UNIVERSITY OF TORONTO

ECE532: Orchestra Conductor Simulator

Final Design Report

Hardik Patel
Parul Seth
Marian Kotormus

10 April, 2015

Table of Contents

List of Figures

Overview

Motivation

The idea of the project came from existing games which are used to simulate playing musical instruments, such as Guitar Hero. We wanted to build something similar. Therefore we came up with the idea of turning gestures to audio and visual, to let the user experience conducting music in orchestra. Based on our survey of existing designs we have found only one application for iOS with similar goals, called Bravo Gustavo. In this case the user uses mobile device itself as conductor's baton, which may not be as comfortable due to its shape and size.

We decided FPGA will be suitable for this application since we can design a dedicated hardware components to speed up gesture tracking and recognition. We can also utilize external camera hardware to recognize gestures thus allowing user to make gestures with a real baton and a marker attached to it. As the last motivation, we chose to do this project because we wanted to create a real-world application with image and audio processing on FPGA, to implement all knowledge we received in the lectures practically and all at the same time create something which is efficient, fast and most of all user friendly.

Goals

As the main goal, we have established to simulate change in soundtrack tempo with a set of gestures that are used to control beats per measure in a real orchestra conducting. The functionality of the design would be to recognizing gesture by collecting maker position with a camera. After data is collected we would recognize the gesture and change the tempo of the soundtrack. To make design more user-friendly would will also provide visual feedback by outputting current gesture on screen. This would also indicate to the user where there gesture is invalid and whether they are performing the right gestures. Initial goal was to track 4 gestures which control the tempo, as can be found here.

Specifications

Hardware Requirements

- Stereo camera
- DAC - In the form of a 4th order Low Pass Butterworth Filter on the Nexys-4 board.

- Audio speakers
- Monitor

Constraints/Requirements

- Only one instrument is being played
- For gesture recognition, we have to move the hand slowly otherwise the number of points that will collected will not be enough to recognize the gesture
- We have to ensure that we do not exceed the memory provided on the board.
- The marker being used to do the gesture should a bright white LED.
- Using a dark background will help reduce the noise making it easy to track and recognize.
- The gestures fed in should not be over complicated, as the camera spee would not be able to support the motion tracking ip to give out enough points for recognition.

Block Diagram

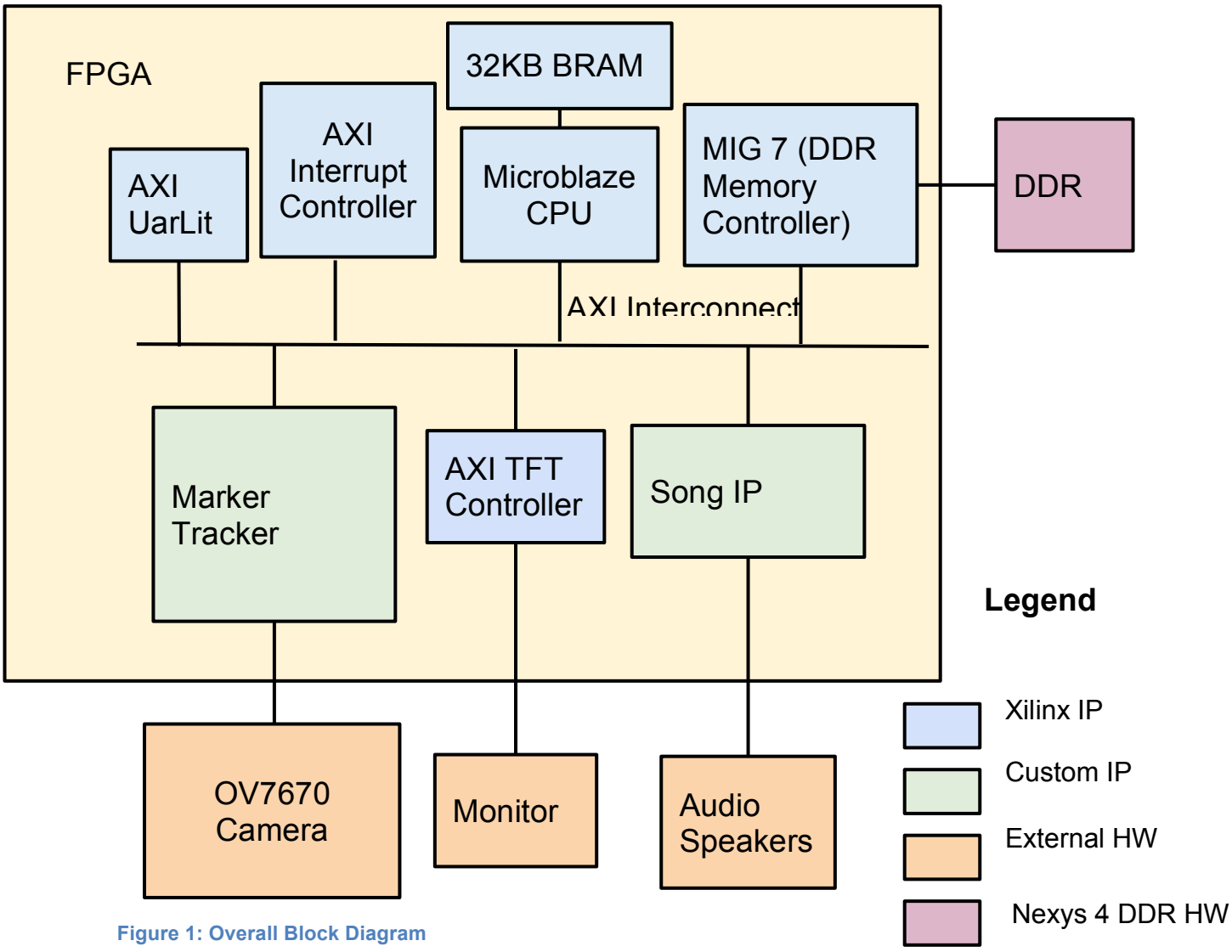

Figure 1: Overall Block Diagram

The design consists of 5 Xilinx IP cores, 2 custom cores, and a Microblaze CPU system with BRAM memory of 32KB.

IP Core listing

- **AXI Uarlite**: Xilinx IP used to output debug info on SDK console
- **AXI Interrupt Controller**: Xilinx IP for managing interrupts from AXI UartLite and MarkerTracker
- **Microblaze CPU + 32KB BRAM:** Xilinx IP cores which are parts of Microblaze processor system running our software.

- **MIG7 Memory Controller:** Xilinx IP to connect the design to 128Mb on board memory chip.
- **AXI TFT Controller:** Xilinx IP used for display on the monitor
- **MarkerTracker:** Custom IP core used to track marker with OV7670 camera
- **Song IP:** Custom IP used to generate the soundtrack

Final Design Outcome

We have been able to meet all our design requirements. We have been able to track marker using custom IP As a result our final block diagram does not differ from what was originally planned.

While debugging the system in SDK, it was found that the camera was not being able to give enough points for overly complicated gestures, therefore gesture recognition algorithm was not being able to produce complete sequences for the input feed hence making them difficult to recognize. Even though the MATLAB simulations for sample videos, showed that we could recognize those complicated gestures, it was difficult to get enough points for the algorithm to work for a few complicated gestures. So we had to use simpler gestures for the final design.

We planned to generate audio notes that sound like an actual musical instrument. An algorithm called Karplus-Strong string synthesis was implemented on MATLAB to simulate the desired outcome of the design. However, we faced a few complications while implementing the algorithm using Verilog and we decided to use the original method of audio generation. Another possible method of producing audio that was looked in to was using an audio codec module to read audio stored in the memory. The next step to improve the current design will be to use a sine wave instead of the square wave to improve the audio quality.

Project Schedule

1. Week 1
    a. Test the gesture recognition algorithm and design basic prototype on matlab
    b. Display some image on screen.
    c. Design basic audio output prototype which produces some sound
2. Week 2
    a. Write simple software application to drive inputs for hardware modules
    b. Draw 2D gesture on screen
    c. Produce audio with control on audio tempo
3. Week 3
    a. Recognize multiple gestures
    b. Recognize, translate and Draw 2D gesture on screen
    c. Play a song with control over tempo and create IP core for the module
4. Week 4
    a. Combine audio IP with gesture recognition peripheral
    b. Integrating all modules
5. Week 5
    a. Testing and integration. Demo.

In overall we have been able to follow our milestones except the major deviation from schedule due to MarkerTracker IP core not working as expected by Week 2. As a result, this postponed our integration testing.

Description of the Blocks

Marker Tracking

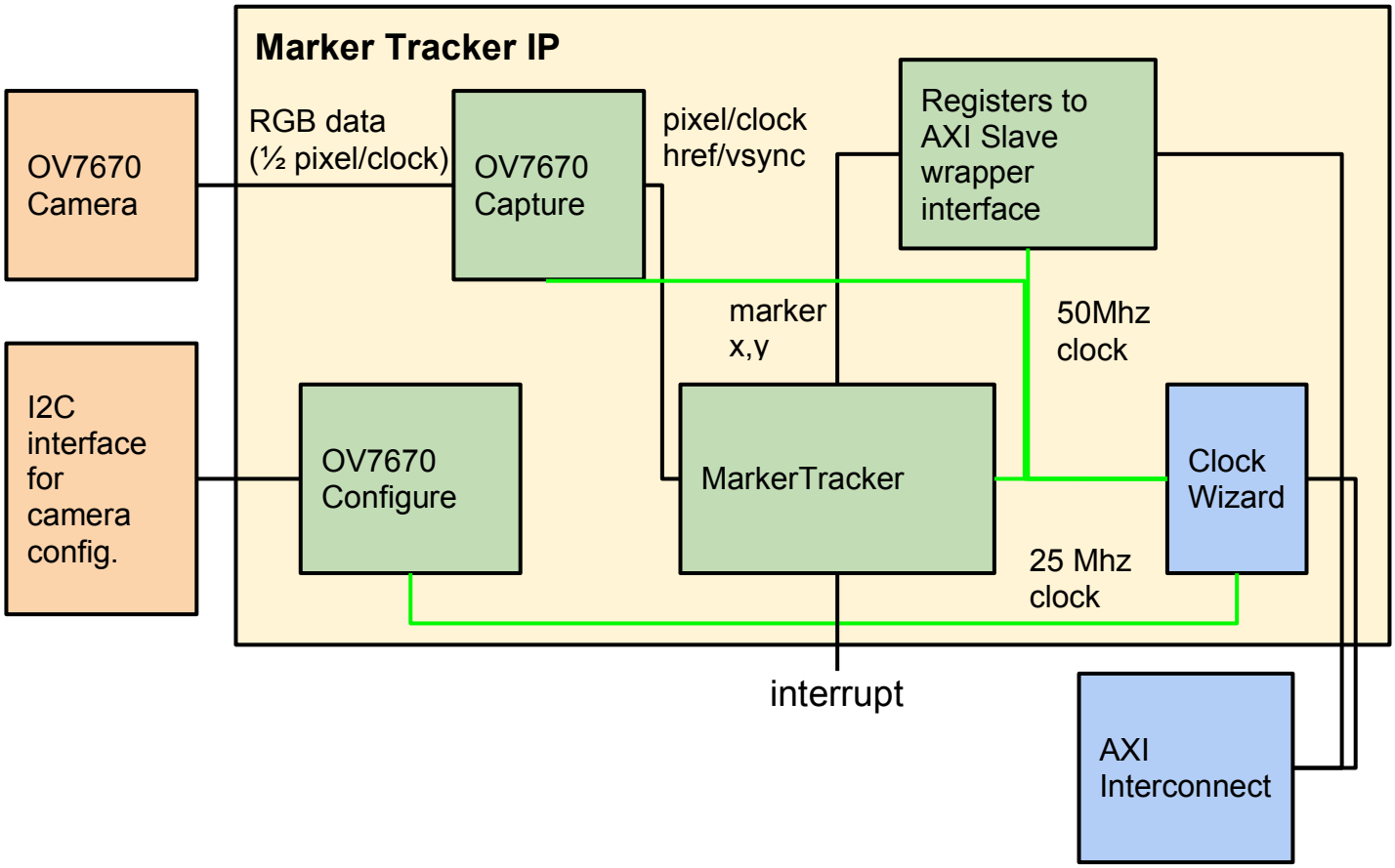

Figure 2: Marker Tracker Block Diagram

The marker tracker IP core consists of four Verilog modules which are used to configure the camera, collect data about the image in RGB format and produce x,y position which is the provided on AXI-bus in a set of registers.

The OV7670 Capture is a modified version of a module from a reference design posted on ECE532 Piazza portal. It reads camera pixel data 7 bits at a camera clock speed of 25 Mhz. Each complete pixel is passed to MarkerTracker along with href and vsync signals from the camera. MarkerTracker then keeps count of pixel positions within a frame and uses href and vsync signals to synchronize with each line and end of the frame respectively. It uses RGB value of the pixel to compare with the color specified by the user and counts only pixels that are equal or above (since we are using high intensity white light). The color value is specified by writing to the base register on AXI-Slave interface before starting the core, as is indicated in Figure 3. Once the color written user can write 1 to bit 0 of

base+1 (control register) and the marker tracking starts. After each processed frame MarkerTracker module will output x and y values of the marker, which captured into base+2 and base+3 respectively. It also asserts high value on the interrupt wire to indicate interrupt controller to issue MarkerTracker interrupt so that software can use the values. The core can be stopped by writing 0 to the lowest bit of base+1/control register. The interrupts can be enabled/disabled by writing 1 or 0 to bit 2 of the control register respectively.

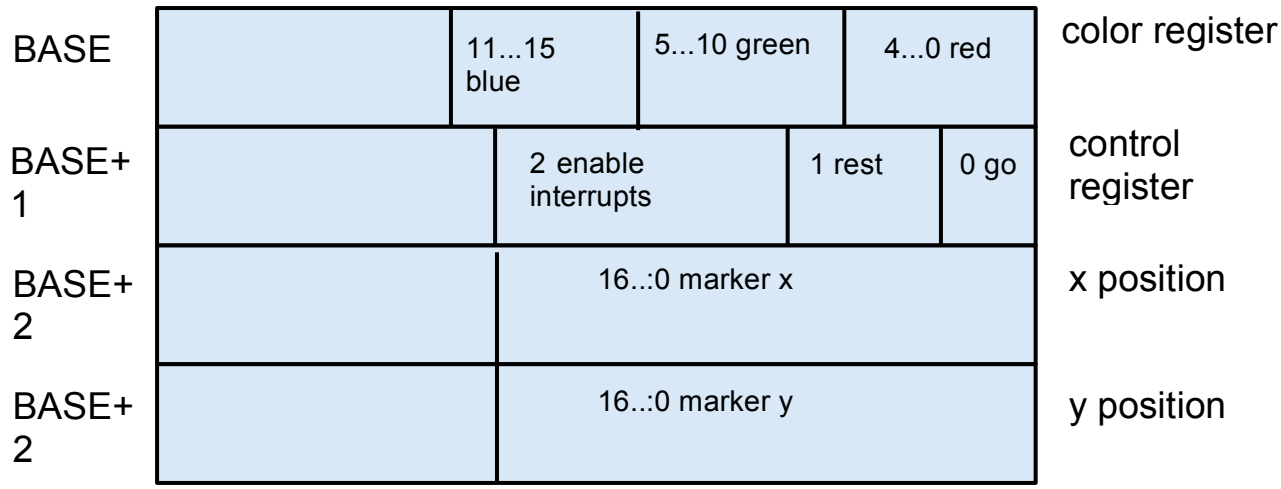

Figure 3: Marker Tracker Register Space

MarkerTracker IP also contains the OV7670 Configure module, which is an unmodified version of the same module from a reference example design of OV7670 camera on Piazza portal. User can configure the camera by writing 1 to bit 1 of control register, which also resets the entire core. OV7670 Configure module then pushes data to the camera over I2C interface.

The OV7670 Configuration module requires 25 Mhz clock while others can operate at 50Mhz, therefore the MarkerTracker IP has an embedded Xilinx Clock Wizard IP which takes as an input the CPU/100 Mhz clock and generates 25 and 50Mhz clock signals.

Gesture Recognition

The thing that makes a gesture unique is the relation between its X and Y coordinates. However to make it independent of the size, relative relation between X and Y coordinates should be taken. To do this, the points where the gesture goes from the higher part of the frame to the lower part or vice versa in Y direction and from left to right and vice versa in X direction. While this restricts

the gesture to be more or less aligned with the axes, it still can work on skewed angles as well. To remove the size dependency, we threshold the gesture around its midpoint and get a binary vector for the X and the Y coordinate.

This sequence is the size independent vector.

However since the gesture is manmade, it has to be made time invariant too. In order to do that, the binary signature undergoes a process where all multiple consecutive occurrences of either 1 or 0 are removed from sections. The sections are decided based on a zero crossing in either the X or the Y coordinate. Whenever a zero crossing is encountered, a new section is started and the value of that section is the value of the signature in that section. Removal of all duplicate entries, ensures that the algorithm does not depend on the number of frames and can do recognition irrespective of the number of X and Y coordinates and hence not on the recorded video's duration.

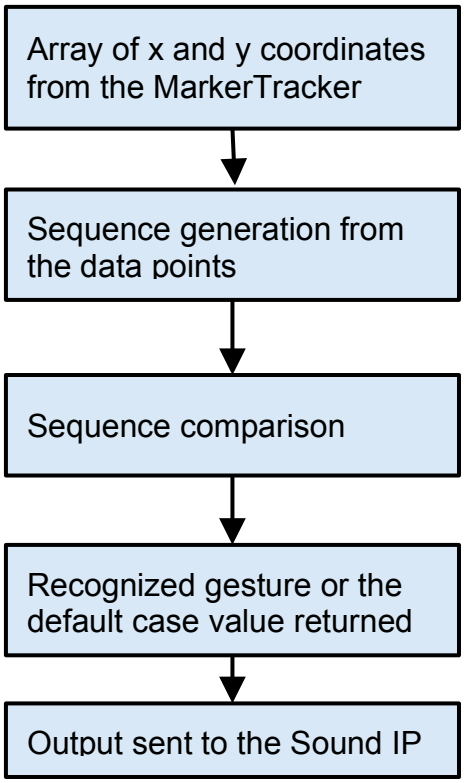

Figure 4: Gesture Recognition Flow Chart

The is the flow-chart in Figure 4 for data-flow for the gesture recognition algorithm. The data from the motion-tracking module is pushed into an array and this array is used for the gesture recognition. The data is then used by the sequence generating function that generates a sequence that is unique to the

gesture or the points collected by the motion-tracking module. This unique sequence is tested against the existing sequences that have been created for the gestures that were short-listed to be added in our design. Once the comparison is done, depending on the gesture it has recognized or the default case (not a recognizable gesture), the output value is returned and sent out to the sound IP module.

The sequencing function uses the coordinate positions of the X and Y points. It first find out the maximum and minimum of X and Y respectively and then finds the mid of the gesture for each direction. One that is done, it then makes an array and stores distance between other points and the mid-point we have found out. All the points with positive distance are assigned a value 1 and the other points with negative or zero difference are assigned 0. This array is then checked for transitions from 0 to 1 or vice versa in both X and Y directions. Whenever we find this transition in any one of the 2, then at this the value is stored for both of them, and a counter is incremented each time we save the values. These stored values give us sequences for the gesture in X and Y direction. The counter gives us the size of the sequence.

Once the sequences are generated, we compare them against the sequences that have been created by the same process for the desired gestures. This comparison takes into account the length of the sequence as one of the major parameters to check whether the sequence matches the existing sequences.

If the system matches the gesture to any one of the gestures, it returns the value associated with that gesture otherwise it will return the default value i.e the tempo value that is already playing.

Once we get the value from the compare function, the output is sent to Song IP ore to change the tempo according to the gesture recognized.

Song IP

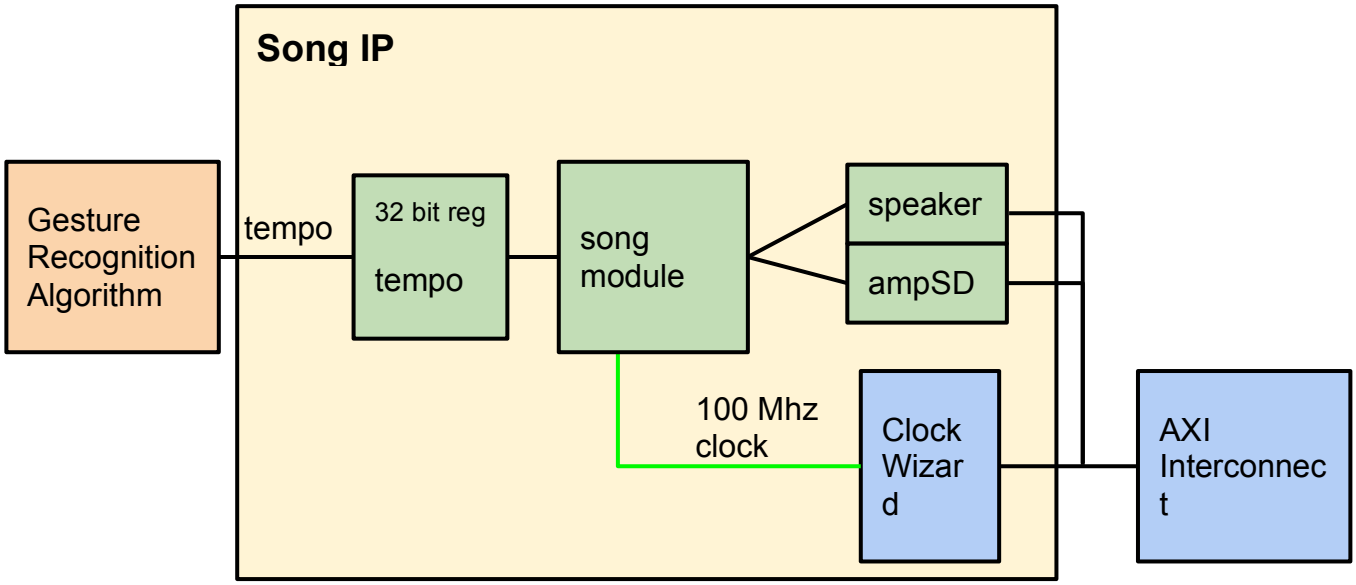

Figure 5: Song IP Block Diagram

The song IP is a custom IP core designed to generate audio. It consists of a Verilog module which generates song based on the tempo selected through the gesture recognition algorithm. The output from the gesture recognition algorithm is stored at the base address of the song IP.

The song IP uses a 100 MHz clock an input. A counter, “note_selector” is used to select the note to be played. Another down counter is set with the note that determines the period for which a particular note is to be played. The tempo register choses the note selection pattern.

Once a note is selected it sets another down counter that is equal to the ‘clock frequency/ note frequency / 2’. At each positive edge of the clock this counter counts down by one. When it reaches zero it inverts the output of the speaker pin. The speaker pin is the input of the low pass filter on the Nexys 4 FPGA Board. Figure 6, below, shows the low pass filter. The ‘AUD_PWM’ is the speaker pin. The AUD_SD pin is set high for higher gain. The ‘AUD_SD’ pin is indicated by the pin ‘ampSD’ in the song IP core.

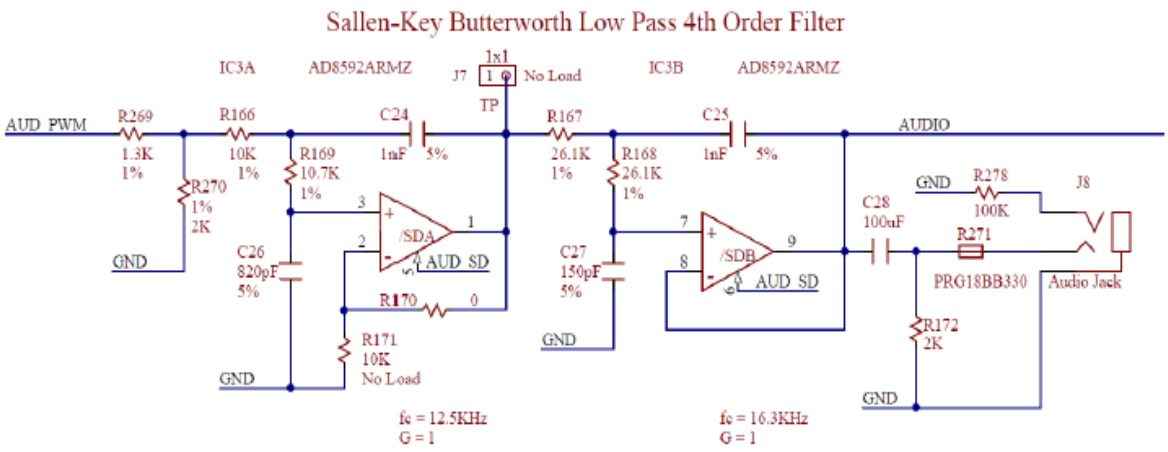

Figure 6: Nexys 4 Audio Amplifier Circuit

AXI TFT Controller

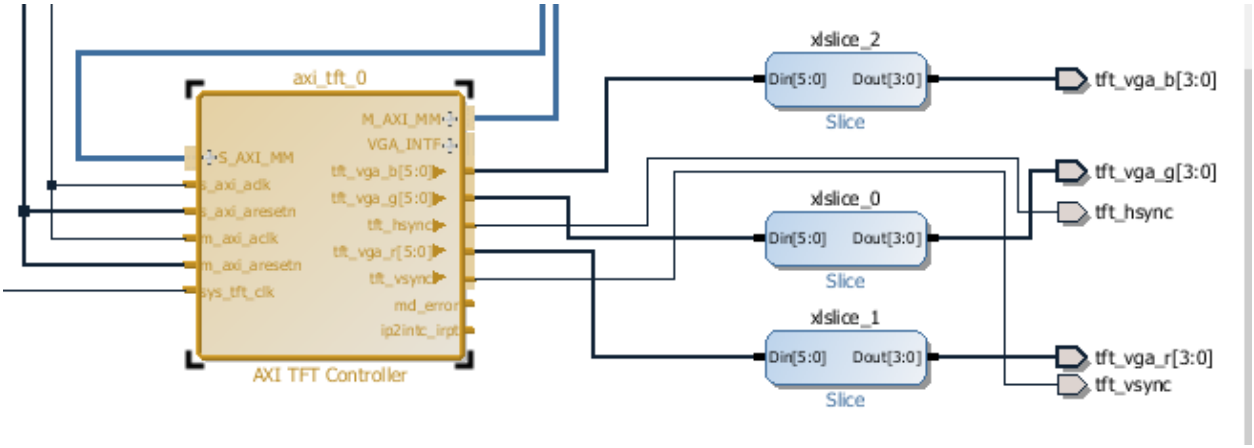

Figure 7: AXI TFT Controller Configuration

Xilinx IP configured for VGA with slice blocks on R, G and B components to truncate lower bits and convert to VGA444 format required for output on the Nexys 4 DDR board.

Microblaze Processor System with 32KB BRAM

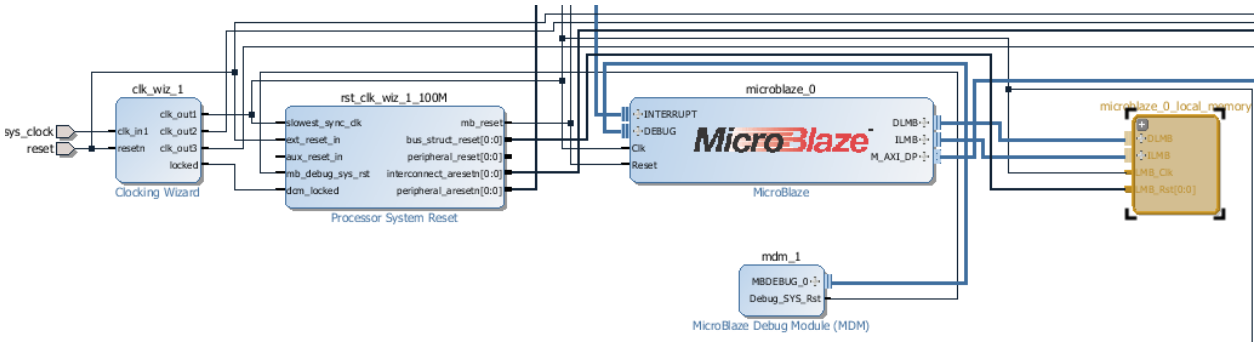

Figure 8: Microblaze processory system block diagram

Xilinx Microblaze processor IP which is running our software component of the design. The core is configured to run with 100 Mhz clock by the clock wizard. The rightmost block on Figure 8 is 32KB local BRAM memory used by Microblaze to store software instructions and data. The clock wizard is also generating 200 Mhz clock for MIG7 memory controller as well as 25 Mhz clock used by AXI TFT controller.

The software running on the Microblaze CPU is used to handle interrupts from MarkerTracker custom IP and collect marker positions in front of the camera. It then runs our custom algorithm, for gesture recognition logic, to decide the type of gesture a user has made. It then writes the sound tempo to the Sound custom IP, which produces the soundtrack at the specified tempo on the speakers. We also draw positions of marker on the TFT buffer to display them on the monitor via the VGA interface.

AXI Interrupt Controller

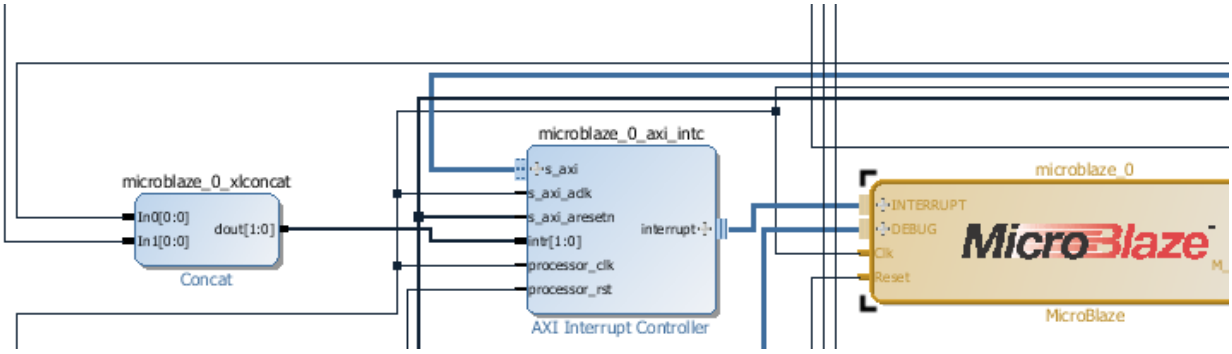

Figure 9: AXI Interrupt Controller block diagram

The AXI interrupt controller as in Figure 9, was configured with two interrupts (connected on left side of xconcat block) to handle interrupts from AXI Uarlite and MarkerTracker IP cores.

AXI Uarlite

This Xilinx IP core is used to print status of our application to UART console of the SDK for debugging purposes.

MIG7 Memory Controller

Enables the design to use 128Mb of DDR memory available on the Nexys4 DDR board. The memory is used to store a single VGA display buffer starting at base address of the DDR memory. This buffer is also being accessed by AXI TFT controller to draw gesture on the monitor.

Description of the Design Tree

Our project is hosted in the following Github repository:

https://github.com/mkotormus/G3_OrchestraConductorDemo

/src/IP directory
This directoty contains our custom IP cores where ***ov7670_marker_tracker_ip*** is the MarkerTracker IP core and ***song_ip_1.0*** is the song IP core. Before synthesizing the hardware bitstream this repository should be added to the main project's list of repositories in the project settings.

/src/ov7670_marker_tracker_use/ov7670_marker_tracker_use.xpr
This is the main project created with Vivado 2014.1. It contains all of the hardware as well as SDK project with our software component.

/src/ov7670_marker_tracker_use/ov7670_marker_tracker_use.sdk/SDK/SDK_Export/new/
This directory contains the "new" project which is Xilinx-C application created with SDK and contains our latest C-code in the **/src/helloworld.c** file.

/doc/
The following directory contains report as well a video demonstrating all of the implemented gestures.

Tips and Tricks

- Create prototypes in software (such as MATLAB) before implementing in HW
- Do simulation before running on HW
- Understand timing diagrams in the manual before attempting to do simulation of external hardware
- Use Logic Analyzers if the simulation does not work