REFERENCES

1. Altera Corp. (2006). Introduction to Quartus II, http://www.altera.com/literature /manual/intro_to_quartus2.pdf, 15 Feb 2007

2. Altera Corp. (2006). Altera UP2 Education Board, http://www.altera.com/education /univ/materials/boards/unv-up2-board.html, 15 Feb 2007

3. Altera Corp. (2006). University Program UP2 Education Kit, http://www.altera.com /literature/univ/upds.pdf, 15 Feb 2007

4. Altera Corp. (2006). Quartus II Introduction Using Verilog Design, http://www.altera.com /education/univ/materials/manual/labs/tut_quartus_intro _verilog.pdf, 15 Feb 2007

5. fpga4fun.com. (2007). FPGA Project, http://www.fpga4fun.com/MusicBox.html, 10 Jan 2007

6. fpga4fun.com. (2007). FPGA Project, http://www.fpga4fun.com/MusicBox1.html, 10 Jan 2007

7. fpga4fun.com. (2007). FPGA Project, http://www.fpga4fun.com/MusicBox2.html, 10 Jan 2007

8. fpga4fun.com. (2007). FPGA Project, http://www.fpga4fun.com/MusicBox3.html, 10 Jan 2007

9. Introduction to FPGA design, http://www.aps.anl.gov/epics/meetings/2006-06/Embedded_Controllers/P0_Feedback.ppt, 15 Feb 2007

10. Introduction to Verilog, http://www.doe.carleton.ca/~shams/97350/ PetervrlK.pdf, 15 Feb 2007

11. Padmanabhan, T.R. and Sundari, B.B.T. (2004). Design Through Verilog HDL (A John & Sons Inc., Publication)

12. Verilog Dot Com. (2006)., http://www.verilog.com/, 15 Feb 2007

13. Verilog Hardware Description Language, http://www.ece.cmu.edu/~thomas /VSLIDES.pdf, 15 Feb 2007

14. Wikipedia. (2006). Field-programmable gate array, http://en.wikipedia.org /wiki/FPGA,15 Feb 2007

15. Xilinx. (2006). Introduction to Verilog v8.1, http://www.xilinx.com/support /training/abstracts/lang12000-ilt.pdf, 15 Feb 2007

16. Zeidmanconsulting. (2006). Introduction to Verilog, http://www.zeidmanconsulting. Com/documents/Introduction to Verilog.sample.pdf, 15 Feb 2007

APPENDICES

APPENDIX A:

VERILOG code for music1

```
module music1 (clk,q);
input clk;
input o1;
output q;

reg [15:0] counter;

always @(posedge clk)
                counter <= counter + 15'b1;

        assign q = counter[15];

endmodule
```

APPENDIX B:

VERILOG code for music2

```
module music2(clk, q);
input clk;
output q;
parameter clkdivider = 25000000/440/2;

reg [23:0] tone;
always @(posedge clk) tone <= tone + 24'b1;

reg [14:0] counter;
always @(posedge clk)
        if(counter==0)
                counter <= (tone[23] ? clkdivider - 1'b1 : clkdivider/2-1);
        else
                counter <= counter - 1'b1;

reg q;
always @(posedge clk)
        if(counter==0)
                q <= ~q;

endmodule
```

APPENDIX C:

VERILOG code for music3

```
module music3(clk, q);
input clk;
output q;

reg [22:0] tone;
always @(posedge clk) tone <= tone + 1'b1;

wire [6:0] ramp = (tone[22] ? tone[21:15] : ~tone[21:15]);
wire [14:0] clkdivider = {2'b01, ramp, 6'b000000};

reg [14:0] counter;
always @(posedge clk)
	if(counter==0)
		counter <= clkdivider;
	else
		counter <= counter - 1'b1;

reg q;
always @(posedge clk)
	if(counter==0)
		q <= ~q;

endmodule
```

APPENDIX D:

VERILOG code for music4

```
module music4(clk, q);
input clk;
output q;

reg [27:0] tone;
always @(posedge clk)
        tone <= tone + 1'b1;

wire [6:0] fastsweep = (tone[22] ? tone[21:15] : ~tone[21:15]);
wire [6:0] slowsweep = (tone[25] ? tone[24:18] : ~tone[24:18]);
wire [14:0] clkdivider = {2'b01, (tone[27] ? slowsweep : fastsweep), 6'b000000};

reg [14:0] counter;
always @(posedge clk)
        if(counter==0)
                counter <= clkdivider;
        else
                counter <= counter - 1'b1;

reg q;
always @(posedge clk)
        if(counter==0)
                q <= ~q;
endmodule
```

APPENDIX E:

VERILOG code for music5

```
module music5(clk, q);
input clk;
output q;

reg [27:0] tone;
always @(posedge clk)
        tone <= tone + 1'b1;

wire [5:0] fullnote = tone[27:22];

wire [2:0] octave;
wire [3:0] note;
divide_by12 divby12(.numer(fullnote[5:0]), .quotient(octave), .remain(note));

reg [8:0] clkdivider;
always @(note)
case(note)
  4'b0: clkdivider = 9'b100000000 - 1'b1; // A
  4'b0001: clkdivider = 9'b111100011 - 1'b1; // A#/Bb
  4'b0010: clkdivider = 9'b111001000 - 1'b1; // B
  4'b0011: clkdivider = 9'b110101111 - 1'b1; // C
  4'b0100: clkdivider = 9'b110010110 - 1'b1; // C#/Db
  4'b0101: clkdivider = 9'b110000000 - 1'b1; // D
  4'b0110: clkdivider = 9'b101101010 - 1'b1; // D#/Eb
  4'b0111: clkdivider = 9'b101010110 - 1'b1; // E
  4'b1000: clkdivider = 9'b101000011 - 1'b1; // F
  4'b1001: clkdivider = 9'b100110000 - 1'b1; // F#/Gb
  4'b1010: clkdivider = 9'b100011111 - 1'b1; // G
  4'b1011: clkdivider = 9'b100001111 - 1'b1; // G#/Ab
  4'b1100: clkdivider = 1'b0; // should never happen
  4'b1101: clkdivider = 1'b0; // should never happen
  4'b1110: clkdivider = 1'b0; // should never happen
  4'b1111: clkdivider = 1'b0; // should never happen
endcase

reg [8:0] counter_note;
```

```
always @(posedge clk)
	if(counter_note==0)
		counter_note <= clkdivider;
	else
		counter_note <= counter_note - 1'b1;

reg [7:0] counter_octave;
always @(posedge clk)
if(counter_note==0)
begin
 if(counter_octave==0)
  counter_octave <= (octave==0?8'b11111111:
                                        octave==1?8'b01111111:
                                        octave==3?8'b00111111:
                                        octave==3?8'b00011111:
                                        octave==4?8'b00001111:7);
                                        //octave==3'b111);
 else
  counter_octave <= counter_octave - 1'b1;
end

reg q;
always @(posedge clk)
	if(counter_note==0 && counter_octave==0)
		q <= ~q;
endmodule
```

APPENDIX F:

VERILOG code for divide_by12

```
module divide_by12(numer, quotient, remain);
input [5:0] numer;
output [2:0] quotient;
output [3:0] remain;

reg [2:0] quotient;
reg [3:0] remain_bit3_bit2;

assign remain = {remain_bit3_bit2, numer[1:0]}; // the first 2 bits are copied through

always @(numer[5:2]) // and just do a divide by "3" on the remaining bits
case(numer[5:2])
  0: begin quotient = 0; remain_bit3_bit2 = 0; end
  1: begin quotient = 0; remain_bit3_bit2 = 1; end
  2: begin quotient = 0; remain_bit3_bit2 = 2; end
  3: begin quotient = 1; remain_bit3_bit2 = 0; end
  4: begin quotient = 1; remain_bit3_bit2 = 1; end
  5: begin quotient = 1; remain_bit3_bit2 = 2; end
  6: begin quotient = 2; remain_bit3_bit2 = 0; end
  7: begin quotient = 2; remain_bit3_bit2 = 1; end
  8: begin quotient = 2; remain_bit3_bit2 = 2; end
  9: begin quotient = 3; remain_bit3_bit2 = 0; end
  10: begin quotient = 3; remain_bit3_bit2 = 1; end
  11: begin quotient = 3; remain_bit3_bit2 = 2; end
  12: begin quotient = 4; remain_bit3_bit2 = 0; end
  13: begin quotient = 4; remain_bit3_bit2 = 1; end
  14: begin quotient = 4; remain_bit3_bit2 = 2; end
  15: begin quotient = 5; remain_bit3_bit2 = 0; end
endcase

endmodule
```