ECE 532: Digital Hardware

Karaoke Machine

Group Report

Jen Pollock (991 855 738)
Frances Lau (991 654 900)

May 2005

Table of Contents

1. Overview

 The goal of this project was to create a karaoke machine using the MP3 player from the modules. The project needed to play an MP3 file mixed with the input from the microphone, at the same time as displaying lyrics synchronized to the music on the serial port. As well, it needed to respond to playback and volume controls implemented through buttons and switches. A system block diagram is shown in Figure 1.

Figure 1: System block diagram
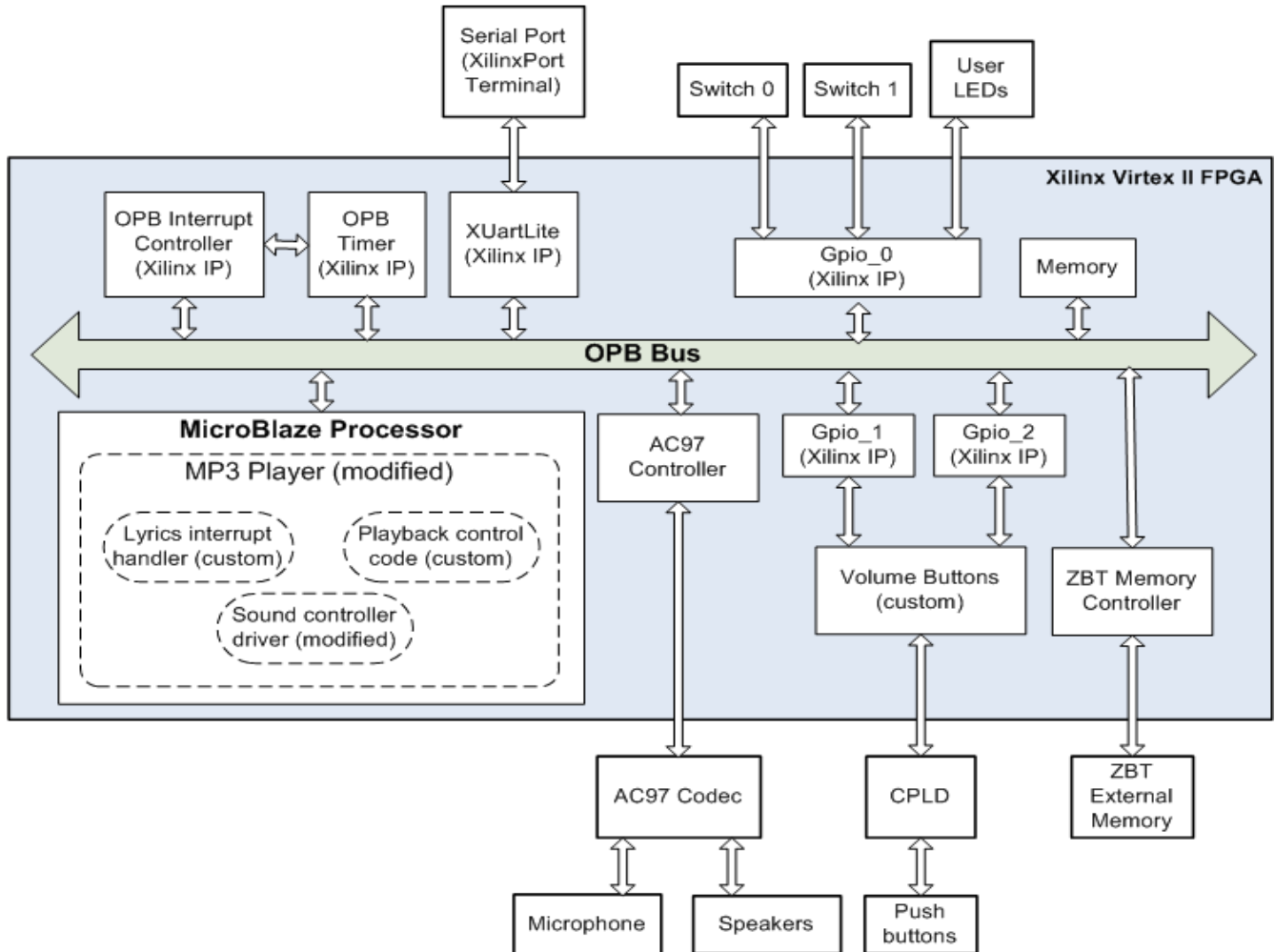
In addition to the MP3 player, which was heavily modified to perform the other tasks needed, several other IP blocks were used. The ZBT memory controller contained in the MP3 Player was unchanged. The GPIO, UARTlite, Interrupt Controller and Timer/Counter supplied by Xilinx were added and used without modifications. The AC97 controller provided with the MP3 player was modified slightly. A custom hardware block was created, based on the push button example from Xilinx, to interface with the volume buttons.

2. Outcome

The final project satisfies most of the original objectives and has almost all the fundamental features of a Karaoke Machine. It plays music, mixes microphone input with the music, displays synchronized lyrics, and has volume and playback controls.

The only feature that was not implemented was the reading of the music and lyrics file from compact flash. Although we started investigation for this milestone four weeks before schedule, due to poor documentation, we decided that it was not wise to spend an excessive amount of time on this extra feature, and we decided to concentrate on the core functions of this Karaoke Machine instead.

We originally intended to have one push button that functioned as "volume up", and one push button that functioned as "volume down", similar to traditional volume control buttons. However, after reviewing the documentation for the push buttons, we discovered that the push buttons were controlled by the CPLD. Push button status was only transmitted to the FPGA when the *ENTER* push button was pressed, so it was not possible to have "volume up" and "volume down" buttons. Therefore, we decided to implement a workaround by requiring that the user enter the volume level in a binary format. As shown in Figure 2 below, the user must choose one of the two yellow buttons (PB1 and PB2) on the left to indicate whether the volume of the song or the microphone is to be changed. Then, the user needs to enter the volume level in a binary format using PB8 to PB10 buttons. To send this selection to the FPGA, the user needs to press the Enter button.

Figure 2: Push buttons for volume control
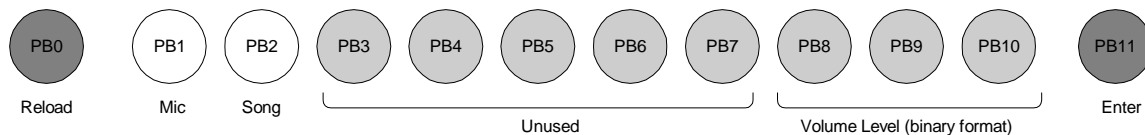
We also decided to use the user switches (SW0 and SW1) to control playback (pause and restart) instead of using the push buttons. This was done because we originally thought that there would not be enough push buttons to control both the volume and the playback. There were six unused push buttons in the final project, so in the future, the project could be modified to use push buttons for both playback and volume control if desired.

This project could be improved by implementing the feature to read the music and lyrics file from the compact flash. This would enable larger music files to be played and would shorten the time needed to download the music and lyrics file into the system. Also, video mode could be used for the push buttons so that the user would be restricted to only be able to select either PB1 (microphone) or PB2 (song), not both. This would prevent invalid input. More push buttons could be used to indicate the volume level so that all five volume control bits in the AC97 could be changed.

4

3. Description of Blocks

    The components of this project can be divided into existing (unmodified), modified, and custom components.

a) Existing components

These components were used without modifications.

- Microblaze processor
    - o Version: 2.00.a
    - o From the MP3 Player project, unmodified

- AC97 Controller
    - o Version: 1.00.a
    - o From the MP3 Player project, unmodified

- OPB_GPIO
    - o 3 instances of this block were created.
        - ▪ Instance #1: to connect to the User Switches and User LEDs
        - ▪ Instance #2: to connect to the port in the custom Volume Buttons hardware block that output the microphone volume
        - ▪ Instance #3: to connect to the port in the custom Volume Buttons hardware block that output the song volume

        A separate instance was needed for connecting to the User Switches and User LEDs because a different C_GPIO_WIDTH was needed. Separate instances were needed to connect to the microphone volume and song volume ports to eliminate the error "`multiple drivers found for connector`".

    - o Instance #1:
        - ▪ Version: v3.01.a
        - ▪ Parameters:
            - • C_GPIO_WIDTH = 4
        - ▪ Ports:
            - • GPIO_IO (External; Net name: opb_gpio_0_GPIO_IO; Range set to [0:3] in the External Ports Connections list). In the system .ucf file, the following code was added to connect this port to the switches and LEDs:

```
# User LED1
Net opb_gpio_0_GPIO_IO<0> LOC=B27;
# User INPUT1
Net opb_gpio_0_GPIO_IO<1> LOC=F14;
# User LED0
Net opb_gpio_0_GPIO_IO<2> LOC=B22;
# User INPUT0 (can use this now, b/c not used
for reset anymore)
```

```
Net opb_gpio_0_GPIO_IO<3> LOC=D10;
```

In the C program the following code was added to refer to the switches and LEDs. Note that the system is big endian.

```
#define LED1    0x8
#define SWITCH1 0x4
#define LED0    0x2
#define SWITCH0 0x1
```

- o Instance #2:
    - ▪ Version: v3.01.a
    - ▪ Parameters:
        - • C_GPIO_WIDTH = 3
        - • C_ALL_INPUTS = 1
        - • C_IS_BIDIR = 0
    - ▪ Ports:
        - • GPIO_in (Internal; Net name: opb_gpio_1_GPIO_in). This port is connected to the volume_buttons mic_volume port.

- o Instance #3:
    - ▪ Version: v3.01.a
    - ▪ Parameters:
        - • C_GPIO_WIDTH = 3
        - • C_ALL_INPUTS = 1
        - • C_IS_BIDIR = 0
    - ▪ Ports:
        - • GPIO_in (Internal; Net name: opb_gpio_2_GPIO_in). This port is connected to the volume_buttons song_volume port.

- Interrupt controller
    - o Version: 1.00.c
    - o Ports:
        - ▪ Intr (Internal, Net name; timer_to_intc). This port is connected to the Interrupt port of the OPB_Timer
        - ▪ Irq (Internal, Net name; intc_to_myblaze)

- OPB_Timer
    - o Version: 1.00.b
    - o Ports: Interrupt (Internal; Net name: timer_to_intc). This port is connected to the Intr port of the OPB_INTC

- OPB_UARTlite
    - o Version: 1.00.b
    - o Parameters:
        - ▪ C_Use_Parity = 0
        - ▪ C_Odd_Parity = 0

- C_Clk_Freq = 100000000
- C_Baudrate = 9600
- C_Data_Bits = 8

Note that the clock frequency is 100MHz, not 27MHz because the system clock for the MP3 Player Project is 100MHz.
- Ports:
  - OPB_Clk = sys_clk (Internal)
  - RX (External)
  - TX (External)
  - Req_to_Send_pin (External; Connected to net_gnd)
- In Software Platform Settings, the stdin and stdout had to be changed to this instance of OPB_UARTlite

b) Modified components

The MP3 player from tutorial module 13 was used and heavily modified. Hardware was added to interface with the buttons, switches, and serial port, and time the lyrics display and generate interrupts for the lyrics. As well, code was added to the main file (madlld.c) to set up and make use of the added hardware. As a lot of small polling tasks added to the main decoding loop, the sound became choppy. To fix this, only one of the added tasks was executed in a given loop, and some loops didn't execute any of them. Also, the code was optimized to reduce delays to improve the quality of the sound.

Modifications for the mixing of microphone input and pre-recorded music

We added code in the `init_sound` function in the `sound_controller.c` file to enable the AC97 Codec to mix the microphone input with the pre-recorded song. The Mic Volume Register (Register 0x0E) was set to 0x0040 to unmute the microphone and give a 20dB boost to the microphone volume. The PCM Out Vol Register (Register 0x18) was set to 0x1818 to attenuate the music so that the microphone could be heard. Values for the registers were determined by consulting the AC97 Codec LM4549 datasheet.

Modifications for playback control

User Switch 1 is used to restart the song. The status of this switch is polled in the main decoding loop using OPB_GPIO, and if the switch is down, the program enters a `while` loop and the timer for the lyrics is stopped. Once the switch is up again, the song is restarted. This is accomplished by setting the variable `first_time` to 1, calling the `initializeLyrics()` function, and restarting the timer for the lyrics.

User Switch 0 is used to pause the playback of the song. Switch 0 is normally used for system reset, but this was changed in this project. Details regarding this are on page 9 in the Custom Components section of this report. The status of this switch is polled using OPB_GPIO, and if the switch is down, the program enters a `while` loop and

the timer for the lyrics is stopped. Once the switch is up again, the song continues and the timer for the lyrics interrupts is restarted.

<u>Modifications for displaying lyrics</u>

One instance of each of a timer/counter and an interrupt controller was added to the main file (madlld.c), and initialized and set up appropriately in the code, as described in the Existing Components section of this report. In the initialization section of the code, the first line of lyrics and the time that the lyrics should be displayed are read from memory. The timer/counter is set up to generate an interrupt at the time that the first line of lyrics should be displayed. The interrupt handler displays the line of lyrics that had been read in, reads the next line and the time that it should be displayed from memory, and sets up the timer/counter to generate an interrupt for the next line. Both the interrupt handler and the initial setup are capable of displaying multiple lines set to display at the same time. All display is on the serial port, using the UARTlite.

<u>Modifications for volume control</u>

The OPB_GPIO is polled in the main decoding loop in madlld.c to determine the status of the volume_buttons hardware block. These values are used to set microphone and song volume.

c) Custom components

<u>Volume_buttons hardware block (volume_buttons.v)</u>

This Verilog block is based on `PB_SCAN_DATA_IN.v` from the Xilinx website ([http://www.xilinx.com/products/boards/multimedia/ examples.htm](http://www.xilinx.com/products/boards/multimedia/examples.htm), last update 2001/06/19). It decodes the push button status from data transmitted by the CPLD on the Microblaze board. The CPLD shifts the data out on `PB_DATA_P` on the falling edge of `PB_CLOCK_P` and latches the data on the rising edge of `PB_CLOCK_P`. The data is transferred to a latch every time `data_bit_count` reaches 0xA.

As shown in Figure 2 in the Results section, PB 1 and PB2 are used to indicate whether the volume for the microphone or the song is to be changed. PB8 to PB10 are used to select the volume level. Therefore, this module transfers the status of PB8 to PB10 to the mic_volume output when PB1 is selected and to the song_volume output when PB2 is selected. It ignores PB3 to PB7. The user specifies the volume level in binary format using PB8 to PB10, with 000 as the lowest volume and 111 as the highest volume.

This module is designed to be connected to the OPB Bus through two instances of OPB_GPIO. Since we only needed to read the status of two registers, we decided that this simpler approach would be better because we would not have to deal with the OPB protocol ourselves, and this gave us a much higher chance of success. Since these ports are internal, we had to set C_ALL_INPUTS = 1 to specify that we do not want to use

GPIO_IO for input instead of GPIO_in. We also had to set C_IS_BIDIR = 0 to eliminate the GPIO_IO, GPIO_d_out, and GPIO_t_out ports. Details are in the OPB_GPIO datasheet.

The reset for this module is active low (reset_b), so it can be connected to `extend_dcm_reset`, which is active low as well.   The startup port should be connected to net_vcc if it is not already connected. Detailed instructions for adding this custom Verilog core to the MicroBlaze system are in Appendix A and in the README. In the future, this core can be modified and used for other projects that require push buttons.

Simulations were performed using Modelsim. A testbench is found in sim/testbench.v. `PB_CLOCK_P` only turns on when data is being shifted out. Therefore, this testbench tries to simulate this by having one signal `top_data_shifting` that is on when data is being shifted out, and a clock that is continuously on (`top_PB_CLOCK_P_continuous`). Then, `top_PB_CLOCK_P = top_PB_CLOCK_P_continuous & top_data_shifting`. Note that `mic_volume` and `song_volume` hold their previous value until a new value is set. Figure 3 shows the simulation output.

Figure 3: Simulation of volume_buttons block
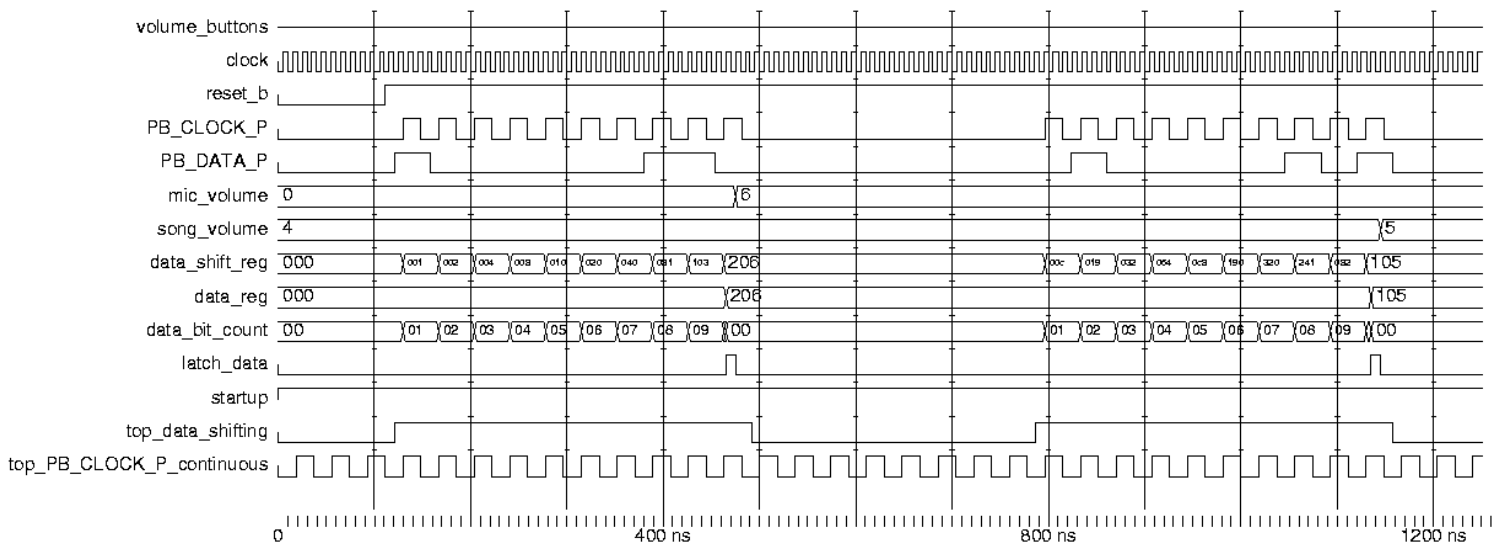
Switches for playback control

Two switches are needed so that the user can restart and pause the song. However, Switch 0 is normally used for system reset. For this project, we connect the system reset to the CPLD reset signal `extend_dcm_reset` so that Switch 0 could be used. The procedure for doing this is:

1. Delete the net `ext_reset` in the system.ucf file that was originally connected to User Switch 0 (pin D10).

2. In all locations in the system.ucf and system.mhs file that refer to `ext_reset`, replace this with `extend_dcm_reset`, the net that connects to the CPLD reset signal.
3. `extend_dcm_reset` is active low while `ext_reset` was active high. Therefore, go to the Add/Edit Cores dialog, and check each peripheral to see if it has the `C_EXT_RESET_HIGH` parameter. If it does, add this parameter and set its value to 0. Now, the entire system has an active low reset.

4. Description of Design Tree

The top level folders in the design tree are described in Table 1 below. Further details are in the README file.

Table 1: Design Tree

Folder name	Description
xps	XPS design
volume_buttons	Simulation and rtl files for custom volume_buttons hardware core
songs	Sample karaoke songs
doc	Documentation

Appendix A: README: Instructions on how to add the volume_buttons Verilog core to your XPS project

1. Copy the entire volume_buttons directory into your pcores directory. The name of the folder should remain "volume_buttons".
2. Look through the directory structure of the volume_buttons core. There is a data folder inside the volume_buttons directory. Inside this folder are three files. The volume_buttons_v2_1_0.mpd file is a Microprocessor Peripheral Description file, which defines the ports and parameters in the module. The volume_buttons_v2_1_0.pao file is a Peripheral Analyze Order file that contains references to all the HDL files. Note that the references do not include the extension ".v". If more HDL files need to be added in the future, the order of the references need to reflect the order required to resolve dependencies (i.e., with the top-level design listed last). The volume_buttons_v2_1_0.bbd file is a Black-Box Definition file that includes references to the netlist files in the netlist directory. There are no netlist files in this project. For more information on these files and on integrating a Verilog design into a MicroBlaze system, see Module m9 for EDK 6.2i. Note that the files are named in a particular way so that XPS will be able to detect the core.
3. Open your XPS project. If it was already open, you must restart XPS so that the list of cores is refreshed.
4. Open the Add/Edit Cores dialog.
5. The volume_buttons core should appear in the list of peripherals. Add the volume_buttons core.
6. This core is not connected directly to the OPB Bus, so you don't need any bus connections and you don't need to specify addresses. There are also no parameters for this core.
7. In the Ports tab, add the clock, reset_b, PB_CLOCK_P, PB_DATA_P, mic_volume, and song_volume ports. Make the PB_CLOCK_P and PB_DATA_P ports external. Leave the remainder of the ports as internal. Connect clock to sys_clk and reset_b to extend_dcm_reset (an active low reset from the CPLD).
8. Check if there is already a port called "startup" in the External Ports Connections list. If there is, you do not need to add the startup port. If there is not, add the startup port and make it external. IMPORTANT: make sure the startup port is connected to net_vcc.
9. Now, you must add two instances of the OPB_GPIO to interface with the volume_buttons core. Connect these as a slave to the OPB bus and specify address ranges, with a size of 512.
10. In the Ports tab, add the GPIO_in port as an internal port for each instance. You can leave the net names as opb_gpio_1_GPIO_in and opb_gpio_2_GPIO_in.
11. Change the following parameters for each instance:
    - C_GPIO_WIDTH = 3
    - C_ALL_INPUTS = 1
    - C_IS_BIDIR = 0

Since this port is an internal port, we need to set C_IS_BIDIR = 0 to specify that we do not want to use GPIO_IO for input instead of GPIO_in. We also need to set C_IS_BIDIR = 0 to eliminate the GPIO_IO, GPIO_d_out, and GPIO_t_out ports. See the OPB_GPIO datasheet for further details.

12. In the Internal Ports Connections list, connect the volume_buttons core to the OPB_GPIO by changing the net name for mic_volume to opb_gpio_1_GPIO_in and the net name for song_volume to opb_gpio_2_GPIO_in.

13. In your C program, add
```
#define MIC 0x7
#define SONG 0x7
```

14. Add the following code to initialize the OPB_GPIO instances in your C program:
```
XStatus GpioStatus;
XGpio Gpio_micvol;
XGpio Gpio_songvol;

GpioStatus = XGpio_Initialize(&Gpio_micvol,
XPAR_OPB_GPIO_1_DEVICE_ID);
if (GpioStatus != XST_SUCCESS)
    {
        printf("GPIO 1 initialization error\n\r\r");
    }

GpioStatus = XGpio_Initialize(&Gpio_songvol,
XPAR_OPB_GPIO_2_DEVICE_ID);

if (GpioStatus != XST_SUCCESS)
    {
        printf("GPIO 2 initialization error\n\r\r");
    }


/* Set the direction for all signals to be inputs*/
XGpio_SetDataDirection(&Gpio_micvol, 1, MIC);
XGpio_SetDataDirection(&Gpio_songvol, 1, SONG);
```

15. Reading the status of mic_volume and song_volume is accomplished using the code:
```
XGpio_DiscreteRead(&Gpio_micvol, 1 ) & MIC;
XGpio_DiscreteRead(&Gpio_songvol, 1 ) & SONG;
```