){
        filter_out = XIo_In32(FILTER_OUT);

        //see what the output of the filter is
        if((switches & SWITCH2) && !spi_sent){
            shift = switches >> 2;
            dac_out(filter_out>>shift);
            spi_sent = 1;
        }

        //record the sample
        sample_spot = (sample_spot+1)%SAMPLE_WINDOW;
        mid_spot = (mid_spot+1)%SAMPLE_WINDOW;

        samples[sample_spot] = filter_out;

        //peak detect
        if(peak_detect(samples, SAMPLE_WINDOW, mid_spot)){
            breath_count = 0;
            if(!spi_sent){
                dac_out(0xFFF); //shows us when
            }
        }
        else{
            //no breath, add to count but don't need to go over APNEA_TIME
            if(breath_count < APNEA_TIME){
                breath_count++;
            }
            if(!spi_sent){
                dac_out(0);
            }
        }

        /* this displays a nice count on the display
         * no binary count up just a light up one LED after the other */
        shift = 8 - breath_count/(APNEA_TIME/8);
        leds = 0xFF>>shift;
        /* once we're above a certain point start to flash the LEDs */
        if(leds & 0x20){
            if(blink){
                blink = 0;
            }
            else{
                leds = 0;
                blink = 1;
            }
        }

        if((switches & SWITCH8)){
            //this is the reset/disable switch
            breath_count = 0;
            leds = 0x80; //let the person know it's disabled
        }
        XIo_Out32(LEDS_OUT, leds); //output whatever has been decided
    }

}

/* detect a peak in the given window */
int peak_detect(int *samples, int size, int mid_spot){
    int max;

    max = find_max(samples, size);

    if(max < MIN_MAX_LEVEL){
        return 0;
    }

    if(samples[mid_spot] == max){
        return 1;
    }
    return 0;
}

/* find the maximum of the array */
int find_max(int *array, int size){
    int i, max = 0; //not looking for negative maximums

    for(i = 0; i < size; i++){
        if(array[i] > max){
            max = array[i];
        }
    }
    return max;
}

/* this function happens each time an interrupt from the timer occurs
 * which should be at a frequency of TARGET_FREQ */
void isr(){
    int state;
    int sample;

    breath_detect();

    /* clear interrupt, needs to happen for this loop to work*/
    //grab the timer state
    state = XIo_In32(XPAR_TIMER_BASEADDR);
    //put it back, thereby resetting the interrupt
    XIo_Out32(XPAR_TIMER_BASEADDR, state);
}
```

Appendix C: spi.c

```
/* Code to talk to the Xilinx SPI core
 * Created for CPE 329 Spring 2010
 * Modified for Senior Project Spring 2011 - Brian Berg
 *
 * Brian Berg and Andrew Carrillo
 *
 * Created with reference to code from Dr. John Oliver
 */

#include "xparameters.h"
#include "xio.h"

#define SPICR  0x60
#define SPISR  0x64
#define SPIDTR 0x68
#define SPIDRR 0x6C
#define SPISSR 0x70
#define IPISR  0x20

#define INHIBIT_DISABLE  0x194
#define INHIBIT_ENABLE   0x196
#define UNINHIBIT_ENABLE 0x096

int spi_in(int base_address){
   int upper_byte, low_byte;

   //set SPI mode, inhibit master, disable SPI
   XIo_Out32(base_address + SPICR, INHIBIT_DISABLE);
   //Write junk to clear the DTR
   XIo_Out8(base_address + SPIDTR, 0xF); //upper byte first
   XIo_Out32(base_address + SPISSR, 0xFFF); //Turn on ADC
   //set SPI mode, inhibit master, enable SPI
   XIo_Out32(base_address + SPICR, INHIBIT_ENABLE);
   XIo_Out32(base_address + SPISSR, 0); //Enable the ADC
   //Unihibit but leave enabled
   XIo_Out32(base_address + SPICR, UNINHIBIT_ENABLE);
   //Wait till transfer done
   while ( (XIo_In32(base_address + SPISR) & 0x4) == 0);

   XIo_Out32(base_address + SPICR, INHIBIT_ENABLE); //inhibit master
   //Get upper byte from ADC
  upper_byte = (XIo_In32(base_address + SPIDRR) & 0x000000FF);
   //write dummy data to
   XIo_Out8(base_address + SPIDTR, 0xF); //lower byte now
   XIo_Out32(base_address + SPICR, UNINHIBIT_ENABLE); //uninhibit master
   //wait till transfer is done
   while ( (XIo_In32(base_address + SPISR) & 0x4) == 0);
   XIo_Out32(base_address + SPICR, INHIBIT_ENABLE); //inhibit master
   //Get lower byte from ADC
   low_byte = (XIo_In32(base_address + SPIDRR) & 0x000000FF);
   XIo_Out32(base_address + SPISSR, 0xFFF); //turn off the ADC
```

```
    //Set mode, inhibit Master, disable SPI
    XIo_Out32(base_address + SPICR, INHIBIT_DISABLE);

   return (upper_byte << 8) | low_byte;
}

/* this is rather inelegant, but becuase the DA2 has shared clock and select lines
 * both channels are changed even if it's not meant to be. So we write to both channels at the same time
 */
void spi_out2(int base, int value, int base2, int value2){
    int first_byte, second_byte, DTR_empty;
    int first_byte2, second_byte2;

    value = value & 0x0FFF; //make sure its in range
    first_byte = value >> 8; //get the high order byte
    second_byte = (value & 0xFF); //get the low order byte
            value2 = value2 & 0x0FFF;
    first_byte2 = value2 >>8;
    second_byte2 = (value2 & 0xFF);

    //set SPI mode, inhibit master, disable SPI
    XIo_Out32(base + SPICR, INHIBIT_DISABLE);
    XIo_Out32(base2 + SPICR, INHIBIT_DISABLE);
    //write MSByte to SPIDTR
    XIo_Out8(base + SPIDTR, first_byte);
    XIo_Out8(base2 + SPIDTR, first_byte2);
    //negate SS-bar (write 1s to SPISSR)
    XIo_Out32(base + SPISSR, 0xFFF);
    XIo_Out32(base2 + SPISSR, 0xFFF);
    //enable SPI
    XIo_Out32(base + SPICR, INHIBIT_ENABLE);
    XIo_Out32(base2 + SPICR, INHIBIT_ENABLE);
    //assert SS-bar (write 0s to SPISSR)
    XIo_Out32(base + SPISSR, 0);
    XIo_Out32(base2 + SPISSR, 0);
    //un-inhibit master
    XIo_Out32(base + SPICR, UNINHIBIT_ENABLE);
    XIo_Out32(base2 + SPICR, UNINHIBIT_ENABLE);
    //wait for DTR-empty flag to be asserted
    while ( (XIo_In32(base + SPISR) & 0x04) == 0);
    //inhibit master
    XIo_Out32(base + SPICR, INHIBIT_ENABLE);
    XIo_Out32(base2 + SPICR, INHIBIT_ENABLE);
    //clear DTR-empty flag
    XIo_Out32(base + IPISR, 0xFF);
    XIo_Out32(base2 + IPISR, 0xFF);
    //write LSByte to SPIDTR
    XIo_Out8(base + SPIDTR, second_byte);
    XIo_Out8(base2 + SPIDTR, second_byte2);
    //un-inhibit master
    XIo_Out32(base + SPICR, UNINHIBIT_ENABLE);
    XIo_Out32(base2 + SPICR, UNINHIBIT_ENABLE);
    //wait for DTR-empty flag to be asserted
    while ( (XIo_In32(base + SPISR) & 0x04) == 0);
    //inhibit master
    XIo_Out32(base + SPICR, INHIBIT_ENABLE);
    XIo_Out32(base2 + SPICR, INHIBIT_ENABLE);
    //clear DTR-empty flag
    XIo_Out32(base + IPISR, 0xFF);
    XIo_Out32(base2 + IPISR, 0xFF);
    DTR_empty = 0;
    //negate SS-bar (write 1s to SPISSR)
    XIo_Out32(base + SPISSR, 0xFFF);
    XIo_Out32(base2 + SPISSR, 0xFFF);
    //inhibit master, disable SPI
    XIo_Out32(base + SPICR, INHIBIT_DISABLE);
    XIo_Out32(base2 + SPICR, INHIBIT_DISABLE);
}

void spi_out(int base, int value){
    int first_byte, second_byte, DTR_empty;

    value = value & 0x0FFF; //make sure its in range
    first_byte = value >> 8; //get the high order byte
    second_byte = (value & 0xFF); //get the low order byte

    //set SPI mode, inhibit master, disable SPI
    XIo_Out32(base + SPICR, INHIBIT_DISABLE);
    //write MSByte to SPIDTR
    XIo_Out8(base + SPIDTR, first_byte);
    //negate SS-bar (write 1s to SPISSR)
    XIo_Out32(base + SPISSR, 0xFFF);
    XIo_Out32(base + SPICR, INHIBIT_ENABLE); //enable SPI
    //assert SS-bar (write 0s to SPISSR)
    XIo_Out32(base + SPISSR, 0);
    XIo_Out32(base + SPICR, UNINHIBIT_ENABLE); //un-inhibit master
    //wait for DTR-empty flag to be asserted
    while ( (XIo_In32(base + SPISR) & 0x04) == 0);
    XIo_Out32(base + SPICR, INHIBIT_ENABLE); //inhibit master
    //clear DTR-empty flag
    XIo_Out32(base + IPISR, 0xFF);
    XIo_Out8(base + SPIDTR, second_byte); //write LSByte to SPIDTR
    XIo_Out32(base + SPICR, UNINHIBIT_ENABLE); //un-inhibit master
    //wait for DTR-empty flag to be asserted
    while ( (XIo_In32(base + SPISR) & 0x04) == 0);
    XIo_Out32(base + SPICR, INHIBIT_ENABLE); //inhibit master
    XIo_Out32(base + IPISR, 0xFF); //clear DTR-empty flag
    DTR_empty = 0;
    XIo_Out32(base + SPISSR, 0xFFF); //negate SS-bar (write 1s to SPISSR)
    //inhibit master, disable SPI
    XIo_Out32(base + SPICR, INHIBIT_DISABLE);
}
```

Appendix D: Matlab Breath detection

RealTimeBreathDetect.m - Ricky Hennessy

```
% BreathDetect.m
% Ricky Hennessy
% 1/6/2010

close all, clear all

%% Variables
low_stop = 300;
high_stop = 1000;
WindowEnvelope = 0.1;    % Length of envelope averaging filter window
(seconds)
MaximaWindow = 2.4;      % Length of window to find local maxima
(seconds)
DownSamp = 200;          % Frequency to downsampe envelope to (Hz)
Threshold = 0.2;         % Percentage of mean maxima value that a
breath must be above
Fs = 8000;               % Sampling Frequency
nbits = 8;               % Bits of Precision When Sampling
WindowTime = 5;          % Length of recorded time processed in each
batch (seconds)

%% Initialize
BreathCountTotal = 0;
y = audiorecorder(Fs,nbits,1);
meanbreath = 0;

%% Create Bandpass Filter
F=[(low_stop-100) low_stop high_stop (high_stop+100)];   % band limits
A=[0 1 0];                                 % band type: 0='stop', 1='pass'
dev=[0.0001 10^(0.1/20)-1 0.0001];              % ripple/attenuation spec
[M,Wn,beta,typ]= kaiserord(F,A,dev,Fs);          % window parameters
b = fir1(M,Wn,typ,kaiser(M+1,beta),'noscale'); % filter design

%% Initialize
loopcount = 0;
EXIT = 1;
BreathTotal = [];

%% Initial Recoding
signal = zeros(1,Fs*WindowTime);
record(y)

while EXIT == 1
    %% Counter
    loopcount = loopcount + 1;

    %% Filter Signal
    signal_f = fftfilt(b,signal);
    %% Find Envelope
    signal_h = hilbert(signal_f); % Hilbert Transform
    envelope = sqrt(signal_h.*conj(signal_h));
    envelope_f =
filter(ones(1,round(Fs*WindowEnvelope))/round(Fs*.1),1,envelope);
    envelope_fDS = downsample(envelope_f,round(Fs*(1/DownSamp)));

    %% Find Local Maxima
    windowsize = DownSamp * MaximaWindow / 4;
    BreathCount = 0;
    Breath = [];
    for ix = 1:length(envelope_fDS) - windowsize
        maxima = max(envelope_fDS(ix:ix+windowsize));
        if loopcount < 2    % Ignore first 2 loops
        elseif loopcount < 100    % Use next 100 loops to get breath
threshold
            if maxima == envelope_fDS(ix+windowsize/2)
                BreathCount = BreathCount + 1;
                Breath(BreathCount) = ix + windowsize/2;
            end
        else    % Begin using and updating threshold
            if (maxima == envelope_fDS(ix+windowsize/2)) && maxima >
(Threshold * meanbreath)
                BreathCount = BreathCount + 1;
                Breath(BreathCount) = ix + windowsize/2;
            end
        end
    end
    BreathTotal = cat(2,BreathTotal,envelope_fDS(Breath));
    meanbreath = mean(BreathTotal);
    if isempty(Breath) && loopcount > 100 == 1
        for i = 1:10
            beep
            pause(.1)
        end
        EXIT = 0;
    end
    %% Keep Recording
    stop(y)
    signal_new = getaudiodata(y,'double');
    record(y)
    signal(1:end-length(signal_new)) =
signal(1+length(signal_new):end);
    signal(end-length(signal_new)+1:end) = signal_new;
    clear signal_new

    %% Plot
    figure(1)
    subplot(2,1,1)
    signalplot =  signal;
    time1 = (0:length(signalplot)-1) / (Fs/4);
    refresh
    plot(time1,signalplot)
    title('Raw Signal')
    xlabel('Time (seconds)')
    ylabel('Intensity')

    subplot(2,1,2)
    envelopeplot = envelope_fDS;
    time = (0:length(envelopeplot)-1) / (DownSamp/4);
    plot(time,envelopeplot)
    hold on

plot(Breath/(DownSamp/4),envelope_fDS(Breath),'rx','MarkerSize',16,'li
newidth',4)
    title('Envelope of Breath Signal w/ Breath Markers')
    xlabel('time (sec)')
    ylabel('Arbitrary Units')
    hold off
end
```

Appendix E: VHDL filter code

user_logic.vhd

```
------------------------------------------------------------------------------
-- user_logic.vhd - entity/architecture pair
------------------------------------------------------------------------------
--
-- ***************************************************************************
-- ** Copyright (c) 1995-2010 Xilinx, Inc.  All rights reserved.            **
-- **                                                                       **
-- ** Xilinx, Inc.                                                          **
-- ** XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"         **
-- ** AS A COURTESY TO YOU, SOLELY FOR USE IN DEVELOPING PROGRAMS AND       **
-- ** SOLUTIONS FOR XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE,        **
-- ** OR INFORMATION AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE,        **
-- ** APPLICATION OR STANDARD, XILINX IS MAKING NO REPRESENTATION           **
-- ** THAT THIS IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,     **
-- ** AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE      **
-- ** FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY              **
-- ** WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE               **
-- ** IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR        **
-- ** REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF       **
-- ** INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS       **
-- ** FOR A PARTICULAR PURPOSE.                                             **
-- **                                                                       **
-- ***************************************************************************
--
------------------------------------------------------------------------------
-- Filename:          user_logic.vhd
-- Version:           1.00.a
-- Description:       User logic.
-- Date:              Wed Mar 09 23:46:09 2011 (by Create and Import Peripheral Wizard)
-- VHDL Standard:     VHDL'93
------------------------------------------------------------------------------
-- Naming Conventions:
--   active low signals:                    "*_n"
--   clock signals:                         "clk", "clk_div#", "clk_#x"
--   reset signals:                         "rst", "rst_n"
--   generics:                              "C_*"
--   user defined types:                    "*_TYPE"
--   state machine next state:              "*_ns"
--   state machine current state:           "*_cs"
--   combinatorial signals:                 "*_com"
--   pipelined or register delay signals:   "*_d#"
--   counter signals:                       "*cnt*"
--   clock enable signals:                  "*_ce"
--   internal version of output port:       "*_i"
--   device pins:                           "*_pin"
--   ports:                                 "- Names begin with Uppercase"
--   processes:                             "*_PROCESS"
--   component instantiations:              "<ENTITY_>I_<#|FUNC>"
------------------------------------------------------------------------------

-- DO NOT EDIT BELOW THIS LINE --------------------
library ieee;
USE IEEE.numeric_std.ALL;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library proc_common_v3_00_a;
use proc_common_v3_00_a.proc_common_pkg.all;

-- DO NOT EDIT ABOVE THIS LINE --------------------

--USER libraries added here

------------------------------------------------------------------------------
-- Entity section
------------------------------------------------------------------------------
-- Definition of Generics:
--   C_SLV_DWIDTH                 -- Slave interface data bus width
--   C_NUM_REG                    -- Number of software accessible registers
--
-- Definition of Ports:
--   Bus2IP_Clk                   -- Bus to IP clock
--   Bus2IP_Reset                 -- Bus to IP reset
--   Bus2IP_Data                  -- Bus to IP data bus
--   Bus2IP_BE                    -- Bus to IP byte enables
--   Bus2IP_RdCE                  -- Bus to IP read chip enable
--   Bus2IP_WrCE                  -- Bus to IP write chip enable
--   IP2Bus_Data                  -- IP to Bus data bus
--   IP2Bus_RdAck                 -- IP to Bus read transfer acknowledgement
--   IP2Bus_WrAck                 -- IP to Bus write transfer acknowledgement
--   IP2Bus_Error                 -- IP to Bus error response
------------------------------------------------------------------------------

entity user_logic is
  generic
  (
    -- ADD USER GENERICS BELOW THIS LINE ---------------
    --USER generics added here
    -- ADD USER GENERICS ABOVE THIS LINE ---------------

    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol parameters, do not add to or delete
    C_SLV_DWIDTH                   : integer              := 32;
    C_NUM_REG                      : integer              := 3
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
  );
  port
  (
    -- ADD USER PORTS BELOW THIS LINE ------------------
    F_CLK                    : in std_logic;
    -- ADD USER PORTS ABOVE THIS LINE ------------------

    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol ports, do not add to or delete
    Bus2IP_Clk               : in  std_logic;
    Bus2IP_Reset             : in  std_logic;
    Bus2IP_Data              : in  std_logic_vector(0 to C_SLV_DWIDTH-1);
    Bus2IP_BE                : in  std_logic_vector(0 to C_SLV_DWIDTH/8-1);
    Bus2IP_RdCE              : in  std_logic_vector(0 to C_NUM_REG-1);
    Bus2IP_WrCE              : in  std_logic_vector(0 to C_NUM_REG-1);
    IP2Bus_Data              : out std_logic_vector(0 to C_SLV_DWIDTH-1);
    IP2Bus_RdAck             : out std_logic;
    IP2Bus_WrAck             : out std_logic;
    IP2Bus_Error             : out std_logic
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
  );

  attribute SIGIS : string;
  attribute SIGIS of Bus2IP_Clk    : signal is "CLK";
  attribute SIGIS of Bus2IP_Reset  : signal is "RST";

end entity user_logic;

------------------------------------------------------------------------------
-- Architecture section
------------------------------------------------------------------------------

architecture IMP of user_logic is

  --USER signal declarations added here, as needed for user logic
  --**********************************************************************************
  --**********************************************************************************
  COMPONENT envelopeHDL
  PORT( clk                    :   IN    std_logic;
        reset                  :   IN    std_logic;
        clk_enable             :   IN    std_logic;
        In1                    :   IN    std_logic_vector(15 DOWNTO 0);  -- int16
        ce_out                 :   OUT   std_logic;
        Out1                   :   OUT   std_logic_vector(31 DOWNTO 0)  -- sfix32_En19
        );
  END COMPONENT;

  --**********************************************************************************
  --**********************************************************************************
  FOR ALL : envelopeHDL
    USE ENTITY work.envelopeHDL(rtl);

  SIGNAL Out_Filter                    : std_logic_vector(31 downto 0);
  SIGNAL Filter_Signed                 : std_logic_vector(15 downto 0);
  SIGNAL In_Filter                     : std_logic_vector(15 downto 0);
  SIGNAL Test_In                        : std_logic_vector(31 downto 0);


  -- SIGNALS FROM CLOCK DIVIDER
  SIGNAL div_count                 : IEEE.numeric_std.unsigned(8 DOWNTO 0);
  SIGNAL slow_clock                : std_logic;
  --END USER SIGNALS
  --********************************************************************************
  --********************************************************************************

  ------------------------------------------
  -- Signals for user logic slave model s/w accessible register example
  ------------------------------------------
  signal slv_reg0                  : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal slv_reg1                  : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal slv_reg2                  : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal slv_reg_write_sel           : std_logic_vector(0 to 2);
  signal slv_reg_read_sel            : std_logic_vector(0 to 2);
  signal slv_ip2bus_data             : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal slv_read_ack               : std_logic;
  signal slv_write_ack              : std_logic;

begin

  --USER logic implementation added here
  --********************************************************************************
  --********************************************************************************
   u_Envelope_HDL : envelopeHDL
   PORT MAP( clk => slow_clock,
           reset => Bus2IP_Reset,
           clk_enable => '1',
           In1 => Filter_Signed,
           Out1 =>  Out_Filter
           );

  In_filter <= slv_reg0 (16 to 31);
  Filter_Signed <= In_filter - slv_reg2(16 to 31);

  Test_In(15 downto 0) <= In_Filter;

  --********************************************************************************
  --********************************************************************************
  -- CLOCK DIVIDER 8MHz -> 8KHz
  clk_div : PROCESS(F_CLK)
  BEGIN
   IF Bus2IP_Reset = '1' THEN
      div_count <= to_unsigned(1, 9);
   ELSIF F_CLK'event AND F_CLK = '1' THEN
      IF div_count = to_unsigned(499, 9) THEN
        div_count <= to_unsigned(0, 9);
        slow_clock <= not slow_clock;
      ELSE
      div_count <= div_count + 1;
    END IF;
  END IF;
END PROCESS clk_div;

--End User implementation
--**************************************************************************************
--**************************************************************************************

------------------------------------------
-- Example code to read/write user logic slave model s/w accessible registers
--
-- Note:
-- The example code presented here is to show you one way of reading/writing
-- software accessible registers implemented in the user logic slave model.
-- Each bit of the Bus2IP_WrCE/Bus2IP_RdCE signals is configured to correspond
-- to one software accessible register by the top level template. For example,
-- if you have four 32 bit software accessible registers in the user logic,
-- you are basically operating on the following memory mapped registers:
--
--    Bus2IP_WrCE/Bus2IP_RdCE   Memory Mapped Register
--                     "1000"   C_BASEADDR + 0x0
--                     "0100"   C_BASEADDR + 0x4
--                     "0010"   C_BASEADDR + 0x8
--                     "0001"   C_BASEADDR + 0xC
--
------------------------------------------
slv_reg_write_sel <= Bus2IP_WrCE(0 to 2);
slv_reg_read_sel  <= Bus2IP_RdCE(0 to 2);
slv_write_ack     <= Bus2IP_WrCE(0) or Bus2IP_WrCE(1) or Bus2IP_WrCE(2);
slv_read_ack      <= Bus2IP_RdCE(0) or Bus2IP_RdCE(1) or Bus2IP_RdCE(2);

-- implement slave model software accessible register(s)
SLAVE_REG_WRITE_PROC : process( Bus2IP_CLK ) is
begin

  if Bus2IP_CLK'event and Bus2IP_CLK = '1' then
    if Bus2IP_Reset = '1' then
      slv_reg0 <= (others => '0');
      slv_reg1 <= (others => '0');
      slv_reg2 <= (others => '0');
    else
      case slv_reg_write_sel is
        when "100" =>
          for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
            if ( Bus2IP_BE(byte_index) = '1' ) then
              slv_reg0(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
            end if;
          end loop;
        when "010" =>
          for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
            if ( Bus2IP_BE(byte_index) = '1' ) then
            slv_reg1(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
          end if;
        end loop;
      when "001" =>
        for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
          if ( Bus2IP_BE(byte_index) = '1' ) then
            slv_reg2(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
          end if;
        end loop;
      when others => null;
    end case;
  end if;
end if;

end process SLAVE_REG_WRITE_PROC;

-- implement slave model software accessible register(s) read mux
SLAVE_REG_READ_PROC : process( slv_reg_read_sel, slv_reg0, slv_reg1, slv_reg2, Out_Filter ) is
begin

  case slv_reg_read_sel is
    --when "100" => slv_ip2bus_data <= slv_reg0;
    when "100" => slv_ip2bus_data <= Test_In;
    when "010" => slv_ip2bus_data <= Out_Filter;
    when "001" => slv_ip2bus_data <= slv_reg2;
    when others => slv_ip2bus_data <= (others => '0');
  end case;

end process SLAVE_REG_READ_PROC;

------------------------------------------
-- Example code to drive IP to Bus signals
------------------------------------------
IP2Bus_Data  <= slv_ip2bus_data when slv_read_ack = '1' else
                (others => '0');

IP2Bus_WrAck <= slv_write_ack;
IP2Bus_RdAck <= slv_read_ack;
IP2Bus_Error <= '0';

end IMP;
```

envelopeHDL.vhd

```
-- -------------------------------------------------------------
-- File Name: C:\Users\Ricky\Desktop\Apnea\Simulink\HDL\envelopeHDL.vhd
-- Created: 2011-06-05 16:35:16
--
-- Generated by MATLAB 7.10 and Simulink HDL Coder 1.7
-- -------------------------------------------------------------


-- -------------------------------------------------------------
-- Module: envelopeHDL
-- Source Path: envelopeHDL
-- Hierarchy Level: 0
-- -------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY envelopeHDL IS
  PORT( clk                     :   IN    std_logic;
        reset                   :   IN    std_logic;
        clk_enable              :   IN    std_logic;
        In1                     :   IN    std_logic_vector(15 DOWNTO 0);  -- int16
        ce_out                  :   OUT   std_logic;
        Out1                    :   OUT   std_logic_vector(31 DOWNTO 0)  -- sfix32_En19
        );
END envelopeHDL;

ARCHITECTURE rtl OF envelopeHDL IS

  -- Component Declarations
  COMPONENT Timing_Controller
    PORT( clk                     :   IN    std_logic;
          reset                   :   IN    std_logic;
          clk_enable              :   IN    std_logic;
          enb                     :   OUT   std_logic;
          enb_1_500_0             :   OUT   std_logic;
          enb_1_500_1             :   OUT   std_logic
          );
  END COMPONENT;

  COMPONENT Discrete_FIR_Filter1
    PORT( clk                     :   IN    std_logic;
          enb                     :   IN    std_logic;
          reset                   :   IN    std_logic;
          Discrete_FIR_Filter1_in          :   IN    std_logic_vector(15 DOWNTO 0);  -- int16
          Discrete_FIR_Filter1_out         :   OUT   std_logic_vector(31 DOWNTO 0)  -- sfix32_En17
          );
  END COMPONENT;

  COMPONENT Discrete_FIR_Filter
    PORT( clk                     :   IN    std_logic;
          enb_1_500_0             :   IN    std_logic;
        reset                           :   IN     std_logic;
        Discrete_FIR_Filter_in          :   IN     std_logic_vector(15 DOWNTO 0);  -- int16
        Discrete_FIR_Filter_out         :   OUT    std_logic_vector(31 DOWNTO 0)  -- sfix32_En19
        );
  END COMPONENT;

  -- Component Configuration Statements
  FOR ALL : Timing_Controller
    USE ENTITY work.Timing_Controller(rtl);

  FOR ALL : Discrete_FIR_Filter1
    USE ENTITY work.Discrete_FIR_Filter1(rtl);

  FOR ALL : Discrete_FIR_Filter
    USE ENTITY work.Discrete_FIR_Filter(rtl);

  -- Signals
  SIGNAL enb_1_500_0                      : std_logic;
  SIGNAL enb                              : std_logic;
  SIGNAL enb_1_500_1                      : std_logic;
  SIGNAL Discrete_FIR_Filter1_out1        : std_logic_vector(31 DOWNTO 0);  -- ufix32
  SIGNAL Discrete_FIR_Filter1_out1_signed : signed(31 DOWNTO 0);  -- sfix32_En17
  SIGNAL Downsample_bypass_reg            : signed(31 DOWNTO 0);  -- sfix32
  SIGNAL Downsample_out1                  : signed(31 DOWNTO 0);  -- sfix32_En17
  SIGNAL Abs_y                            : signed(31 DOWNTO 0);  -- sfix32_En17
  SIGNAL Abs_cast                         : signed(32 DOWNTO 0);  -- sfix33_En17
  SIGNAL Abs_cast_1                       : signed(32 DOWNTO 0);  -- sfix33_En17
  SIGNAL Abs_out1                         : signed(15 DOWNTO 0);  -- int16
  SIGNAL Abs_out1_1                       : std_logic_vector(15 DOWNTO 0);  -- ufix16
  SIGNAL Discrete_FIR_Filter_out1         : std_logic_vector(31 DOWNTO 0);  -- ufix32

BEGIN
  u_Timing_Controller : Timing_Controller
    PORT MAP( clk => clk,
              reset => reset,
              clk_enable => clk_enable,
              enb => enb,
              enb_1_500_0 => enb_1_500_0,
              enb_1_500_1 => enb_1_500_1
              );

  u_Discrete_FIR_Filter1 : Discrete_FIR_Filter1
    PORT MAP( clk => clk,
              enb => enb,
              reset => reset,
              Discrete_FIR_Filter1_in => In1,  -- int16
              Discrete_FIR_Filter1_out => Discrete_FIR_Filter1_out1  -- sfix32_En17
              );

  u_Discrete_FIR_Filter : Discrete_FIR_Filter
    PORT MAP( clk => clk,
              enb_1_500_0 => enb_1_500_0,
            reset => reset,
            Discrete_FIR_Filter_in => Abs_out1_1,  -- int16
            Discrete_FIR_Filter_out => Discrete_FIR_Filter_out1  -- sfix32_En19
            );

  Discrete_FIR_Filter1_out1_signed <= signed(Discrete_FIR_Filter1_out1);

  -- Downsample: Downsample by 500, Sample offset 0
  -- Downsample bypass register
  Downsample_bypass_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      Downsample_bypass_reg <= to_signed(0, 32);
    ELSIF clk'EVENT AND clk = '1' THEN
      IF enb_1_500_1 = '1' THEN
        Downsample_bypass_reg <= Discrete_FIR_Filter1_out1_signed;
      END IF;
    END IF;
  END PROCESS Downsample_bypass_process;

  Downsample_out1 <= Discrete_FIR_Filter1_out1_signed WHEN enb_1_500_1 = '1' ELSE
      Downsample_bypass_reg;

  Abs_cast <= resize(Downsample_out1, 33);
  Abs_cast_1 <=  - (Abs_cast);

  Abs_y <= Abs_cast_1(31 DOWNTO 0) WHEN Downsample_out1 < 0 ELSE
      Downsample_out1;
  Abs_out1 <= resize(Abs_y(31 DOWNTO 17), 16);

  Abs_out1_1 <= std_logic_vector(Abs_out1);

  Out1 <= Discrete_FIR_Filter_out1;

  ce_out <= enb_1_500_1;

END rtl;
```

Discrete_FIR_Filter1.vhd

```
-- -------------------------------------------------------------
-- File Name: C:\Users\Ricky\Desktop\Apnea\Simulink\HDL\Discrete_FIR_Filter1
-- Created: 2011-06-05 16:35:12
-- Generated by MATLAB 7.10 and Simulink HDL Coder 1.7
-- -------------------------------------------------------------
--
-- -------------------------------------------------------------
-- Module: Discrete_FIR_Filter1
-- Source Path: envelopeHDL/Discrete FIR Filter1
-- -------------------------------------------------------------
--
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.ALL;
ENTITY Discrete_FIR_Filter1 IS
  PORT( clk                               :   IN    std_logic;
        enb                               :   IN    std_logic;
        reset                             :   IN    std_logic;
        Discrete_FIR_Filter1_in           :   IN    std_logic_vector(15 DOWNTO 0); -- sfix16
        Discrete_FIR_Filter1_out          :   OUT   std_logic_vector(31 DOWNTO 0)  -- sfix32_En17
        );

END Discrete_FIR_Filter1;

-------------------------------------------------------------
--Module Architecture: Discrete_FIR_Filter1
-------------------------------------------------------------
ARCHITECTURE rtl OF Discrete_FIR_Filter1 IS
  -- Local Functions
  -- Type Definitions
  TYPE delay_pipeline_type IS ARRAY (NATURAL range <>) OF signed(15 DOWNTO 0); -- sfix16
  -- Constants
  CONSTANT coeff1                         : signed(15 DOWNTO 0) := to_signed(1856, 16); -- sfix16_En17
  CONSTANT coeff2                         : signed(15 DOWNTO 0) := to_signed(3960, 16); -- sfix16_En17
  CONSTANT coeff3                         : signed(15 DOWNTO 0) := to_signed(9506, 16); -- sfix16_En17
  CONSTANT coeff4                         : signed(15 DOWNTO 0) := to_signed(16448, 16); -- sfix16_En17
  CONSTANT coeff5                         : signed(15 DOWNTO 0) := to_signed(22113, 16); -- sfix16_En17
  CONSTANT coeff6                         : signed(15 DOWNTO 0) := to_signed(24286, 16); -- sfix16_En17

  -- Signals
  SIGNAL delay_pipeline                   : delay_pipeline_type(0 TO 9); -- sfix16
  SIGNAL Discrete_FIR_Filter1_in_regtype  : signed(15 DOWNTO 0); -- sfix16
  SIGNAL tapsum1                          : signed(15 DOWNTO 0); -- sfix16
  SIGNAL add_cast                         : signed(15 DOWNTO 0); -- sfix16
  SIGNAL add_cast_1                       : signed(15 DOWNTO 0); -- sfix16
  SIGNAL add_temp                         : signed(16 DOWNTO 0); -- sfix17
  SIGNAL tapsum_mcand                     : signed(15 DOWNTO 0); -- sfix16
  SIGNAL tapsum2                          : signed(15 DOWNTO 0); -- sfix16
  SIGNAL add_cast_2                       : signed(15 DOWNTO 0); -- sfix16
  SIGNAL add_cast_3                       : signed(15 DOWNTO 0); -- sfix16
  SIGNAL add_temp_1                       : signed(16 DOWNTO 0); -- sfix17
  SIGNAL tapsum_mcand_1                   : signed(15 DOWNTO 0); -- sfix16
  SIGNAL tapsum3                     : signed(15 DOWNTO 0); -- sfix16
  SIGNAL add_cast_4                  : signed(15 DOWNTO 0); -- sfix16
  SIGNAL add_cast_5                  : signed(15 DOWNTO 0); -- sfix16
  SIGNAL add_temp_2                   : signed(16 DOWNTO 0); -- sfix17
  SIGNAL tapsum_mcand_2                   : signed(15 DOWNTO 0); -- sfix16
  SIGNAL tapsum4                     : signed(15 DOWNTO 0); -- sfix16
  SIGNAL add_cast_6                  : signed(15 DOWNTO 0); -- sfix16
  SIGNAL add_cast_7                  : signed(15 DOWNTO 0); -- sfix16
  SIGNAL add_temp_3                   : signed(16 DOWNTO 0); -- sfix17
  SIGNAL tapsum_mcand_3                   : signed(15 DOWNTO 0); -- sfix16
  SIGNAL tapsum5                     : signed(15 DOWNTO 0); -- sfix16
  SIGNAL add_cast_8                  : signed(15 DOWNTO 0); -- sfix16
  SIGNAL add_cast_9                  : signed(15 DOWNTO 0); -- sfix16
  SIGNAL add_temp_4                   : signed(16 DOWNTO 0); -- sfix17
  SIGNAL tapsum_mcand_4                   : signed(15 DOWNTO 0); -- sfix16
  SIGNAL product6                    : signed(31 DOWNTO 0); -- sfix32_En17
  SIGNAL product5                    : signed(31 DOWNTO 0); -- sfix32_En17
  SIGNAL product4                    : signed(31 DOWNTO 0); -- sfix32_En17
  SIGNAL product3                    : signed(31 DOWNTO 0); -- sfix32_En17
  SIGNAL product2                    : signed(31 DOWNTO 0); -- sfix32_En17
  SIGNAL product1_cast                 : signed(31 DOWNTO 0); -- sfix32_En17
  SIGNAL product1                    : signed(31 DOWNTO 0); -- sfix32_En17
  SIGNAL sum1                       : signed(31 DOWNTO 0); -- sfix32_En17
  SIGNAL add_cast_10                  : signed(31 DOWNTO 0); -- sfix32_En17
  SIGNAL add_cast_11                  : signed(31 DOWNTO 0); -- sfix32_En17
  SIGNAL add_temp_5                   : signed(32 DOWNTO 0); -- sfix33_En17
  SIGNAL sum2                       : signed(31 DOWNTO 0); -- sfix32_En17
  SIGNAL add_cast_12                  : signed(31 DOWNTO 0); -- sfix32_En17
  SIGNAL add_cast_13                  : signed(31 DOWNTO 0); -- sfix32_En17
  SIGNAL add_temp_6                   : signed(32 DOWNTO 0); -- sfix33_En17
  SIGNAL sum3                       : signed(31 DOWNTO 0); -- sfix32_En17
  SIGNAL add_cast_14                  : signed(31 DOWNTO 0); -- sfix32_En17
  SIGNAL add_cast_15                  : signed(31 DOWNTO 0); -- sfix32_En17
  SIGNAL add_temp_7                   : signed(32 DOWNTO 0); -- sfix33_En17
  SIGNAL sum4                       : signed(31 DOWNTO 0); -- sfix32_En17
  SIGNAL add_cast_16                  : signed(31 DOWNTO 0); -- sfix32_En17
  SIGNAL add_cast_17                  : signed(31 DOWNTO 0); -- sfix32_En17
  SIGNAL add_temp_8                   : signed(32 DOWNTO 0); -- sfix33_En17
  SIGNAL sum5                       : signed(31 DOWNTO 0); -- sfix32_En17
  SIGNAL add_cast_18                  : signed(31 DOWNTO 0); -- sfix32_En17
  SIGNAL add_cast_19                  : signed(31 DOWNTO 0); -- sfix32_En17
  SIGNAL add_temp_9                   : signed(32 DOWNTO 0); -- sfix33_En17
  SIGNAL output_typeconvert              : signed(31 DOWNTO 0); -- sfix32_En17

BEGIN

  -- Block Statements
  Delay_Pipeline_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      delay_pipeline(0 TO 9) <= (OTHERS => (OTHERS => '0'));
  ELSIF clk'event AND clk = '1' THEN
    IF enb = '1' THEN
      delay_pipeline(0) <= signed(Discrete_FIR_Filter1_in);
      delay_pipeline(1 TO 9) <= delay_pipeline(0 TO 8);
    END IF;
  END IF;
END PROCESS Delay_Pipeline_process;

Discrete_FIR_Filter1_in_regtype <= signed(Discrete_FIR_Filter1_in);

add_cast <= Discrete_FIR_Filter1_in_regtype;
add_cast_1 <= delay_pipeline(9);
add_temp <= resize(add_cast, 17) + resize(add_cast_1, 17);
tapsum1 <= add_temp(15 DOWNTO 0);

tapsum_mcand <= tapsum1;

add_cast_2 <= delay_pipeline(0);
add_cast_3 <= delay_pipeline(8);
add_temp_1 <= resize(add_cast_2, 17) + resize(add_cast_3, 17);
tapsum2 <= add_temp_1(15 DOWNTO 0);

tapsum_mcand_1 <= tapsum2;

add_cast_4 <= delay_pipeline(1);
add_cast_5 <= delay_pipeline(7);
add_temp_2 <= resize(add_cast_4, 17) + resize(add_cast_5, 17);
tapsum3 <= add_temp_2(15 DOWNTO 0);

tapsum_mcand_2 <= tapsum3;

add_cast_6 <= delay_pipeline(2);
add_cast_7 <= delay_pipeline(6);
add_temp_3 <= resize(add_cast_6, 17) + resize(add_cast_7, 17);
tapsum4 <= add_temp_3(15 DOWNTO 0);

tapsum_mcand_3 <= tapsum4;

add_cast_8 <= delay_pipeline(3);
add_cast_9 <= delay_pipeline(5);
add_temp_4 <= resize(add_cast_8, 17) + resize(add_cast_9, 17);
tapsum5 <= add_temp_4(15 DOWNTO 0);

tapsum_mcand_4 <= tapsum5;

product6 <= delay_pipeline(4) * coeff6;

product5 <= tapsum_mcand_4 * coeff5;

product4 <= tapsum_mcand_3 * coeff4;

product3 <= tapsum_mcand_2 * coeff3;
```

```
  product2 <= tapsum_mcand_1 * coeff2;

  product1_cast <= product1;

  product1 <= tapsum_mcand * coeff1;

  add_cast_10 <= product1_cast;
  add_cast_11 <= product2;
  add_temp_5 <= resize(add_cast_10, 33) + resize(add_cast_11, 33);
  sum1 <= add_temp_5(31 DOWNTO 0);

  add_cast_12 <= sum1;
  add_cast_13 <= product3;
  add_temp_6 <= resize(add_cast_12, 33) + resize(add_cast_13, 33);
  sum2 <= add_temp_6(31 DOWNTO 0);

  add_cast_14 <= sum2;
  add_cast_15 <= product4;
  add_temp_7 <= resize(add_cast_14, 33) + resize(add_cast_15, 33);
  sum3 <= add_temp_7(31 DOWNTO 0);

  add_cast_16 <= sum3;
  add_cast_17 <= product5;
  add_temp_8 <= resize(add_cast_16, 33) + resize(add_cast_17, 33);
  sum4 <= add_temp_8(31 DOWNTO 0);

  add_cast_18 <= sum4;
  add_cast_19 <= product6;
  add_temp_9 <= resize(add_cast_18, 33) + resize(add_cast_19, 33);
  sum5 <= add_temp_9(31 DOWNTO 0);

  output_typeconvert <= sum5;

  -- Assignment Statements
  Discrete_FIR_Filter1_out <= std_logic_vector(output_typeconvert);
END rtl;
```

Discrete_FIR_Filter.vhd

```
-- -------------------------------------------------------------
-- File Name: C:\Users\Ricky\Desktop\Apnea\Simulink\HDL\Discrete_FIR_Filter
-- Created: 2011-06-05 16:35:14
-- Generated by MATLAB 7.10 and Simulink HDL Coder 1.7
-- -------------------------------------------------------------
--
-- -------------------------------------------------------------
-- Module: Discrete_FIR_Filter
-- Source Path: envelopeHDL/Discrete FIR Filter
-- -------------------------------------------------------------
--
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.ALL;
ENTITY Discrete_FIR_Filter IS
  PORT( clk                             :   IN    std_logic;
        enb_1_500_0                     :   IN    std_logic;
        reset                           :   IN    std_logic;
        Discrete_FIR_Filter_in          :   IN    std_logic_vector(15 DOWNTO 0); -- sfix16
        Discrete_FIR_Filter_out         :   OUT   std_logic_vector(31 DOWNTO 0)  -- sfix32_En19
        );

END Discrete_FIR_Filter;

-------------------------------------------------------------
--Module Architecture: Discrete_FIR_Filter
-------------------------------------------------------------
ARCHITECTURE rtl OF Discrete_FIR_Filter IS
  -- Local Functions
  -- Type Definitions
  TYPE delay_pipeline_type IS ARRAY (NATURAL range <>) OF signed(15 DOWNTO 0); -- sfix16
  -- Constants
  CONSTANT coeff1                         : signed(15 DOWNTO 0) := to_signed(17476, 16); -- sfix16_En19
  CONSTANT coeff2                         : signed(15 DOWNTO 0) := to_signed(17476, 16); -- sfix16_En19
  CONSTANT coeff3                         : signed(15 DOWNTO 0) := to_signed(17476, 16); -- sfix16_En19
  CONSTANT coeff4                         : signed(15 DOWNTO 0) := to_signed(17476, 16); -- sfix16_En19
  CONSTANT coeff5                         : signed(15 DOWNTO 0) := to_signed(17476, 16); -- sfix16_En19
  CONSTANT coeff6                         : signed(15 DOWNTO 0) := to_signed(17476, 16); -- sfix16_En19
  CONSTANT coeff7                         : signed(15 DOWNTO 0) := to_signed(17476, 16); -- sfix16_En19
  CONSTANT coeff8                         : signed(15 DOWNTO 0) := to_signed(17476, 16); -- sfix16_En19
  CONSTANT coeff9                         : signed(15 DOWNTO 0) := to_signed(17476, 16); -- sfix16_En19
  CONSTANT coeff10                        : signed(15 DOWNTO 0) := to_signed(17476, 16); -- sfix16_En19
  CONSTANT coeff11                        : signed(15 DOWNTO 0) := to_signed(17476, 16); -- sfix16_En19
  CONSTANT coeff12                        : signed(15 DOWNTO 0) := to_signed(17476, 16); -- sfix16_En19
  CONSTANT coeff13                        : signed(15 DOWNTO 0) := to_signed(17476, 16); -- sfix16_En19
  CONSTANT coeff14                        : signed(15 DOWNTO 0) := to_signed(17476, 16); -- sfix16_En19
  CONSTANT coeff15                        : signed(15 DOWNTO 0) := to_signed(17476, 16); -- sfix16_En19

  -- Signals
  SIGNAL delay_pipeline                   : delay_pipeline_type(0 TO 28); -- sfix16
  SIGNAL Discrete_FIR_Filter_in_regtype   : signed(15 DOWNTO 0); -- sfix16
SIGNAL tapsum1              : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_cast             : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_cast_1           : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_temp             : signed(16 DOWNTO 0); -- sfix17
SIGNAL tapsum_mcand         : signed(15 DOWNTO 0); -- sfix16
SIGNAL tapsum2              : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_cast_2           : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_cast_3           : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_temp_1           : signed(16 DOWNTO 0); -- sfix17
SIGNAL tapsum_mcand_1       : signed(15 DOWNTO 0); -- sfix16
SIGNAL tapsum3              : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_cast_4           : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_cast_5           : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_temp_2           : signed(16 DOWNTO 0); -- sfix17
SIGNAL tapsum_mcand_2       : signed(15 DOWNTO 0); -- sfix16
SIGNAL tapsum4              : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_cast_6           : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_cast_7           : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_temp_3           : signed(16 DOWNTO 0); -- sfix17
SIGNAL tapsum_mcand_3       : signed(15 DOWNTO 0); -- sfix16
SIGNAL tapsum5              : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_cast_8           : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_cast_9           : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_temp_4           : signed(16 DOWNTO 0); -- sfix17
SIGNAL tapsum_mcand_4       : signed(15 DOWNTO 0); -- sfix16
SIGNAL tapsum6              : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_cast_10          : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_cast_11          : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_temp_5           : signed(16 DOWNTO 0); -- sfix17
SIGNAL tapsum_mcand_5       : signed(15 DOWNTO 0); -- sfix16
SIGNAL tapsum7              : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_cast_12          : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_cast_13          : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_temp_6           : signed(16 DOWNTO 0); -- sfix17
SIGNAL tapsum_mcand_6       : signed(15 DOWNTO 0); -- sfix16
SIGNAL tapsum8              : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_cast_14          : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_cast_15          : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_temp_7           : signed(16 DOWNTO 0); -- sfix17
SIGNAL tapsum_mcand_7       : signed(15 DOWNTO 0); -- sfix16
SIGNAL tapsum9              : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_cast_16          : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_cast_17          : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_temp_8           : signed(16 DOWNTO 0); -- sfix17
SIGNAL tapsum_mcand_8       : signed(15 DOWNTO 0); -- sfix16
SIGNAL tapsum10             : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_cast_18          : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_cast_19          : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_temp_9           : signed(16 DOWNTO 0); -- sfix17
SIGNAL tapsum_mcand_9       : signed(15 DOWNTO 0); -- sfix16
SIGNAL tapsum11             : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_cast_20          : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_cast_21                    : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_temp_10                    : signed(16 DOWNTO 0); -- sfix17
SIGNAL tapsum_mcand_10                : signed(15 DOWNTO 0); -- sfix16
SIGNAL tapsum12                       : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_cast_22                    : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_cast_23                    : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_temp_11                    : signed(16 DOWNTO 0); -- sfix17
SIGNAL tapsum_mcand_11                : signed(15 DOWNTO 0); -- sfix16
SIGNAL tapsum13                       : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_cast_24                    : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_cast_25                    : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_temp_12                    : signed(16 DOWNTO 0); -- sfix17
SIGNAL tapsum_mcand_12                : signed(15 DOWNTO 0); -- sfix16
SIGNAL tapsum14                       : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_cast_26                    : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_cast_27                    : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_temp_13                    : signed(16 DOWNTO 0); -- sfix17
SIGNAL tapsum_mcand_13                : signed(15 DOWNTO 0); -- sfix16
SIGNAL tapsum15                       : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_cast_28                    : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_cast_29                    : signed(15 DOWNTO 0); -- sfix16
SIGNAL add_temp_14                    : signed(16 DOWNTO 0); -- sfix17
SIGNAL tapsum_mcand_14                : signed(15 DOWNTO 0); -- sfix16
SIGNAL product15                      : signed(31 DOWNTO 0); -- sfix32_En19
SIGNAL product14                      : signed(31 DOWNTO 0); -- sfix32_En19
SIGNAL product13                      : signed(31 DOWNTO 0); -- sfix32_En19
SIGNAL product12                      : signed(31 DOWNTO 0); -- sfix32_En19
SIGNAL product11                      : signed(31 DOWNTO 0); -- sfix32_En19
SIGNAL product10                      : signed(31 DOWNTO 0); -- sfix32_En19
SIGNAL product9                       : signed(31 DOWNTO 0); -- sfix32_En19
SIGNAL product8                       : signed(31 DOWNTO 0); -- sfix32_En19
SIGNAL product7                       : signed(31 DOWNTO 0); -- sfix32_En19
SIGNAL product6                       : signed(31 DOWNTO 0); -- sfix32_En19
SIGNAL product5                       : signed(31 DOWNTO 0); -- sfix32_En19
SIGNAL product4                       : signed(31 DOWNTO 0); -- sfix32_En19
SIGNAL product3                       : signed(31 DOWNTO 0); -- sfix32_En19
SIGNAL product2                       : signed(31 DOWNTO 0); -- sfix32_En19
SIGNAL product1_cast                  : signed(31 DOWNTO 0); -- sfix32_En19
SIGNAL product1                       : signed(31 DOWNTO 0); -- sfix32_En19
SIGNAL sum1                           : signed(31 DOWNTO 0); -- sfix32_En19
SIGNAL add_cast_30                    : signed(31 DOWNTO 0); -- sfix32_En19
SIGNAL add_cast_31                    : signed(31 DOWNTO 0); -- sfix32_En19
SIGNAL add_temp_15                    : signed(32 DOWNTO 0); -- sfix33_En19
SIGNAL sum2                           : signed(31 DOWNTO 0); -- sfix32_En19
SIGNAL add_cast_32                    : signed(31 DOWNTO 0); -- sfix32_En19
SIGNAL add_cast_33                    : signed(31 DOWNTO 0); -- sfix32_En19
SIGNAL add_temp_16                    : signed(32 DOWNTO 0); -- sfix33_En19
SIGNAL sum3                           : signed(31 DOWNTO 0); -- sfix32_En19
SIGNAL add_cast_34                    : signed(31 DOWNTO 0); -- sfix32_En19
SIGNAL add_cast_35                    : signed(31 DOWNTO 0); -- sfix32_En19
SIGNAL add_temp_17                    : signed(32 DOWNTO 0); -- sfix33_En19
SIGNAL sum4                           : signed(31 DOWNTO 0); -- sfix32_En19
```

```
  SIGNAL add_cast_36                   : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_cast_37                   : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_temp_18                   : signed(32 DOWNTO 0); -- sfix33_En19
  SIGNAL sum5                          : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_cast_38                   : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_cast_39                   : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_temp_19                   : signed(32 DOWNTO 0); -- sfix33_En19
  SIGNAL sum6                          : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_cast_40                   : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_cast_41                   : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_temp_20                   : signed(32 DOWNTO 0); -- sfix33_En19
  SIGNAL sum7                          : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_cast_42                   : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_cast_43                   : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_temp_21                   : signed(32 DOWNTO 0); -- sfix33_En19
  SIGNAL sum8                          : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_cast_44                   : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_cast_45                   : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_temp_22                   : signed(32 DOWNTO 0); -- sfix33_En19
  SIGNAL sum9                          : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_cast_46                   : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_cast_47                   : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_temp_23                   : signed(32 DOWNTO 0); -- sfix33_En19
  SIGNAL sum10                         : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_cast_48                   : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_cast_49                   : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_temp_24                   : signed(32 DOWNTO 0); -- sfix33_En19
  SIGNAL sum11                         : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_cast_50                   : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_cast_51                   : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_temp_25                   : signed(32 DOWNTO 0); -- sfix33_En19
  SIGNAL sum12                         : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_cast_52                   : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_cast_53                   : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_temp_26                   : signed(32 DOWNTO 0); -- sfix33_En19
  SIGNAL sum13                         : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_cast_54                   : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_cast_55                   : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_temp_27                   : signed(32 DOWNTO 0); -- sfix33_En19
  SIGNAL sum14                         : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_cast_56                   : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_cast_57                   : signed(31 DOWNTO 0); -- sfix32_En19
  SIGNAL add_temp_28                   : signed(32 DOWNTO 0); -- sfix33_En19
  SIGNAL output_typeconvert            : signed(31 DOWNTO 0); -- sfix32_En19

BEGIN

  -- Block Statements
  Delay_Pipeline_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      delay_pipeline(0 TO 28) <= (OTHERS => (OTHERS => '0'));
    ELSIF clk'event AND clk = '1' THEN
      IF enb_1_500_0 = '1' THEN
        delay_pipeline(0) <= signed(Discrete_FIR_Filter_in);
        delay_pipeline(1 TO 28) <= delay_pipeline(0 TO 27);
      END IF;
    END IF;
  END PROCESS Delay_Pipeline_process;

  Discrete_FIR_Filter_in_regtype <= signed(Discrete_FIR_Filter_in);

  add_cast <= Discrete_FIR_Filter_in_regtype;
  add_cast_1 <= delay_pipeline(28);
  add_temp <= resize(add_cast, 17) + resize(add_cast_1, 17);
  tapsum1 <= add_temp(15 DOWNTO 0);

  tapsum_mcand <= tapsum1;

  add_cast_2 <= delay_pipeline(0);
  add_cast_3 <= delay_pipeline(27);
  add_temp_1 <= resize(add_cast_2, 17) + resize(add_cast_3, 17);
  tapsum2 <= add_temp_1(15 DOWNTO 0);

  tapsum_mcand_1 <= tapsum2;

  add_cast_4 <= delay_pipeline(1);
  add_cast_5 <= delay_pipeline(26);
  add_temp_2 <= resize(add_cast_4, 17) + resize(add_cast_5, 17);
  tapsum3 <= add_temp_2(15 DOWNTO 0);

  tapsum_mcand_2 <= tapsum3;

  add_cast_6 <= delay_pipeline(2);
  add_cast_7 <= delay_pipeline(25);
  add_temp_3 <= resize(add_cast_6, 17) + resize(add_cast_7, 17);
  tapsum4 <= add_temp_3(15 DOWNTO 0);

  tapsum_mcand_3 <= tapsum4;

  add_cast_8 <= delay_pipeline(3);
  add_cast_9 <= delay_pipeline(24);
  add_temp_4 <= resize(add_cast_8, 17) + resize(add_cast_9, 17);
  tapsum5 <= add_temp_4(15 DOWNTO 0);

  tapsum_mcand_4 <= tapsum5;

  add_cast_10 <= delay_pipeline(4);
  add_cast_11 <= delay_pipeline(23);
  add_temp_5 <= resize(add_cast_10, 17) + resize(add_cast_11, 17);
  tapsum6 <= add_temp_5(15 DOWNTO 0);

  tapsum_mcand_5 <= tapsum6;
```

```
add_cast_12 <= delay_pipeline(5);
add_cast_13 <= delay_pipeline(22);
add_temp_6 <= resize(add_cast_12, 17) + resize(add_cast_13, 17);
tapsum7 <= add_temp_6(15 DOWNTO 0);

tapsum_mcand_6 <= tapsum7;

add_cast_14 <= delay_pipeline(6);
add_cast_15 <= delay_pipeline(21);
add_temp_7 <= resize(add_cast_14, 17) + resize(add_cast_15, 17);
tapsum8 <= add_temp_7(15 DOWNTO 0);

tapsum_mcand_7 <= tapsum8;

add_cast_16 <= delay_pipeline(7);
add_cast_17 <= delay_pipeline(20);
add_temp_8 <= resize(add_cast_16, 17) + resize(add_cast_17, 17);
tapsum9 <= add_temp_8(15 DOWNTO 0);

tapsum_mcand_8 <= tapsum9;

add_cast_18 <= delay_pipeline(8);
add_cast_19 <= delay_pipeline(19);
add_temp_9 <= resize(add_cast_18, 17) + resize(add_cast_19, 17);
tapsum10 <= add_temp_9(15 DOWNTO 0);

tapsum_mcand_9 <= tapsum10;

add_cast_20 <= delay_pipeline(9);
add_cast_21 <= delay_pipeline(18);
add_temp_10 <= resize(add_cast_20, 17) + resize(add_cast_21, 17);
tapsum11 <= add_temp_10(15 DOWNTO 0);

tapsum_mcand_10 <= tapsum11;

add_cast_22 <= delay_pipeline(10);
add_cast_23 <= delay_pipeline(17);
add_temp_11 <= resize(add_cast_22, 17) + resize(add_cast_23, 17);
tapsum12 <= add_temp_11(15 DOWNTO 0);

tapsum_mcand_11 <= tapsum12;

add_cast_24 <= delay_pipeline(11);
add_cast_25 <= delay_pipeline(16);
add_temp_12 <= resize(add_cast_24, 17) + resize(add_cast_25, 17);
tapsum13 <= add_temp_12(15 DOWNTO 0);

tapsum_mcand_12 <= tapsum13;

add_cast_26 <= delay_pipeline(12);
add_cast_27 <= delay_pipeline(15);
add_temp_13 <= resize(add_cast_26, 17) + resize(add_cast_27, 17);
```

```
tapsum14 <= add_temp_13(15 DOWNTO 0);

tapsum_mcand_13 <= tapsum14;

add_cast_28 <= delay_pipeline(13);
add_cast_29 <= delay_pipeline(14);
add_temp_14 <= resize(add_cast_28, 17) + resize(add_cast_29, 17);
tapsum15 <= add_temp_14(15 DOWNTO 0);

tapsum_mcand_14 <= tapsum15;

product15 <= tapsum_mcand_14 * coeff15;

product14 <= tapsum_mcand_13 * coeff14;

product13 <= tapsum_mcand_12 * coeff13;

product12 <= tapsum_mcand_11 * coeff12;

product11 <= tapsum_mcand_10 * coeff11;

product10 <= tapsum_mcand_9 * coeff10;

product9 <= tapsum_mcand_8 * coeff9;

product8 <= tapsum_mcand_7 * coeff8;

product7 <= tapsum_mcand_6 * coeff7;

product6 <= tapsum_mcand_5 * coeff6;

product5 <= tapsum_mcand_4 * coeff5;

product4 <= tapsum_mcand_3 * coeff4;

product3 <= tapsum_mcand_2 * coeff3;

product2 <= tapsum_mcand_1 * coeff2;

product1_cast <= product1;

product1 <= tapsum_mcand * coeff1;

add_cast_30 <= product1_cast;
add_cast_31 <= product2;
add_temp_15 <= resize(add_cast_30, 33) + resize(add_cast_31, 33);
sum1 <= add_temp_15(31 DOWNTO 0);

add_cast_32 <= sum1;
add_cast_33 <= product3;
add_temp_16 <= resize(add_cast_32, 33) + resize(add_cast_33, 33);
sum2 <= add_temp_16(31 DOWNTO 0);
```

```
add_cast_34 <= sum2;
add_cast_35 <= product4;
add_temp_17 <= resize(add_cast_34, 33) + resize(add_cast_35, 33);
sum3 <= add_temp_17(31 DOWNTO 0);

add_cast_36 <= sum3;
add_cast_37 <= product5;
add_temp_18 <= resize(add_cast_36, 33) + resize(add_cast_37, 33);
sum4 <= add_temp_18(31 DOWNTO 0);

add_cast_38 <= sum4;
add_cast_39 <= product6;
add_temp_19 <= resize(add_cast_38, 33) + resize(add_cast_39, 33);
sum5 <= add_temp_19(31 DOWNTO 0);

add_cast_40 <= sum5;
add_cast_41 <= product7;
add_temp_20 <= resize(add_cast_40, 33) + resize(add_cast_41, 33);
sum6 <= add_temp_20(31 DOWNTO 0);

add_cast_42 <= sum6;
add_cast_43 <= product8;
add_temp_21 <= resize(add_cast_42, 33) + resize(add_cast_43, 33);
sum7 <= add_temp_21(31 DOWNTO 0);

add_cast_44 <= sum7;
add_cast_45 <= product9;
add_temp_22 <= resize(add_cast_44, 33) + resize(add_cast_45, 33);
sum8 <= add_temp_22(31 DOWNTO 0);

add_cast_46 <= sum8;
add_cast_47 <= product10;
add_temp_23 <= resize(add_cast_46, 33) + resize(add_cast_47, 33);
sum9 <= add_temp_23(31 DOWNTO 0);

add_cast_48 <= sum9;
add_cast_49 <= product11;
add_temp_24 <= resize(add_cast_48, 33) + resize(add_cast_49, 33);
sum10 <= add_temp_24(31 DOWNTO 0);

add_cast_50 <= sum10;
add_cast_51 <= product12;
add_temp_25 <= resize(add_cast_50, 33) + resize(add_cast_51, 33);
sum11 <= add_temp_25(31 DOWNTO 0);

add_cast_52 <= sum11;
add_cast_53 <= product13;
add_temp_26 <= resize(add_cast_52, 33) + resize(add_cast_53, 33);
sum12 <= add_temp_26(31 DOWNTO 0);

add_cast_54 <= sum12;
```

```
  add_cast_55 <= product14;
  add_temp_27 <= resize(add_cast_54, 33) + resize(add_cast_55, 33);
  sum13 <= add_temp_27(31 DOWNTO 0);

  add_cast_56 <= sum13;
  add_cast_57 <= product15;
  add_temp_28 <= resize(add_cast_56, 33) + resize(add_cast_57, 33);
  sum14 <= add_temp_28(31 DOWNTO 0);

  output_typeconvert <= sum14;

  -- Assignment Statements
  Discrete_FIR_Filter_out <= std_logic_vector(output_typeconvert);
END rtl;
```

Timing_Controller.vhd

```
-- -------------------------------------------------------------
-- File Name: C:\Users\Ricky\Desktop\Apnea\Simulink\HDL\Timing_Controller.vhd
-- Created: 2011-06-05 16:35:12
--
-- Generated by MATLAB 7.10 and Simulink HDL Coder 1.7
-- -------------------------------------------------------------


-- -------------------------------------------------------------
-- Module: Timing_Controller
-- Source Path: Timing Controller
-- Hierarchy Level: 1
-- -------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY Timing_Controller IS
  PORT( clk                               :   IN    std_logic;
        reset                             :   IN    std_logic;
        clk_enable                        :   IN    std_logic;
        enb                               :   OUT   std_logic;
        enb_1_500_0                       :   OUT   std_logic;
        enb_1_500_1                       :   OUT   std_logic
        );
END Timing_Controller;

ARCHITECTURE rtl OF Timing_Controller IS

  -- Signals
  SIGNAL count500                         : unsigned(8 DOWNTO 0);  -- ufix9
  SIGNAL phase_all                        : std_logic;
  SIGNAL phase_0                          : std_logic;
  SIGNAL phase_0_tmp                      : std_logic;
  SIGNAL phase_1                          : std_logic;
  SIGNAL phase_1_tmp                      : std_logic;

BEGIN
  Counter500 : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      count500 <= to_unsigned(1, 9);
    ELSIF clk'event AND clk = '1' THEN
      IF clk_enable = '1' THEN
        IF count500 = to_unsigned(499, 9) THEN
          count500 <= to_unsigned(0, 9);
        ELSE
          count500 <= count500 + 1;
        END IF;
      END IF;
    END IF;
  END PROCESS Counter500;
  phase_all <= '1' WHEN clk_enable = '1' ELSE '0';

  temp_process1 : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      phase_0 <= '0';
    ELSIF clk'event AND clk = '1' THEN
      IF clk_enable = '1' THEN
        phase_0 <= phase_0_tmp;
      END IF;
    END IF;
  END PROCESS temp_process1;

  phase_0_tmp <= '1' WHEN count500 = to_unsigned(499, 9) AND clk_enable = '1' ELSE '0';

  temp_process2 : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      phase_1 <= '1';
    ELSIF clk'event AND clk = '1' THEN
      IF clk_enable = '1' THEN
        phase_1 <= phase_1_tmp;
      END IF;
    END IF;
  END PROCESS temp_process2;

  phase_1_tmp <= '1' WHEN count500 = to_unsigned(0, 9) AND clk_enable = '1' ELSE '0';

  enb <=  phase_all AND clk_enable;

  enb_1_500_0 <=  phase_0 AND clk_enable;

  enb_1_500_1 <=  phase_1 AND clk_enable;

END rtl;
```