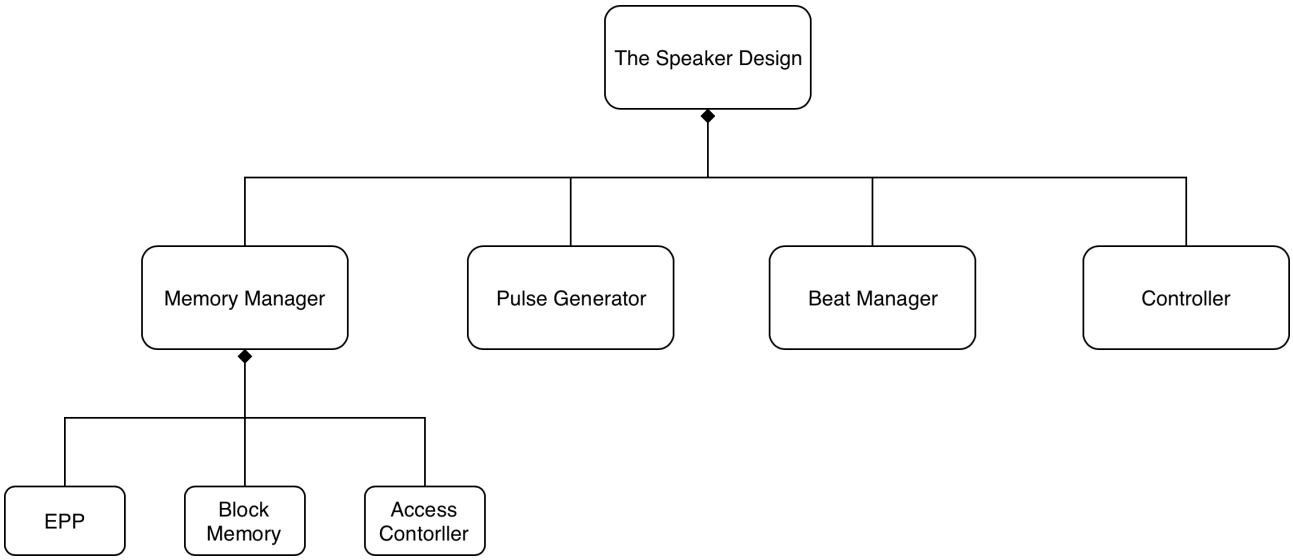
Break down and describe the tasks that are performed on the FPGA.



The VHDL parts is spited to 4 key components.

1. Memory controller
2. Pulse Generator
3. Beat Manager
4. Controller

The Memory controller controls all the behaviour of memory.

Behaviours:

1. giving an address, and return the note and note's length.
2. The access controller are also manager the data transfer from EPP register to the block memory.
3. The data writing will not be accept during running stage.

Design:

1. The reset state allows the EPP loading behaviour access and write data on the memory. The data writing will not be accept during running stage.
2. The access controller has the filly control on the data access. Which is decided by the reset trigger.
3. The controller read the EPP signal and data bus and give the correct write address and translate data to right form to block memory.

Implementation:

This part consist of 3 components(Block Ram, Epp, and Controller).

1. The Epp part is almost identical with the code of which provided from the user manual. We choose the EppReg1 as the input Epp register. After the data is loaded into this register, the Ready output will set to 1 and the data will be transfer through an output vector called memory data bus.

2. The controller will handle all the data in or out relevant with Ram. There is a state machine within this component. This module will wait for the ready signal to write data, Or reset signal to read data.
    1. For writing, The data write address will be set to 0 at initial stages. For each Epp data ready, this module will automatically increase the write address and sent is address with data to Block ram for one clock cycle.
    2. For Reading, This component will return the data to pulse generator and controller for the next move.

3. The blockRam is basically reading data from given address or writing data to given address.

The pulse generator is used to generate pulse according to the frequency of note.

behaviours:

1. Generate given note frequency to speaker.
2. Counting the beat pass from given beat pulse, and output right signal when the beat pass over note length.

design:

1. for each given note, The generator use a mux translate it to the right frequency hurdle. each clock cycle will add 1 on the counter, After the hurdle reached, the output of speaker changed.
2. There is a counter to monitor the time of note passing based on the same principle. but the pulse comes from the beat manager calculation.

Implementation:

2 components(word_mapping, clock2pulse) and 1 state machine

word_mapping module translate the input note to a given hurdle which equals 100mhz/(frequency*2).

clock2pulse module generate pulse and change the output to 0 or 1 after the counter reach the hurdle.

The state machine will count the input the beat pulse signal, once the the number of pulse is bigger than note length. the note done will be set to 1.

Beat Manager

behaviour:

output correct beat pulse to pulse generator based on given tempo.

design:

Using an estimate approach to shorten the tempo hurdle calculation.(e.g. 60 mph, the hurdle of trigger one beat pulse is the number of 60 counted within one sec). To smooth the transaction of tempo, the beat manager count the number of 60 within 4ms and scaled up to 1 sec, So It has a slight different with correct tempo(around 0.1% per second).

After calculation, running counter to generate pulse.

Implementation:

there is 2 components (beatsHurdleCalculator, clock2pulse(reused)).

beatsHurdleCalculator:

count the number of clock pass, when it equal to the frequency, hurdle number will be added by 1. Once 4 ms passed, set the calculation_done signal to 1, and output to the beats manager

clock2pulse:

It is exactly same as the pulse generator in the pulse generator. The input hurdler will be 8 bit right as the hurdle for 1 second.

Controller

behaviour:

1. based on the output signal of pulse generator (note_done), moving the address of memory, and reset the speaker counting.
2. based on the tempo change trigger. reset beat manager to achieve new tempo calculation.
3. Manager the data reading address from the memory block.

design:

- A FSM is based on the input signal. Output the correct signal to other modules. reset speak, reset tempo to do new tempo calculation, or move the memory address to next tempo

Implementation:

There is a state machine within this module. Triggering the given event based on the given input signal. There is 3 types of event(increase/ decrease tempo, or the address increment event).

increase/decrease tempo will reset the beats manager which will recalculate the tempo hurdle.

address increment event will be triggered when a node done signal is arrived.

Link everything Together.

When the reset is on, Memory manager will be able to load data from computer, and save it in the memory block. and reset reading address to 0.

When the reset is off. The controller output the data reading address. The output data will connect to the pulse generator.
pulse generator are generating pulse based on the mapped hurdle, Counting the number of beat which were passed, and setting the note_done to 1 when the beats' number is reach the length of this beat. The note_done will trigger the address moving, and read the new data from memory, and do it again.

What features and components of the Digilent Nexys board are you using and why?

Memory block and USB port for Epp data transfer.

How does the FPGA interface to the hardware and software? You may explain the protocols you used in interfacing.

Basically use the code of which provided by the manufacturer.

Describe the way that you are generating different frequencies and tempo.

It mention in the beats manager and pulse manager module.

Describe how you store songs transferred from PC to FPGA, what is the maximum song length (number of notes or size of the music file) your music player can play continuously.

mentioned in the memory manager. currently it's 63 Notes. But it can be increased to the maximum memory size(288kb/16bit) as required.

Describe the tasks that are performed in software, how they were written and how they interact.