Implementation of a Controllable Function Generator

Sarah Ferguson
Gavin Darcey
December 10, 2010

Standard function generators, when implemented in Verilog, are not easily controlled or measured. The signals they generate are typically set to be a predetermined amplitude, waveform, and frequency. This proposal outlines the details and design methodology for the creation of a controllable function generator with unique features to address these flaws. The system has two primary modules, one to create and control the analog output, and the other to construct a digital video representation of the analog waveforms. Four waveforms were generated, each with controllable amplitude and frequency, as well as duty cycle, when applicable. The steps taken to design each aspect of the function generator are described, as well as the integration of all subsystems present in the design. The system was implemented in Verilog, then tested and debugged on an FPGA. It was found to be possible to implement the function generator using the FPGA, a computer monitor, and a digital to analog convertor; the end result being a fully functional prototype.

Table of Contents

List of Figures

1. OVERVIEW

This project implements a basic function generator capable of creating periodic square, sine, and sawtooth (triangle) waveforms. The output is very similar to that found on a standard lab function generator, and this implementation is unique in that it provides controllable frequency and amplitude for each of the 4 waveforms. In addition to the generation and basic control of the wave through pushbuttons on the labkit, this design implements video output for additional measurement display.

There are two major components to this project – the wave generation and the video output. The wave type is selected by the user via the pushbuttons on the labkit. Wave generation is handled exclusively by one functional state machine (FSM) that accepts user input – wave type, frequency, and amplitude, and generates 8 bits of data that are fed to an 8-bit digital to analog converter in order to create the proper analog waveform. Due to the mechanical "bounce" seen on the labkit buttons when depressed, all button inputs are debounced and synchronized with the necessary clocks before being interpreted by the FSM. At power-up, the frequency and amplitude are standardized to start at 250 Hz and 2.5V peak to peak, respectively, and the generator begins with synthesis of a square wave. The frequency and amplitude can then be adjusted up and down by the user, via the labkit pushbuttons. The FSM maintains measurements of the current frequency and amplitude, as well as the wave type, in order for this information to be properly displayed on the video screen.

The digital to analog converter used is the Analog Devices AD7224, which somewhat limited the maximum frequencies attainable due to the inherent device limitations in high-speed switching. Nonetheless, frequencies of up to 16 kHz are obtainable when generating square waves. Appendix D contains the data sheet for this device; for this design the device was wired to operate in Unipolar Output Operation mode, with all input registers transparent, for simplicity.

The video display is not necessarily a real-time view of the waveform; such a measurement can be quickly verified with an oscilloscope, as was done during the debugging of the project. Instead, it is a real-time summary of what is being generated and what operations are being performed on the wave – it illustrates wave type, and shows on screen the relative increases and decreases in amplitude and frequency. The display component also shows on-screen the maximum amplitude being output by the DAC, and was intended to show the frequency as well, but this portion was not fully debugged. The video encoding used is XVGA, which necessitates the 65 MHz system clock that all other modules rely on.

The two components – wave generation and display – pass information about the wave between them, in order to maintain consistency. This block diagram provides a glimpse of the system as a whole, and show the inputs and outputs to the major modules.

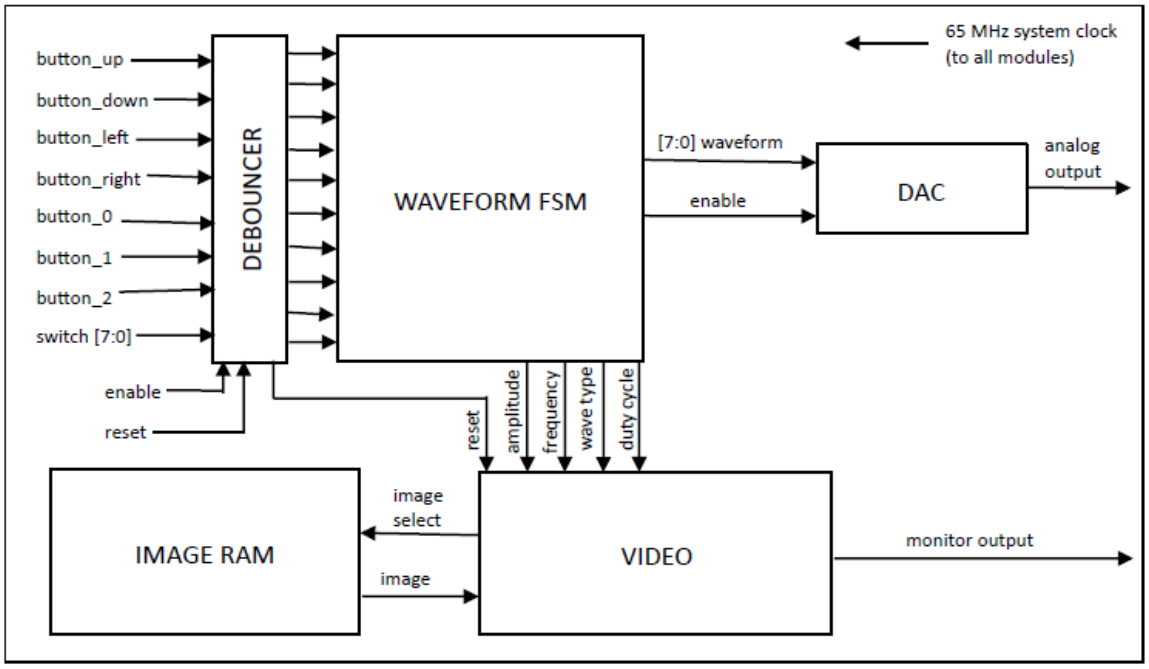


Figure 1: Block diagram of system

2. DESCRIPTION

2.1 Implementation

This section describes in detail each of the major Verilog modules used in the implementation of the project.

2.1.1 Waveform FSM:

The Waveform FSM module is the sole control module behind the digital-to-analog converter (DAC). The DAC operates by using its 8 input bits to determine an appropriate analog voltage output, so this module continually outputs 8 bits (DAC_out[7:0]) that are always changing.

It uses many inputs to operate, nearly all of which are received from buttons on the labkit. These buttons correspond to changes in frequency and amplitude, as well as the selection of the wave type. The module assumes that these inputs have already been debounced using the debouncer module.

The operation of the Waveform FSM is highly dependent on a signal generated within the module called clock_vary. Clock_vary is generated by dividing down the 65 MHz system clock into a clock signal that is slower and more usable for proper signal generation. This dividing down is accomplished by using the standard technique in Verilog to divide signals down to a lower frequency by counting up to a certain limit before issuing a pulse, but the caveat with clock_vary is that this limit is always changing. Hence, even though a separate divider module is used later in the FSM to handle user inputs, clock_vary is generated by separate, explicit code. This allows the limit to be constantly changed by user input, which affects the frequency of clock_vary. Because the states in the FSM are changed at the positive edge of clock_vary, the final wave frequency is controlled by clock_vary.

The states of the waveform FSM are named for the wave types they control: square, ramp, sawtooth (triangle), and sine. The current wave type is controlled by the labkit pushbuttons; however, the system begins at a square wave. The square state operates by analyzing the current 8 bits to the DAC – if they are all 0, then it flips them to all 1s, otherwise it directs them to be all 0. Because of this very simple operation of flipping between eight 0s and eight 1s, the square wave is able to be generated at a much higher frequency than the rest of the waves. The ramp state operates by checking if the DAC's 8 bits are currently less than all 1s – if they are, then it increments them by 1, finally resetting them back to 0 once the limit is reached. This creates a very linear increase in the output coming out of the DAC, followed by a straight drop to 0. Because of the long time it takes to count from 0 to 255 (eight 0s to eight 1s) the ramp is not able to reach nearly the frequencies that the square wave can. The sawtooth, or triangle wave, is very similar to the ramp – it linearly counts up to the limit, but when the limit is reached, it simply counts back down. This counting up or counting down condition is controlled by a simple one-bit register that is TRUE if the count should be incrementing, FALSE otherwise, and set within the sawtooth state. Finally, the sine state operates by using one of Xilinx's intellectual property cores, and creates a sine module that uses 10 bits of input to a signal (here, "theta") and a look-up table to calculate the 8-bit signed value of sine(theta). The FSM shifts this value so it is no longer signed; this allows it to operate between 0 and 5 volts of amplitude. The 8 bit value of sine(theta) is sent to the DAC to generate the proper voltage. Theta is increased by 32 after each successful calculation, which means that although the wave is more jagged than normally expected of a sine wave, it is able to reach higher frequencies.

Though the FSM itself handles the creation of the wavetype and has a frequency determined by clock_vary, it is a simplified control scheme – the FSM is always generating a waveform that varies between the minimum and maximum voltage levels (between 0 and 5 volts). Therefore, another variable must be used to actually scale the output of the FSM before it is sent to the DAC. The amplitude scaling is controlled by an 8-bit register, gain. Gain ranges between 0 and 255, and the output from the FSM (as directed by one of its four states) is multiplied by gain and then divided by 256 in order to scale the final analog voltage. Gain begins at 128 (thus, creating a wave that varies between 0 and 2.5 volts) and is increased or decreased according to user input.

The last component of the waveform FSM is the handling of user input. The FSM begins at a predetermined frequency and gain (to set the amplitude) and in the square state. Using the divider module, a very slow clock at 20 Hz is generated. At the positive edge of this clock, the buttons on the labkit to increase/decrease frequency/amplitude and select the wave type are analyzed; if they are depressed, then the appropriate change is made. By holding the button down, the change is repeated, at 20 times per second. This allows the user to continuously change the frequency and amplitude without repeatedly pressing the button. As a final note, for safety precautions, the FSM will not output any bits to the DAC if its enable signal is false. This allows the labkit to remain on for easy, convenient rewiring.

2.1.2 Video:

The Video module controls the output to the monitor based on inputs from the Waveform FSM. The Video module is the top-level module for several other modules, which generate the appropriate pixels to display the text, real-time waveforms, and waveform images. Three registers store the amplitude, frequency, and duty cycle outputs from the Waveform FSM, which are changed on the positive edge of the 65 MHz clock if the frame is being updated. These values are fed to the Sine Sprite, Ramp Sprite, Triangle Sprite, and Square Sprite modules, which generate the real-time sine, ramp, triangle, and square waveforms. They are also fed to the Counter module, which converts the binary measurements to decimal numbers. The Char String Display module displays these measurements, in addition to the labels, as text on the monitor. The Image ROM module generates the buttons on the right side of the monitor.

The Video module determines the appropriate waveform to display using a simple case statement. The waveform selector input determines which real-time waveform to display. A 3-bit register stores the appropriate waveform, in the form of a pixel. For each waveform selection, this pixel is continuously assigned to the pixel for the corresponding real-time waveform, which is combined with a highlight pixel via an or gate. This highlight pixel is generated with the Blob module. It is a simple rectangle of the same size and position as the waveform being displayed, so that when displayed, it alters the colors of the image to set it apart from the other images. If the square is being displayed, the text for the duty cycle measurement is also added to the register via an or gate.

The Video module has three outputs, which are fed to the computer monitor. The 3-bit pixel output is the combination of all of the pixels necessary to display the text, real-time waveforms, and waveform images. Since the pixels from the Image ROM are delayed by one clock cycle, the pixel output is likewise delayed. One register for each of the horizontal sync, vertical sync, and blanking outputs is used to also delay them by one clock cycle. The combination of these outputs creates an analog output which results in a monitor image representing the waveform, an example of which is shown in Appendix A, with real-time waveforms and measurements which correspond to those of the Waveform FSM.

2.1.3 Image ROM:

The Image ROM module stores and displays the sine, square, ramp, and triangle waveform images. The waveform images were created in MATLAB, then converted to bitmap images. These bitmap images were run through an additional MATLAB script to create coe files. For each image, four coe files are created. One contains the actual image bits, while the other three are color maps for red, green, and blue. For each coe file, a ROM is created using Coregen, resulting in four ROM's per image. An image address, corresponding to the location of each pixel in the image, is fed into the ROM containing the actual image bits. The 8-bit output from this memory is the input into each of the three ROM's containing the red, green, and blue color maps. The 8-bit output from these three ROM's is delayed by one clock cycle, then continuously assigned to the 24-bit pixel output at the positive edge of the clock, if the pixel is in the correct location.

A simple case statement, corresponding to the image selector input from the Waveform FSM, determines which image to display. At all times, the memories are being accessed for each waveform, as described above; however, the pixel output is only assigned to the red, green, and blue color maps of the appropriate waveform. Initially, this enabled the user to simply display static images in place of the real-time waveforms.

The MATLAB code, images, coe files, and ROM Verilog modules used in the Image ROM module are all available in the Appendix.

2.1.4 Sine Sprite:

The Sine Sprite module is used to create the real-time sine waveform, which changes in amplitude and frequency corresponding to the inputs from the Waveform FSM. At the positive edge of the 65 MHz clock, the amplitude and frequency inputs are respectively assigned to 8- and 14-bit registers.

The horizontal and vertical measurements of the sine wave are determined by two wires. A 21-bit wire is continuously assigned to the first nine bits of hcount, multiplied by the 14-bit register storing the frequency, then divided by 256 to create the horizontal measurement. The vertical measurement is stored in a 16-bit wire. To create it, the Sine Real module was generated using the Coregen Trigonometry function. This module uses a lookup table to output the correct value for each value of theta. The theta input corresponds to the first eight bits of the horizontal measurement. The 8-bit output is multiplied by the 8-bit register storing the amplitude, then scaled by 255/256 to create the vertical measurement. This vertical measurement is then shifted to place it in the correct region.

The single output of this module is a 3-bit pixel. At the positive edge of the clock, if the pixel is in the region specified by the x, y, hcount, and vcount inputs, it is assigned to one of two colors, also specified as inputs. If the pixel is above the vertical shifted measurement, it is assigned to one color, while it is assigned to another if it is below. If the pixel is outside of this region, it is assigned to be black. The resulting sine wave corresponds to the amplitude and frequency inputs from the Waveform FSM module.

2.1.5 Ramp Sprite:

The Ramp Sprite module is used to create the real-time ramp waveform, which changes in amplitude and frequency corresponding to the inputs from the Waveform FSM. At the positive edge of the 65 MHz clock, the amplitude and frequency inputs are respectively assigned to 8- and 14-bit registers.

The horizontal and vertical measurements of the ramp wave are determined by two wires. A 21-bit wire is continuously assigned to the first nine bits of hcount, multiplied by the 14-bit register storing the frequency, then divided by 256 to create the horizontal measurement. The vertical measurement is stored in a 16-bit wire. To create it, a simple line is generated. The line has a base measurement of negative HEIGHT/4. The first seven bits of the horizontal measurement are added to the base value so that it is incremented by one with each increase in the horizontal measurement, until the horizontal measurement overflows and returns to zero. The

8-bit measurement is multiplied by the 8-bit register storing the amplitude, then scaled by 255/256 to create the vertical measurement. This vertical measurement is then shifted to place it in the correct region.

The single output of this module is a 3-bit pixel. At the positive edge of the clock, if the pixel is in the region specified by the x, y, hcount, and vcount inputs, it is assigned to one of two colors, also specified as inputs. If the pixel is above the vertical shifted measurement, it is assigned to one color, while it is assigned to another if it is below. If the pixel is outside of this region, it is assigned to be black. The resulting ramp wave corresponds to the amplitude and frequency inputs from the Waveform FSM module.

2.1.6 Triangle Sprite:

The Triangle Sprite module is used to create the real-time triangle waveform, which changes in amplitude and frequency corresponding to the inputs from the Waveform FSM. At the positive edge of the 65 MHz clock, the amplitude and frequency inputs are respectively assigned to 8- and 14-bit registers.

The horizontal and vertical measurements of the triangle wave are determined by two wires. A 21-bit wire is continuously assigned to the first nine bits of hcount, multiplied by the 14-bit register storing the frequency, then divided by 256 to create the horizontal measurement. The vertical measurement is stored in a 16-bit wire. To create it, two simple lines are generated. One line has a base measurement of negative 3*HEIGHT/4. The first eight bits of the horizontal measurement are added to the base value so that it is incremented by one with each increase in the horizontal measurement. The other line has a base measurement of HEIGHT/4. The first eight bits of the horizontal measurement are subtracted from the base value so that it is decremented by one with each increase in the horizontal measurement. Both lines are also shifted upwards by one pixel. If the first eight bits of the horizontal measurement are greater than WIDTH/4, then the first line is used for the vertical measurement; otherwise, the second line is used. The 8-bit measurement is multiplied by the 8-bit register storing the amplitude, then scaled by 255/256 to create the vertical measurement. This vertical measurement is then shifted to place it in the correct region.

The single output of this module is a 3-bit pixel. At the positive edge of the clock, if the pixel is in the region specified by the x, y, hcount, and vcount inputs, it is assigned to one of two colors, also specified as inputs. If the pixel is above the vertical shifted measurement, it is assigned to one color, while it is assigned to another if it is below. If the pixel is outside of this region, it is assigned to be black. The resulting triangle wave corresponds to the amplitude and frequency inputs from the Waveform FSM module.

2.1.7 Square Sprite:

The Square Sprite module is used to create the real-time square waveform, which changes in amplitude, frequency, and duty cycle corresponding to the inputs from the Waveform FSM. At the positive edge of the 65 MHz clock, the amplitude, frequency, and duty cycle inputs are respectively assigned to 8-, 14-, and 3-bit registers.

The horizontal and vertical measurements of the square wave are determined by two wires. A 21-bit wire is continuously assigned to the first nine bits of hcount, multiplied by the 14-bit register storing the frequency, then divided by 256 to create the horizontal measurement. The vertical measurement is stored in a 16-bit wire. To create it, two simple lines are generated. One line is equal to negative HEIGHT/4 + 1, while the other equaled HEIGHT/4 - 1. If the first seven bits of the horizontal measurement are greater than or equal to the 4-bit register storing the duty cycle, then the first line is used for the vertical measurement; otherwise, the second line is used. The 8-bit measurement is multiplied by the 8-bit register storing the amplitude, then scaled by 255/256 to create the vertical measurement. This vertical measurement is then shifted to place it in the correct region.

The single output of this module is a 3-bit pixel. At the positive edge of the clock, if the pixel is in the region specified by the x, y, hcount, and vcount inputs, it is assigned to one of two colors, also specified as inputs. If the pixel is above the vertical shifted measurement, it is assigned to one color, while it is assigned to another if it is below. If the pixel is outside of this region, it is assigned to be black. The resulting triangle wave corresponds to the amplitude and frequency inputs from the Waveform FSM module.

2.1.8 Counter:

The Counter module is used to convert binary numbers to decimal digits. It can convert any binary number with up to sixteen bits to the corresponding decimal number with up to five digits. Several registers are used to accomplish this. Three 16-bit registers store the binary count input, old binary count input, and current count. At the positive edge of the clock, the only other input to the module besides the binary number, one register is assigned to the binary count input, while the other is assigned to the previous value of the binary count input. If these values are not the same, the count register is assigned to the current value of the binary input, and the count begins. A 1-bit register, which is high if the count is incomplete, is assigned to a high value, which remains high until the count is complete.

Five 4-bit registers are used for each of the five decimal digits. When the count begins, these registers are all assigned to zero. With each positive edge of the clock the binary count is decremented by one, while the decimal count is incremented by one, until the binary count is equal to zero and the decimal count is equal to its final value. To handle decimal overflow, a series of if else statements are used to decide the appropriate values for each digit. If the first four digits, which are least significant, are equal to nine, they are all assigned to zero and the fifth digit is assigned to one. A similar process occurs if the first three, first two, or first digits are equal to nine. If none of these cases are true, the first digit is incremented by one. This process continues until the binary count is equal to zero, at which point the 1-bit register is assigned a low value and the digits are no longer incremented. The 20-bit decimal output is continuously assigned to these five digits.

2.1.9 Meas String:

The Meas String module is used to convert a decimal number into strings for display on the monitor. It can convert any decimal number with up to five digits to the appropriate strings. Five case statements, corresponding to each 4-bit digit in the decimal number, are used to accomplish this. For each digit, the string output corresponds to the appropriate decimal number. The five 11-bit registers corresponding to each digit are continuously assigned to the appropriate string, then output by the module.

2.1.10 Char String Display:

The Char String Display module displays 8 x 12 pixel characters by translating inputs of strings of characters into 3-bit pixel output. A Font ROM module, which was generated using Coregen and a coe file, contains all of the characters. For each character in the input string, the module creates the appropriate address, which is input to the Font ROM. The 8-bit output is used to determine which pixels are colored, if they are in the appropriate region.

2.1.11 XVGA:

The XVGA module uses the 65 MHz system clock to determine the appropriate hcount, vcount, hsync, vsync, and blank signals to send to the monitor. Because the clock is 65 MHz, the horizontal display is 1024 pixels wide, and the vertical display is 768 pixels high.

2.1.12 Blob:

The Blob module displays a rectangle on the screen, with the width, height, and color specified by the parameters. It contains two 3-bit registers, which are used to determine the 3-bit pixel output. If the pixel is in the correct region, it is assigned to the color input; otherwise, it is assigned to be black. The pixel is delayed by one clock cycle, then continuously assigned to be the sole output of the module.

2.1.13 Debouncer:

The debouncer module is necessary for this design due to the use of pushbutton switches. The need for a debouncer arises due to the "ringing" associated with a mechanical button. When the button is depressed to create contact between its two metal leads, the button will assuredly continue to vibrate back and forth, potentially creating unwanted noise and generating undesired signals. This debouncer module solves this issue by taking as input the initial button press, along with any transient signals produced by the press. By latching this initial input and requiring it to be stable for 0.01 seconds before passing it along as output, the module essentially provides a perfectly clean 'on' or 'off' signal to the rest of the system. The debouncer module has the added benefit that it synchronizes the button presses with the necessary clock signals as well.

2.1.14 Divider

The divider module is designed to create a 20 Hz clocking signal out of the system's own 65 MHz clock. This slower clock is used when interpreting button presses in the waveform FSM.

2.1.15 Final Project:

These modules are all implemented on the labkit FPGA. The code to load data to the labkit is available elsewhere, but the alterations made to the Final Project module to instantiate the modules described above can be found in the Appendix.

2.2 Testing and Debugging

There were several levels of testing and debugging in the implementation of this controllable function generator. All modules were individually built, and when feasible, individually tested using various inputs and outputs from the labkit FPGA, monitor, and oscilloscope. After individual testing, the modules were connected into two main groups, each controlling the digital and analog output. Once these groups were each functioning, the entire project was wired to its final state.

The monitor was the primary source for debugging the digital output. The Video, XVGA, and Blob modules were all tested first, to ensure that the clocking and display were correct. When rectangles could be displayed in the place of each image, the Image ROM module was tested. Each waveform was successfully displayed in the appropriate location, although the colors were never correct due to an artifact from the MATLAB script. Once these images were correctly displayed, the real-time waveforms were created. To debug these modules, the up, down, left, and right labkit buttons were used to control amplitude and frequency, while switches were used to control the duty cycle. Once the waveforms could all be appropriately displayed using these inputs, the functionality for amplitude, frequency, and duty cycle inputs was added. During testing, switches were used to represent all possible values of amplitude, frequency, and duty cycle.

One aspect of the video module which did not use the monitor for initial debugging was the Counter module. For this module, the 16-digit hex display on the labkit was used to display the decimal digits, while switches were used as the binary inputs. Once this module correctly displayed the decimal digits corresponding to the binary input, the monitor was used to test the Char String Display, Font ROM, and Meas String modules. The correct display of the measurements as pixels on the monitor indicated that all four modules were functional.

The oscilloscope was used to test the output from the Waveform FSM. Initially, the challenge was to display each of the four types of waveforms. After each waveform was displayed cleanly, the functionality to change amplitude and frequency according to the labkit buttons was added. The challenge with waveform generation was mostly due to the waveform's dependence on clock_vary, which is always changing frequency as buttons are pressed. This made it much more difficult than was predicted to properly measure the frequency for display on-screen; though the amplitude was successfully decoded from binary to decimal and displayed on the monitor, frequency was not. The duty cycle displayed on screen was also correct in that it matched the labkit switches that set its value. However, the duty cycle was not implemented successfully with the waveform generator, and so the square waves were stuck at a 50% duty cycle when coming out of the DAC.

Once all of the tests were passed, the modules were wired to their final states. Final testing involved the manipulation of inputs in all possible scenarios. The waveforms displayed on the oscilloscope matched those on the monitor, verifying the functionality of the system.

3. CONCLUSION

This report describes the functionality and implementation of a controllable function generator. Inputs from the labkit allow the user to select and control periodic square, sine, ramp, and triangle waveforms. The 8-bit output is very similar to that found on a standard lab function generator, but this implementation improves usability with the addition of a video display, which contains a digital representation of the waveform, complete with associated measurements for amplitude, frequency, and duty cycle. The video display also has buttons on the side, which light up to indicate which waveform is being displayed, especially useful for cases in which changes in amplitude, frequency, or duty cycle alter the waveform such that it is unrecognizable.

Though the new system is a great improvement over current function generators, further modifications are possible. The frequency measurement is incorrect and should be altered, as described previously. This change, in addition to the ability to change the duty cycle of the analog square wave, would make the displayed measurements completely and accurately match the analog output. A further enhancement would be the addition of mouse control. The four images on the right side of the monitor could function as buttons, which would change the waveform type when clicked. The mouse could also be used to change the amplitude and frequency of the waveform, by pulling the waveform up and down or left and right to indicate changes. These changes would need to be output to the Waveform FSM so that the analog waveform output by the DAC would likewise change. The final possible improvement is to the images displayed by the Image ROM. Diagnosing the problem, most likely with the MATLAB script, would allow for the display of images with the correct color.

One issue that is more of an inconvenience than actual failure is the speed at which frequencies change when the labkit buttons are pressed. Due to the design, it results in extremely rapid changes in frequency if the generator is already producing a higher frequency. Likewise, the generator is slow to change frequencies if it is already producing a slower signal. One solution to get around this is the addition of a quick-jump feature, which would, after a single button press, jump the signal to a desired frequency; from there it could be increased or decreased as desired. This is a minor addition and would only serve to make the use of the function generator a little more user-friendly and convenient.

The design is functional even without these improvements, as rigorous testing demonstrated. When feasible, modules were tested individually, using the labkit FPGA, oscilloscope, and computer monitor. For modules that required a great deal of communication with other modules, specifically the modules controlling the video and analog output, connections were made to the necessary modules, which had already been individually tested. These two clusters of modules were then tested, again using inputs and outputs from the labkit FPGA, oscilloscope, and computer monitor. To finally test the entire system, all modules were wired to their final states, and then all possible scenarios were input. The major problems encountered during testing occurred because the Waveform FSM used different clocks to generate different waveforms, so the frequency changed slower than that of the monitor.

This extensive testing resulted in a controllable function generator with its own display, which provides enhanced usability over standard factory units.

REFERENCES

[1] "AD7224 | LC2MOS 8-Bit DAC with Output Amplifiers | All D/A Converters | Digital to Analog Converters | Analog Devices." *Analog Devices | Mixed-signal and Digital Signal Processing ICs*. Analog Devices, Dec. 1984. Web. 10 Dec. 2010. <http://www.analog.com/en/digital-to-analog-converters/da-converters/ad7224/products/product.html>.

[2] Various modules, including hex display, xvga, blob, font rom, char string display, debouncer, labkit. MIT. "6.111." *6.111 Fall 2005*. Sept. 2005. Web. 10 Dec. 2010. <http://web.mit.edu/6.111/www/f2005/index.html>.

APPENDICES

Appendix A: Screenshots of waveforms and monitor

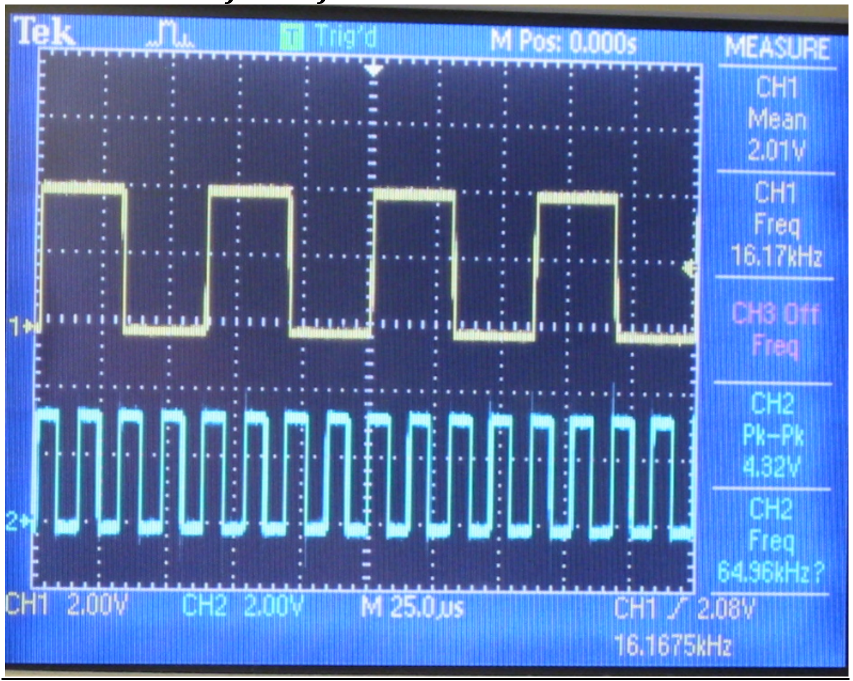


Figure 2: Square wave as viewed on oscilloscope. This screenshot is taken from the in-lab oscilloscope and demonstrates a successful square (in yellow). Clock_vary is shown in blue, below it. Note that the square still maintains clean transitions, even at its maximum frequency of 16 kHz.

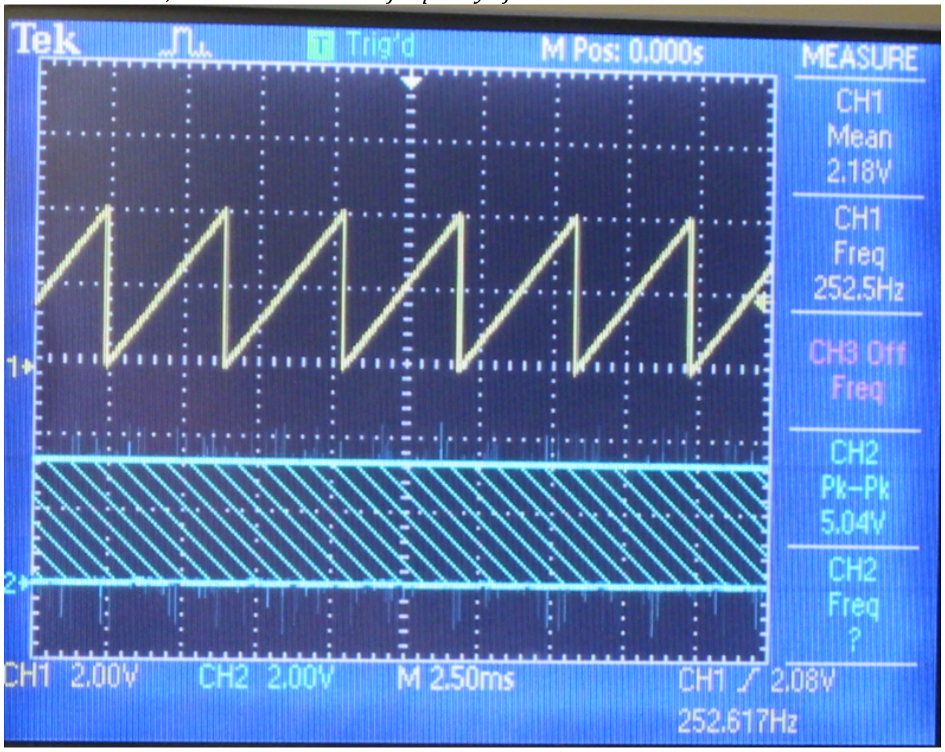


Figure 3: Ramp wave as viewed on oscilloscope. This screenshot is taken from the in-lab oscilloscope and demonstrates a successful ramp (in yellow). Clock_vary is shown in blue, below. The ramp is shown here at its maximum frequency of just above 250 Hz.

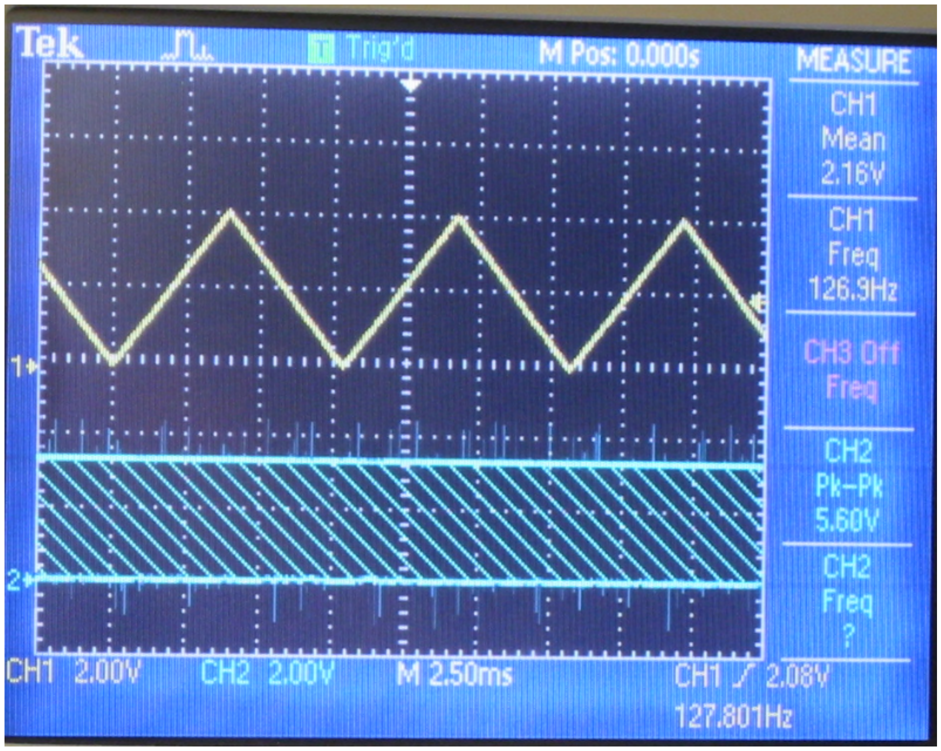


Figure 4: Triangle/sawtooth wave as viewed on oscilloscope. This screenshot is taken from the in-lab oscilloscope and demonstrates a successful triangle (in yellow). Clock_vary is shown in blue, below. The wave is shown here at its maximum frequency of just above 125 Hz.

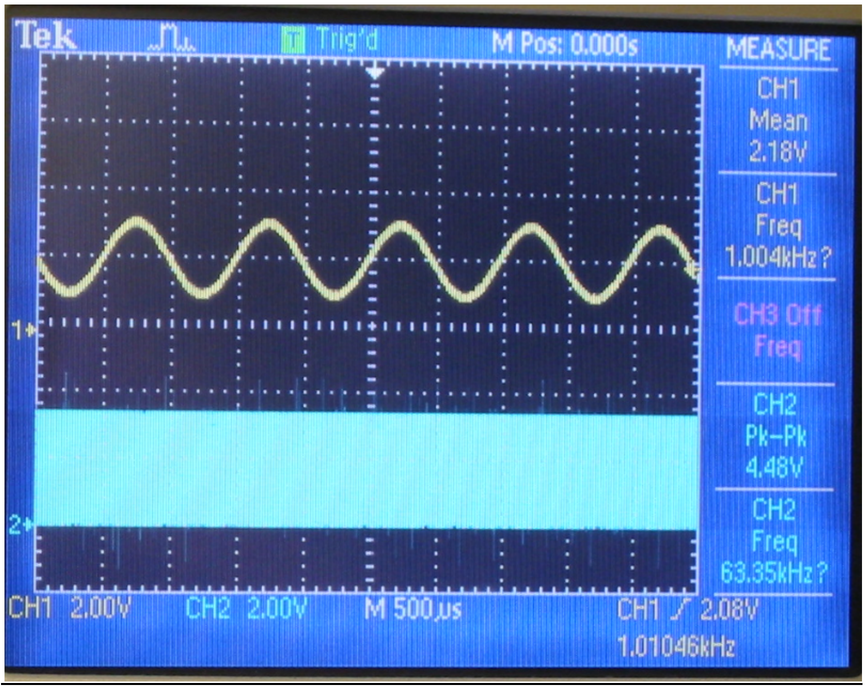
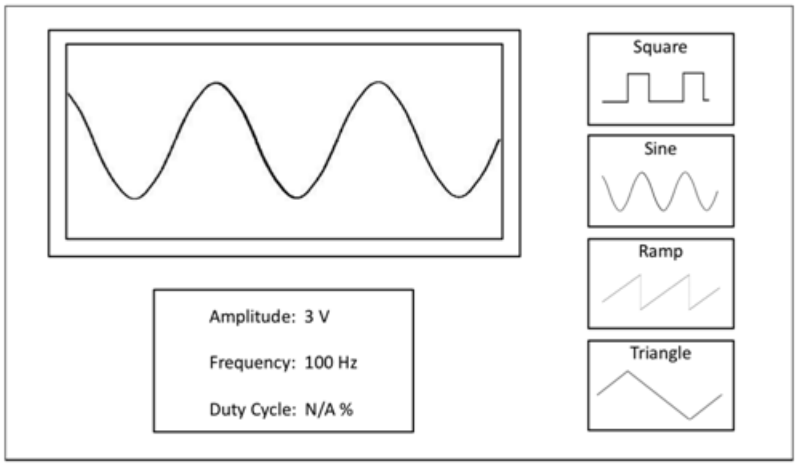


Figure 5: Sine wave as viewed on oscilloscope. This screenshot is taken from the in-lab oscilloscope and demonstrates a successful sine wave (in yellow). Clock_vary is shown in blue, below. The ramp is shown here at its maximum frequency of just above 1 kHz.



Amplitude: 3 V

Frequency: 100 Hz

Duty Cycle: N/A %

Figure 6: Example display shown on the monitor while function generator is active.

Appendix B: Matlab Code

Note: Verilog code is too lengthy to include; please check the code uploaded to website.

```
% Sarah Ferguson
% 6.111 Final Project: A Controllable Function Generator
% Generation of MATLAB images

% define range of plot
x = -2*pi:0.01:2*pi;

%% plot sine wave
figure;
plot(x,sin(x),'m','LineWidth',2);
axis equal;
axis off;

%% plot square wave, 50% duty cycle
figure;
plot(x,square(x,50),'b','LineWidth',2);
axis equal;
axis off;

%% plot square wave, 25% dudy cycle
figure;
plot(x,square(x,25),'b','LineWidth',2);
axis equal;
axis off;

%% plot triangle wave
triangle = zeros(1,length(x));
j = 0;
for i = -2*pi:0.01:2*pi;
    j = j+1;
    if (i>=-2*pi && i<-pi)
        triangle(j) = i;
    elseif (i>=-pi && i<0)
        triangle(j) = -i-2*pi;
    elseif (i>=0 && i<pi)
        triangle(j) = i-2*pi;
    else
        triangle(j) = -i;
    end
end

figure;
plot(x,triangle,'g','LineWidth',2);
axis equal;
axis off;

%% plot ramp wave
ramp = zeros(1,length(x));
j = 0;
for i = -2*pi:0.01:2*pi;
    j = j+1;
    if (i>=-2*pi && i<-pi)
        ramp(j) = i;
    elseif (i>=-pi && i<0)
```

```
        ramp(j) = i-2*pi;
    elseif (i>=0 && i<pi)
        ramp(j) = i-2*pi;
    else
        ramp(j) = i-4*pi;
    end
end

figure;
plot(x,ramp,'y','LineWidth',2);
axis equal;
axis off;
```

```
%6.111 Image Color Table MATLAB deme
%Edgar Twigg bwayr@mit.edu
%4/1/2008 (But I swear this file isn't a joke)

%% How to use this file
%Notice how %% divides up sections?  If you hit ctrl+enter, then MATLAB
%will execute all the lines within that section, but nothing else.  You
can
%also navigate quickly through the file using ctrl+arrow_key

%% Getting 24 bit data
%So when you look at a 24 bit bitmap file, the file specifies three 8
bit
%values for each color, 8 each for red, green, and blue.
[picture] = imread('square256x192.bmp');

%% View the image
%This command image will draw the picture you just loaded
figure                  %opens a new window
image(picture)          %draws your picture
title('24 bit bitmap')  %gives it a title so you don't forget what it
is

%% Manipulate the data in the image
%So now you have a matrix of values that represent the image.  You can
%access them in the following way:
%
%picture(row,column,color)
%
%Remember that MATLAB uses 1-based indexes, and Verilog uses 0!
%
%Also, you can use MATLAB's slice operator to do nifty things.
%picture(:,:,1) would return a 2D matrix with the red value for every
row and
%column.
%
%picture(:,1,2) would return a 1D matrix with the green value for every
row
%in the first column.

%This is how MATLAB indexes the colors
RED = 1;
GREEN = 2;
BLUE = 3;

%So if we wanted to see the red values of the image only, we could say
figure
image(picture(:,:,RED))
title('Red values in 24 bit bitmap')

%Because the image we gave matlab above specifies only one value per
pixel
%rather than usual three (red,blue,green), MATLAB colors each pixel
from
%blue to red based on the value at that pixel.

%% Getting 8 bit data
%When you store an 8 bit bitmap, things get a little more complicated.
```

```
Now
%each pixel in the image only gets one 8 bit value.  But, you need to
send
%the monitor an r,g, and b!  How can this work?
%
%8 bit bitmaps include a table which specifies the rgb values for each
of
%the 8 bits in the image.
%
%So each pixel is represented by one byte, and that byte is an index
into a
%table where each index specifies an r, g, and b value separately.
%
%Because of this, now we need to load both the image and it's colormap.
[picture color_table] = imread('square256x192.bmp');

%% Displaying without the color table
%If we try to display the picture without the colormap, the image does
not
%make sense
figure
image(picture)
title('Per pixel values in 8 bit bitmap')

%% Displaying WITH the color table
%So to display the picture with the proper color table, we need to tell
%MATLAB to set its colormap to be in line with our colorbar.  The image
%quality is somewhat reduced compared to the 24 bit image, but not too
bad.
figure
image(picture)
colormap(color_table)   %This command tells MATLAB to use the image's
color table
colorbar                %This command tells MATLAB to draw the color
table it is using
title('8 bit bitmap displayed using color table')

%% More about the color table
%The color table is in the format:
%
%color_table(color_index,1=r 2=g 3=b)
%
%So to get the r g b values for color index 3, we only need to say:
disp('         r    g              b    for color 3 is:')
disp(color_table(3,:))      %disp = print to console

%Although in the bitmap file the colors are indexed as 0-255 and each
rgb
%value is an integer between 0-255, MATLAB images don't work like that,
so
%MATLAB has automatically scaled them to be indexed 1-256 and to have a
%floating point value between 0 and 1.  To turn the floats into integer
%values between 0 and 256:

color_table_8bit = uint8(round(256*color_table));

disp('    r    g    b    for color 3 in integers is:')
disp(color_table_8bit(3,:))
```

```
%Note that this doesn't fix the indexing (and it can't, since MATLAB
won't
%let you have indexes below 1)

%another way to look at the color table is like this (don't worry about
how
%to make this graph)
figure
stem3(color_table_8bit)
set(gca,'XTick',1:3);
set(gca,'YTick',[1,65,129,193,256]);
set(gca,'YTickLabel',['  0';' 64';'128';'192';'255']);
set(gca,'ZTick',[0,64,128,192,255]);

xlabel('red = 1, green = 2, blue = 3')
ylabel('color index')
zlabel('value')
title('Another way to see the color table')

%% Even smaller bitmaps
%You can extend what we did for 8-bit bitmaps to even more compressed
%forms, such as this 4-bit bitmap.  Now we only have 16 colors to work
with
%though, and our image quality is significantly reduced:
[picture color_table] = imread('square256x192.bmp');

figure
image(picture)
colormap(color_table)
colorbar
title('4 bit bitmap displayed using color table')

%% Writing data to coe files for putting them on the fpga
%You can instantiate BRAMs to take their values from a file you feed
them
%when you flash the FPGA.  You can use this technique to send them
%colortables, image data, anything.  Here's how to send the red
component
%of the color table of the last example

red = color_table(:,3);              %grabs the red part of the
colortable
scaled_data = red*255;               %scales the floats back to 0-255
rounded_data = round(pixel_columns);  %rounds them down
data = dec2bin(rounded_data,8);      %convert the binary data to 8 bit
binary #s

%open a file
output_name = 'square.coe';
file = fopen(output_name,'w');

%write the header info
fprintf(file,'memory_initialization_radix=2;\n');
fprintf(file,'memory_initialization_vector=\n');
fclose(file);

%put commas in the data
```

```
rowxcolumn = size(data);
rows = rowxcolumn(1);
columns = rowxcolumn(2);
output = data;
for i = 1:(rows-1)
    output(i,(columns+1)) = ',';
end
output(rows,(columns+1)) = ';';

%append the numeric values to the file
dlmwrite(output_name,output,'-append','delimiter','', 'newline', 'pc');

%You're done!

%% Turning a 2D image into a 1D memory array
%The code above is all well and good for the color table, since it's 1-
D
%(well, at least you can break it into 3 1-D arrays).  But what about a
2D
%array?  We need to turn it into a 1-D array:

picture_size = size(picture);       %figure out how big the image is
num_rows = picture_size(1);
num_columns = picture_size(2);

pixel_columns = zeros(picture_size(1)*picture_size(2),1,'uint8');
%pre-allocate a space for a new column vector

for r = 1:num_rows
    for c = 1:num_columns
        pixel_columns((r-1)*num_columns+c) = picture(r,c);    %pixel# =
(y*numColumns)+x
    end
end

%so now pixel_columns is a column vector of the pixel values in the
image

%just to make sure that we're doing things correctly
regen_picture = zeros(num_rows,num_columns,'uint8');
for r = 1:num_rows
    for c = 1:num_columns
        regen_picture(r,c) = pixel_columns((r-1)*num_columns+c,1);
    end
end

figure
subplot(121)
image(picture)
axis square
colormap(color_table)
colorbar
title('Original Picture')

subplot(122)
image(regen_picture)
axis square
colormap(color_table)
```

```
colorbar
title('Regenerated Picture')
```

Appendix C: Waveform Images used by Image ROM

The following four images were generated with the first MATLAB script from Appendix A, then used as the inputs for the second MATLAB script from the same Appendix to create coe files. These coe files were used by the Image ROM module to display the images on the monitor.

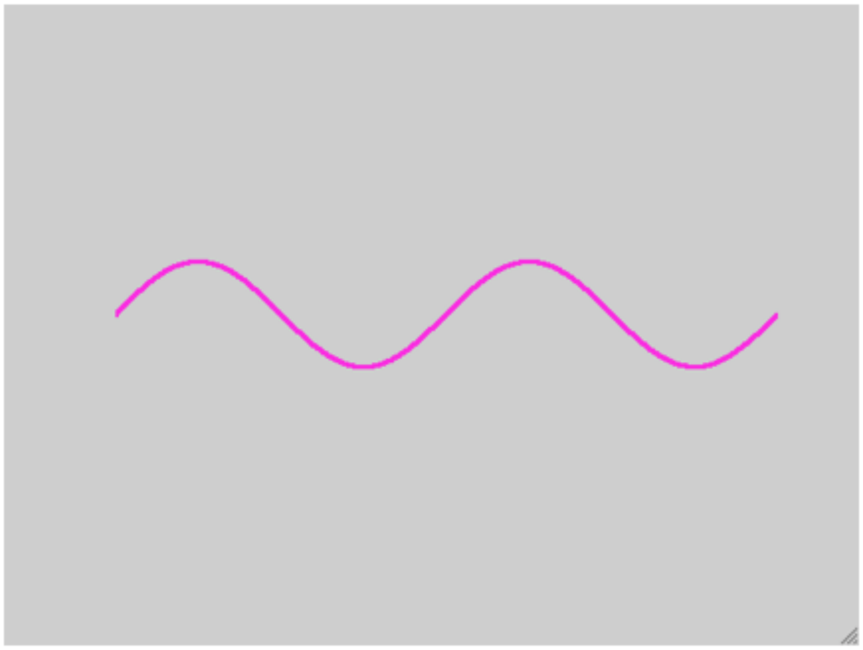


Figure 7: Bitmap image of sine wave, used on display

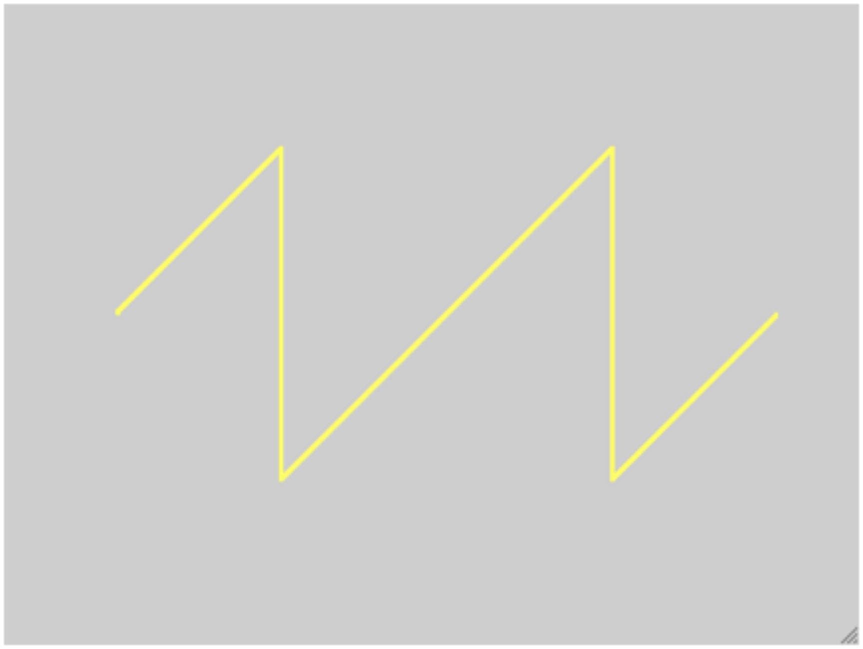


Figure 8: Bitmap image of ramp wave, used on display

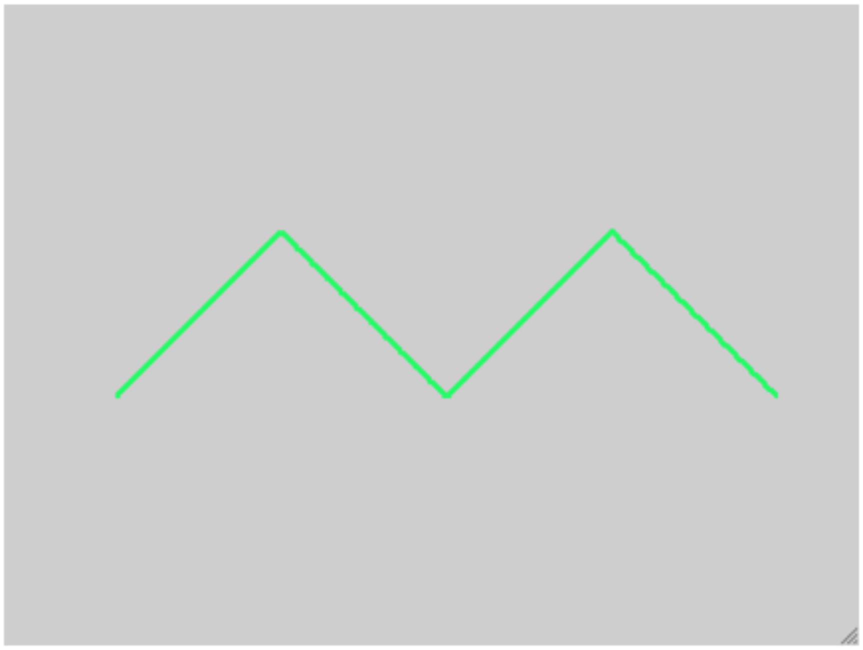


Figure 9: Bitmap image of triangle wave, used on display

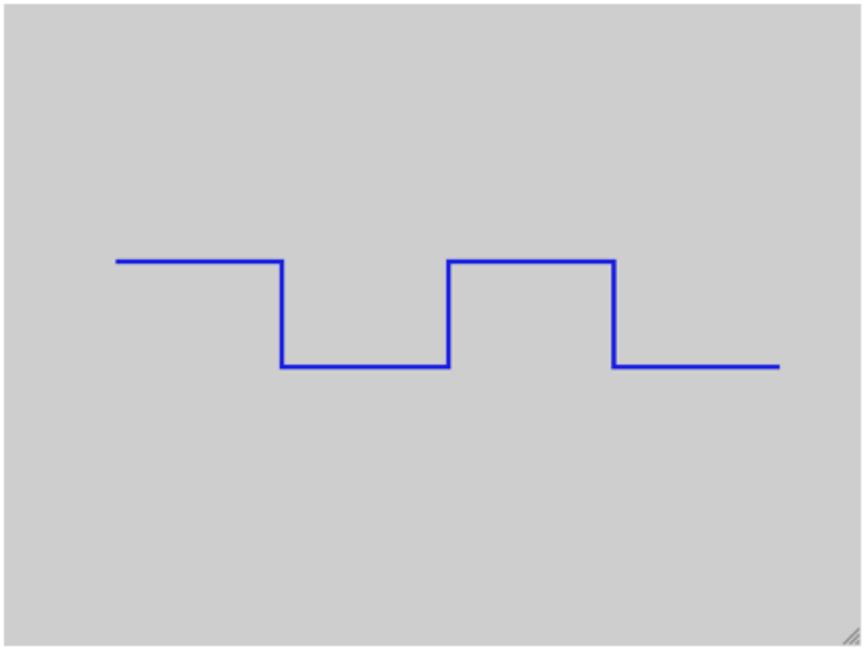


Figure 10: Bitmap image of square wave, used on display

Appendix D: Analog Devices AD7224 Data Sheet

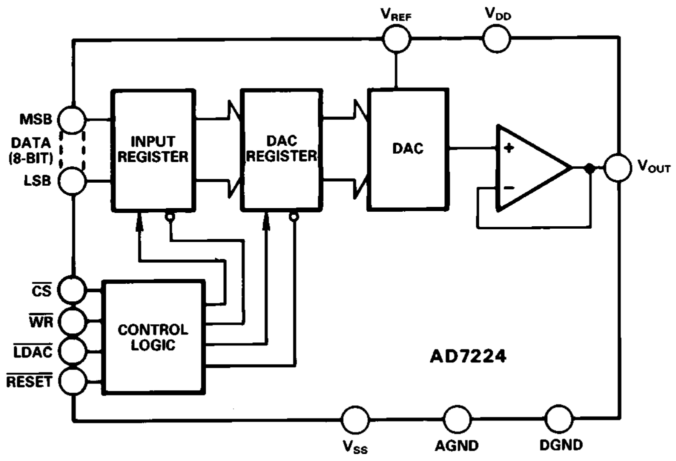
ANALOG DEVICES

LC^{2}MOS 8-Bit DAC with Output Amplifiers

AD7224

FEATURES
8-Bit CMOS DAC with Output Amplifiers
Operates with Single or Dual Supplies
Low Total Unadjusted Error:
Less Than 1 LSB Over Temperature
Extended Temperature Range Operation
µP-Compatible with Double Buffered Inputs
Standard 18-Pin DIPs, and 20-Terminal Surface Mount Package and SOIC Package

FUNCTIONAL BLOCK DIAGRAM



GENERAL DESCRIPTION

The AD7224 is a precision 8-bit voltage-output, digital-to-analog converter, with output amplifier and double buffered interface logic on a monolithic CMOS chip. No external trims are required to achieve full specified performance for the part.

The double buffered interface logic consists of two 8-bit registers–an input register and a DAC register. Only the data held in the DAC registers determines the analog output of the converter. The double buffering allows simultaneous update in a system containing multiple AD7224s. Both registers may be made transparent under control of three external lines, $\overline{CS}$, $\overline{WR}$ and $\overline{LDAC}$. With both registers transparent, the $\overline{RESET}$ line functions like a zero override; a useful function for system calibration cycles. All logic inputs are TTL and CMOS (5 V) level compatible and the control logic is speed compatible with most 8-bit microprocessors.

Specified performance is guaranteed for input reference voltages from +2 V to +12.5 V when using dual supplies. The part is also specified for single supply operation using a reference of +10 V. The output amplifier is capable of developing +10 V across a 2 kΩ load.

The AD7224 is fabricated in an all ion-implanted high speed Linear Compatible CMOS (LC^{2}MOS) process which has been specifically developed to allow high speed digital logic circuits and precision analog circuits to be integrated on the same chip.

PRODUCT HIGHLIGHTS

1. DAC and Amplifier on CMOS Chip
   The single-chip design of the 8-bit DAC and output amplifier is inherently more reliable than multi-chip designs. CMOS fabrication means low power consumption (35 mW typical with single supply).
2. Low Total Unadjusted Error
   The fabrication of the AD7224 on Analog Devices Linear Compatible CMOS (LC^{2}MOS) process coupled with a novel DAC switch-pair arrangement, enables an excellent total unadjusted error of less than 1 LSB over the full operating temperature range.
3. Single or Dual Supply Operation
   The voltage-mode configuration of the AD7224 allows operation from a single power supply rail. The part can also be operated with dual supplies giving enhanced performance for some parameters.
4. Versatile Interface Logic
   The high speed logic allows direct interfacing to most microprocessors. Additionally, the double buffered interface enables simultaneous update of the AD7224 in multiple DAC systems. The part also features a zero override function.

REV. B

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 617/329-4700 **Fax: 617/326-8703**

AD7224–SPECIFICATIONS

DUAL SUPPLY

(V_{DD} = 11.4 V to 16.5 V, V_{SS} = –5 V ± 10%; AGND = DGND = O V; V_{REF} = +2 V to (V_{DD} – 4 V)[1] unless otherwise noted. All specifications T_{MIN} to T_{MAX} unless otherwise noted.)

Parameter	K, B, T Versions[2]	L, C, U Versions[2]	Units	Conditions/Comments
STATIC PERFORMANCE				
Resolution	8	8	Bits	
Total Unadjusted Error	±2	±1	LSB max	V_{DD} = +15 V ± 5%, V_{REF} = +10 V
Relative Accuracy	±1	±1/2	LSB max	
Differential Nonlinearity	±1	±1	LSB max	Guaranteed Monotonic
Full-Scale Error	±3/2	±1	LSB max	
Full-Scale Temperature Coefficient	±20	±20	ppm/°C max	V_{DD} = 14 V to 16.5 V, V_{REF} = +10 V
Zero Code Error	±30	±20	mV max	
Zero Code Error Temperature Coefficient	±50	±30	µV/°C typ	
REFERENCE INPUT				
Voltage Range	2 to (V_{DD} – 4)	2 to (V_{DD} – 4)	V min to V max	
Input Resistance	8	8	kΩ min	
Input Capacitance[3]	100	100	pF max	Occurs when DAC is loaded with all 1s.
DIGITAL INPUTS				
Input High Voltage, V_{INH}	2.4	2.4	V min	
Input Low Voltage, V_{INL}	0.8	0.8	V max	
Input Leakage Current	±1	±1	µA max	V_{IN} = 0 V or V_{DD}
Input Capacitance[3]	8	8	pF max	
Input Coding	Binary	Binary		
DYNAMIC PERFORMANCE				
Voltage Output Slew Rate[3]	2.5	2.5	V/µs min	
Voltage Output Settling Time[3]				
Positive Full-Scale Change	5	5	µs max	V_{REF} = +10 V; Settling Time to ±1/2 LSB
Negative Full-Scale Change	7	7	µs max	V_{REF} = +10 V; Settling Time to ±1/2 LSB
Digital Feedthrough	50	50	nV secs typ	V_{REF} = 0 V
Minimum Load Resistance	2	2	kΩ min	V_{OUT} = +10 V
POWER SUPPLIES				
V_{DD} Range	11.4/16.5	11.4/16.5	V min/V max	For Specified Performance
V_{SS} Range	4.5/5.5	4.5/5.5	V min/V max	For Specified Performance
I_{DD}				
@ 25°C	4	4	mA max	Outputs Unloaded; V_{IN} = V_{INL} or V_{INH}
T_{MIN} to T_{MAX}	6	6	mA max	Outputs Unloaded; V_{IN} = V_{INL} or V_{INH}
I_{SS}				
@ 25°C	3	3	mA max	Outputs Unloaded; V_{IN} = V_{INL} or V_{INH}
T_{MIN} to T_{MAX}	5	5	mA max	Outputs Unloaded; V_{IN} = V_{INL} or V_{INH}
SWITCHING CHARACTERISTICS[3, 4]				
t_1				
@ 25°C	90	90	ns min	Chip Select/Load DAC Pulse Width
T_{MIN} to T_{MAX}	90	90	ns min	
t_2				
@ 25°C	90	90	ns min	Write/Reset Pulse Width
T_{MIN} to T_{MAX}	90	90	ns min	
t_3				
@ 25°C	0	0	ns min	Chip Select/Load DAC to Write Setup Time
T_{MIN} to T_{MAX}	0	0	ns min	
t_4				
@ 25°C	0	0	ns min	Chip Select/Load DAC to Write Hold Time
T_{MIN} to T_{MAX}	0	0	ns min	
t_5				
@ 25°C	90	90	ns min	Data Valid to Write Setup Time
T_{MIN} to T_{MAX}	90	90	ns min	
t_6				
@ 25°C	10	10	ns min	Data Valid to Write Hold Time
T_{MIN} to T_{MAX}	10	10	ns min	

NOTES

[1]Maximum possible reference voltage.

[2]Temperature ranges are as follows:
K, L Versions: –40°C to +85°C
B, C Versions: –40°C to +85°C
T, U Versions: –55°C to +125°C

[3]Sample Tested at 25°C by Product Assurance to ensure compliance.

[4]Switching characteristics apply for single and dual supply operation.

Specifications subject to change without notice.

SINGLE SUPPLY (V_{DD} = +15 V ± 5%; V_{SS} = AGND = DGND = 0 V; V_{REF} = +10 V[1] unless otherwise noted. All specifications T_{MIN} to T_{MAX} unless otherwise noted.)

Parameter	K, B, T Versions[2]	L, C, U Versions[2]	Units	Conditions/Comments
STATIC PERFORMANCE				
Resolution	8	8	Bits	
Total Unadjusted Error	±2	±2	LSB max	
Differential Nonlinearity	±1	±1	LSB max	Guaranteed Monotonic
REFERENCE INPUT				
Input Resistance	8	8	kΩ min	
Input Capacitance[3]	100	100	pF max	Occurs when DAC is loaded with all 1s.
DIGITAL INPUTS				
Input High Voltage, V_{INH}	2.4	2.4	V min	
Input Low Voltage, V_{INL}	0.8	0.8	V max	
Input Leakage Current	±1	±1	μA max	V_{IN} = 0 V or V_{DD}
Input Capacitance[3]	8	8	pF max	
Input Coding	Binary	Binary		
DYNAMIC PERFORMANCE				
Voltage Output Slew Rate[4]	2	2	V/μs min	
Voltage Output Settling Time[4]				
Positive Full-Scale Change	5	5	μs max	Settling Time to ±1/2 LSB
Negative Full-Scale Change	20	20	μs max	Settling Time to ±1/2 LSB
Digital Feedthrough[3]	50	50	nV secs typ	V_{REF} = 0 V
Minimum Load Resistance	2	2	kΩ min	V_{OUT} = +10 V
POWER SUPPLIES				
V_{DD} Range	14.25/15.75	14.25/15.75	V min/V max	For Specified Performance
I_{DD}				
@ 25°C	4	4	mA max	Outputs Unloaded; V_{IN} = V_{INL} or V_{INH}
T_{MIN} to T_{MAX}	6	6	mA max	Outputs Unloaded; V_{IN} = V_{INL} or V_{INH}
SWITCHING CHARACTERISTICS[3, 4]				
t_1				
@ 25°C	90	90	ns min	Chip Select/Load DAC Pulse Width
T_{MIN} to T_{MAX}	90	90	ns min	
t_2				
@ 25°C	90	90	ns min	Write/Reset Pulse Width
T_{MIN} to T_{MAX}	90	90	ns min	
t_3				
@ 25°C	0	0	ns min	Chip Select/Load DAC to Write Setup Time
T_{MIN} to T_{MAX}	0	0	ns min	
t_4				
@ 25°C	0	0	ns min	Chip Select/Load DAC to Write Hold Time
T_{MIN} to T_{MAX}	0	0	ns min	
t_5				
@ 25°C	90	90	ns min	Data Valid to Write Setup Time
T_{MIN} to T_{MAX}	90	90	ns min	
t_6				
@ 25°C	10	10	ns min	Data Valid to Write Hold Time
T_{MIN} to T_{MAX}	10	10	ns min	

NOTES
[1]Maximum possible reference voltage.
[2]Temperature ranges are as follows:
AD7224KN, LN: 0°C to +70°C
AD7224BQ, CQ: –25°C to +85°C
AD7224TD, UD: –55°C to +125°C
[3]See Terminology.
[4]Sample tested at 25°C by Product Assurance to ensure compliance.

Specifications subject to change without notice.

ABSOLUTE MAXIMUM RATINGS[1]

V_{DD} to AGND	–0.3 V, +17 V
V_{DD} to DGND	–0.3 V, +17 V
V_{DD} to V_{SS}	–0.3 V, +24 V
AGND to DGND	–0.3 V, V_{DD}
Digital Input Voltage to DGND	–0.3 V, V_{DD} + 0.3 V
V_{REF} to AGND	–0.3 V, V_{DD} + 0.3 V
V_{OUT} to AGND[2]	V_{SS}, V_{DD}
Power Dissipation (Any Package) to +75°C	450 mW
Derates above 75°C by	6 mW/°C
Operating Temperature	
Commercial (K, L Versions)	–40°C to +85°C
Industrial (B, C Versions)	–40°C to +85°C
Extended (T, U Versions)	–55°C to +125°C
Storage Temperature	–65°C to +150°C
Lead Temperature (Soldering, 10 secs)	+300°C

NOTES
[1]Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.
[2]The outputs may be shorted to AGND provided that the power dissipation of the package is not exceeded. Typically short circuit current to AGND is 60 mA.

ORDERING GUIDE

Model[1]	Temperature Range	Total Unadjusted Error (LSB)	Package Option[2]
AD7224KN	–40°C to +85°C	±2 max	N-18
AD7224LN	–40°C to +85°C	±1 max	N-18
AD7224KP	–40°C to +85°C	±2 max	P-20A
AD7224LP	–40°C to +85°C	±1 max	P-20A
AD7224KR-1	–40°C to +85°C	±2 max	R-20
AD7224LR-1	–40°C to +85°C	±1 max	R-20
AD7224KR-18	–40°C to +85°C	±2 max	R-18
AD7224LR-18	–40°C to +85°C	±1 max	R-18
AD7224BQ	–40°C to +85°C	±2 max	Q-18
AD7224CQ	–40°C to +85°C	±1 max	Q-18
AD7224TQ	–55°C to +125°C	±2 max	Q-18
AD7224UQ	–55°C to +125°C	±1 max	Q-18
AD7224TE	–55°C to +125°C	±2 max	E-20A
AD7224UE	–55°C to +125°C	±1 max	E-20A

NOTES
[1]To order MIL-STD-883 processed parts, add /883B to part number. Contact your local sales office for military data sheet.
[2]E = Leadless Ceramic Chip Carrier; N = Plastic DIP; P = Plastic Leaded Chip Carrier; Q = Cerdip; R = SOIC.

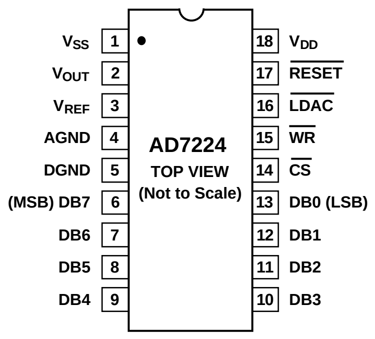
CAUTION

ESD (electrostatic discharge) sensitive device. Electrostatic charges as high as 4000 V readily accumulate on the human body and test equipment and can discharge without detection. Although the AD7224 features proprietary ESD protection circuitry, permanent damage may occur on devices subjected to high energy electrostatic discharges. Therefore, proper ESD precautions are recommended to avoid performance degradation or loss of functionality.

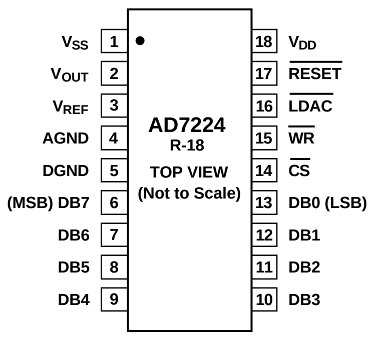
WARNING!
ESD SENSITIVE DEVICE

PIN CONFIGURATIONS

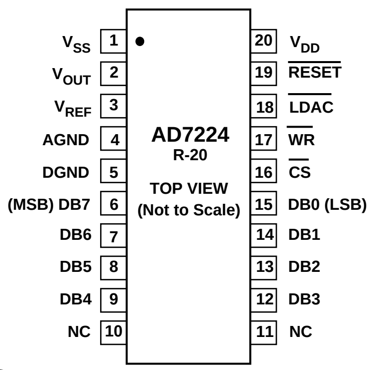
DIP and SOIC



(SOIC)

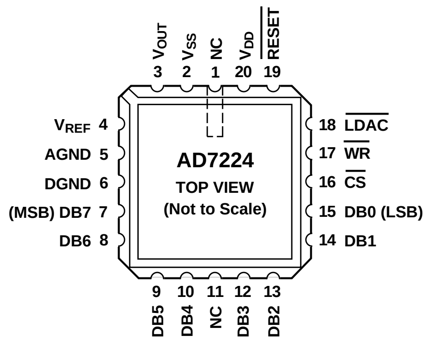


(SOIC)



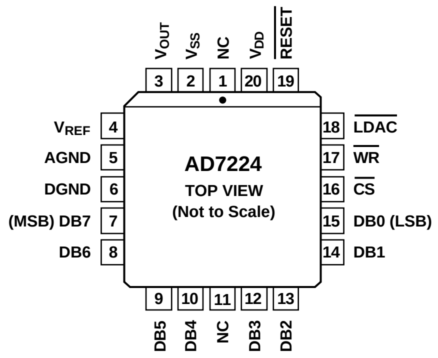
NC = NO CONNECT

LCCC



NC = NO CONNECT

PLCC



NC = NO CONNECT

TERMINOLOGY

TOTAL UNADJUSTED ERROR

Total Unadjusted Error is a comprehensive specification which includes full-scale error, relative accuracy and zero code error. Maximum output voltage is V_{REF} – 1 LSB (ideal), where 1 LSB (ideal) is $V_{REF}/256$. The LSB size will vary over the V_{REF} range. Hence the zero code error, relative to the LSB size, will increase as V_{REF} decreases. Accordingly, the total unadjusted error, which includes the zero code error, will also vary in terms of LSBs over the V_{REF} range. As a result, total unadjusted error is specified for a fixed reference voltage of +10 V.

RELATIVE ACCURACY

Relative Accuracy or endpoint nonlinearity is a measure of the maximum deviation from a straight line passing through the endpoints of the DAC transfer function. It is measured after allowing for zero code error and full-scale error and is normally expressed in LSBs or as a percentage of full-scale reading.

DIFFERENTIAL NONLINEARITY

Differential Nonlinearity is the difference between the measured change and the ideal 1 LSB change between any two adjacent codes. A specified differential nonlinearity of ±1 LSB max over the operating temperature range ensures monotonicity.

DIGITAL FEEDTHROUGH

Digital Feedthrough is the glitch impulse transferred to the output due to a change in the digital input code. It is specified in nV secs and is measured at V_{REF} = 0 V.

FULL-SCALE ERROR

Full-Scale Error is defined as:

Measured Value – Zero Code Error – Ideal Value

CIRCUIT INFORMATION

D/A SECTION

The AD7224 contains an 8-bit voltage-mode digital-to-analog converter. The output voltage from the converter has the same polarity as the reference voltage, allowing single supply operation. A novel DAC switch pair arrangement on the AD7224 allows a reference voltage range from +2 V to +12.5 V.

The DAC consists of a highly stable, thin-film, R-2R ladder and eight high speed NMOS single pole, double-throw switches. The simplified circuit diagram for this DAC is shown in Figure 1.

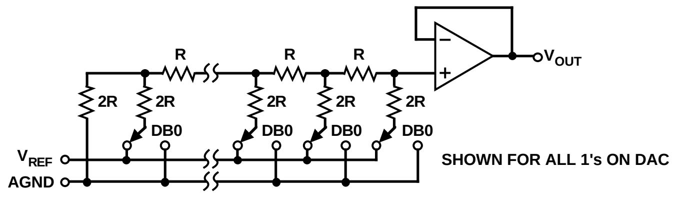


Figure 1. D/A Simplified Circuit Diagram

The input impedance at the V_{REF} pin is code dependent and can vary from 8 kΩ minimum to infinity. The lowest input impedance occurs when the DAC is loaded with the digital code 01010101. Therefore, it is important that the reference presents a low output impedance under changing load conditions. The nodal capacitance at the reference terminals is also code dependent and typically varies from 25 pF to 50 pF.

The V_{OUT} pin can be considered as a digitally programmable voltage source with an output voltage of:

$$V_{OUT} = D \bullet V_{REF}$$

where *D* is a fractional representation of the digital input code and can vary from 0 to 255/256.

OP-AMP SECTION

The voltage-mode D/A converter output is buffered by a unity gain noninverting CMOS amplifier. This buffer amplifier is capable of developing +10 V across a 2 kΩ load and can drive capacitive loads of 3300 pF.

The AD7224 can be operated single or dual supply resulting in different performance in some parameters from the output amplifier. In single supply operation (V_{SS} = 0 V = AGND) the sink capability of the amplifier, which is normally 400 µA, is reduced as the output voltage nears AGND. The full sink capability of 400 µA is maintained over the full output voltage range by tying V_{SS} to –5 V. This is indicated in Figure 2.

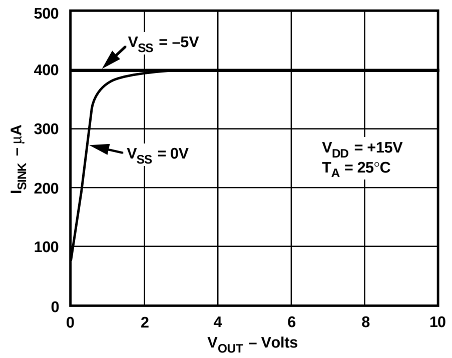


Figure 2. Variation of I_{SINK} with V_{OUT}

Settling-time for negative-going output signals approaching AGND is similarly affected by V_{SS}. Negative-going settling-time for single supply operation is longer than for dual supply operation. Positive-going settling-time is not affected by V_{SS}.

Additionally, the negative V_{SS} gives more headroom to the output amplifier which results in better zero code performance and improved slew-rate at the output, than can be obtained in the single supply mode.

DIGITAL SECTION

The AD7224 digital inputs are compatible with either TTL or 5 V CMOS levels. All logic inputs are static-protected MOS gates with typical input currents of less than 1 nA. Internal input protection is achieved by an on-chip distributed diode between DGND and each MOS gate. To minimize power supply currents, it is recommended that the digital input voltages be driven as close to the supply rails (V_{DD} and DGND) as practically possible.

INTERFACE LOGIC INFORMATION

Table I shows the truth table for AD7224 operation. The part contains two registers, an input register and a DAC register. $\overline{CS}$ and $\overline{WR}$ control the loading of the input register while $\overline{LDAC}$ and $\overline{WR}$ control the transfer of information from the input register to the DAC register. Only the data held in the DAC register will determine the analog output of the converter.

All control signals are level-triggered and therefore either or both registers may be made transparent; the input register by keeping $\overline{CS}$ and $\overline{WR}$ "LOW", the DAC register by keeping $\overline{LDAC}$ and $\overline{WR}$ "LOW". Input data is latched on the rising edge of $\overline{WR}$.

Table I. AD7224 Truth Table

$\overline{RESET}$	$\overline{LDAC}$	$\overline{WR}$	$\overline{CS}$	Function
H	L	L	L	Both Registers are Transparent
H	X	H	X	Both Registers are Latched
H	H	X	H	Both Registers are Latched
H	H	L	L	Input Register Transparent
H	H	↑	L	Input Register Latched
H	L	L	H	DAC Register Transparent
H	L	↑	H	DAC Register Latched
L	X	X	X	Both Registers Loaded With All Zeros
↑	H	H	H	Both Register Latched With All Zeros and Output Remains at Zero
↑	L	L	L	Both Registers are Transparent and Output Follows Input Data

H = High State, L = Low State, X = Don't Care.
All control inputs are level triggered.

The contents of both registers are reset by a low level on the $\overline{RESET}$ line. With both registers transparent, the $\overline{RESET}$ line functions like a zero override with the output brought to 0 V for the duration of the $\overline{RESET}$ pulse. If both registers are latched, a "LOW" pulse on $\overline{RESET}$ will latch all 0s into the registers and the output remains at 0 V after the $\overline{RESET}$ line has returned "HIGH". The $\overline{RESET}$ line can be used to ensure power-up to 0 V on the AD7224 output and is also useful, when used as a zero override, in system calibration cycles. Figure 3 shows the input control logic for the AD7224.

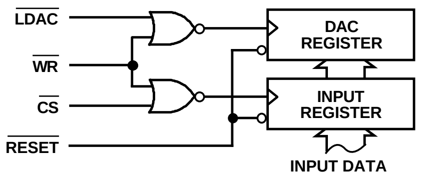
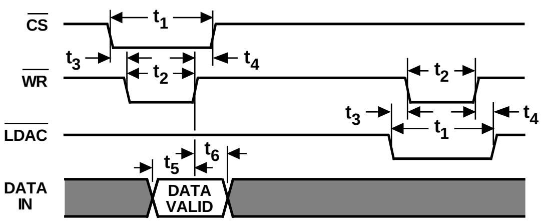


Figure 3. Input Control Logic



NOTES:
1. ALL INPUT SIGNAL RISE AND FALL TIMES MEASURED FROM 10% TO 90% OF V_{DD}. $t_r = t_f$ = 20ns OVER V_{DD} RANGE
2. TIMING MEASUREMENT REFERENCE LEVEL IS $\frac{V_{INH} + V_{INL}}{2}$

Figure 4. Write Cycle Timing Diagram

SPECIFICATION RANGES

For the DAC to maintain specified accuracy, the reference voltage must be at least 4 V below the V_{DD} power supply voltage. This voltage differential is required for correct generation of bias voltages for the DAC switches.

With dual supply operation, the AD7224 has an extended V_{DD} range from +12 V ± 5% to +15 V ± 10% (i.e., from +11.4 V to +16.5 V). Operation is also specified for a single V_{DD} power supply of +15 V ± 5%.

Performance is specified over a wide range of reference voltages from 2 V to (V_{DD} – 4 V) with dual supplies. This allows a range of standard reference generators to be used such as the AD580, a +2.5 V bandgap reference and the AD584, a precision +10 V reference. Note that in order to achieve an output voltage range of 0 V to +10 V, a nominal +15 V ± 5% power supply voltage is required by the AD7224.

GROUND MANAGEMENT

AC or transient voltages between AGND and DGND can cause noise at the analog output. This is especially true in microprocessor systems where digital noise is prevalent. The simplest method of ensuring that voltages at AGND and DGND are equal is to tie AGND and DGND together at the AD7224. In more complex systems where the AGND and DGND intertie is on the backplane, it is recommended that two diodes be connected in inverse parallel between the AD7224 AGND and DGND pins (IN914 or equivalent).

Applying the AD7224

UNIPOLAR OUTPUT OPERATION

This is the basic mode of operation for the AD7224, with the output voltage having the same positive polarity as V_{REF}. The AD7224 can be operated single supply (V_{SS} = AGND) or with positive/negative supplies (see op-amp section which outlines the advantages of having negative V_{SS}). Connections for the unipolar output operation are shown in Figure 5. The voltage at V_{REF} must never be negative with respect to DGND. Failure to observe this precaution may cause parasitic transistor action and possible device destruction. The code table for unipolar output operation is shown in Table II.

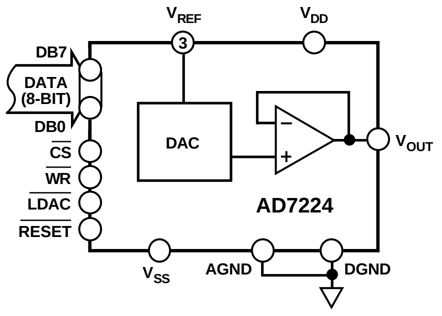


Figure 5. Unipolar Output Circuit

Table III. Unipolar Code Table

DAC Register Contents MSB	LSB	Analog Output
1 1 1 1	1 1 1 1	$+V_{REF}\left(\frac{255}{256}\right)$
1 0 0 0	0 0 0 1	$+V_{REF}\left(\frac{129}{256}\right)$
1 0 0 0	0 0 0 0	$+V_{REF}\left(\frac{128}{256}\right) = +\frac{V_{REF}}{2}$
0 1 1 1	1 1 1 1	$+V_{REF}\left(\frac{127}{256}\right)$
0 0 0 0	0 0 0 1	$+V_{REF}\left(\frac{1}{256}\right)$
0 0 0 0	0 0 0 0	$0\ V$

Note: $1\ LSB = (V_{REF})(2^{-8}) = V_{REF}\left(\frac{1}{256}\right)$

BIPOLAR OUTPUT OPERATION

The AD7224 can be configured to provide bipolar output operation using one external amplifier and two resistors. Figure 6 shows a circuit used to implement offset binary coding. In this case

$$V_O = \left(1+\frac{R2}{R1}\right)\cdot\left(D\ V_{REF}\right)-\left(\frac{R2}{R1}\right)\cdot\left(V_{REF}\right)$$

With $R1 = R2$

$$V_O = (2\,D - 1)\cdot V_{REF}$$

where D is a fractional representation of the digital word in the DAC register.

Mismatch between R1 and R2 causes gain and offset errors; therefore, these resistors must match and track over temperature. Once again, the AD7224 can be operated in single supply or from positive/negative supplies. Table III shows the digital code versus output voltage relationship for the circuit of Figure 6 with R1 = R2.

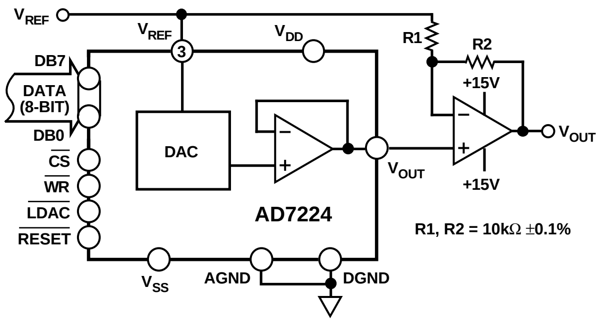


Figure 6. Bipolar Output Circuit

Table III. Bipolar (Offset Binary) Code Table

DAC Register Contents MSB	LSB	Analog Output
1 1 1 1	1 1 1 1	$+V_{REF}\left(\frac{127}{128}\right)$
1 0 0 0	0 0 0 1	$+V_{REF}\left(\frac{1}{128}\right)$
1 0 0 0	0 0 0 0	$0\ V$
0 1 1 1	1 1 1 1	$-V_{REF}\left(\frac{1}{128}\right)$
0 0 0 0	0 0 0 1	$-V_{REF}\left(\frac{127}{128}\right)$
0 0 0 0	0 0 0 0	$-V_{REF}\left(\frac{128}{128}\right)=-V_{REF}$

AGND BIAS

The AD7224 AGND pin can be biased above system GND (AD7224 DGND) to provide an offset "zero" analog output voltage level. Figure 7 shows a circuit configuration to achieve this. The output voltage, V_{OUT}, is expressed as:

$$V_{OUT} = V_{BIAS} + D\cdot(V_{IN})$$

where D is a fractional representation of the digital word in DAC register and can vary from 0 to 255/256.

For a given V_{IN}, increasing AGND above system GND will reduce the effective V_{DD}–V_{REF} which must be at least 4 V to ensure specified operation. Note that V_{DD} and V_{SS} for the AD7224 must be referenced to DGND.

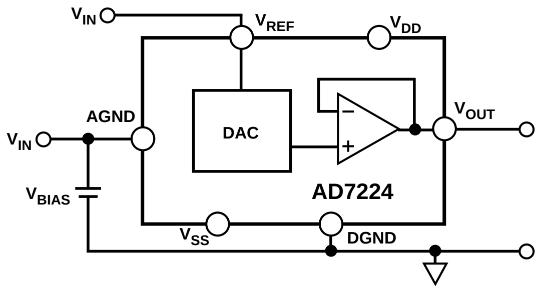


Figure 7. AGND Bias Circuit

MICROPROCESSOR INTERFACE

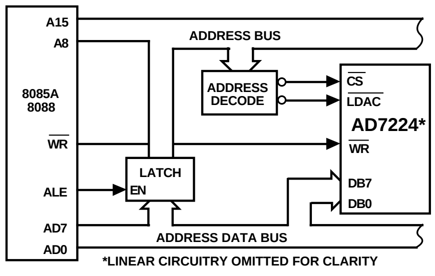


Figure 8. AD7224 to 8085A/8088 Interface

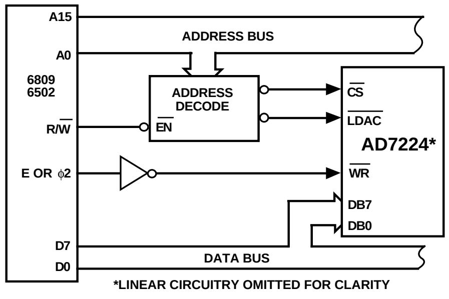


Figure 9. AD7224 to 6809/6502 Interface

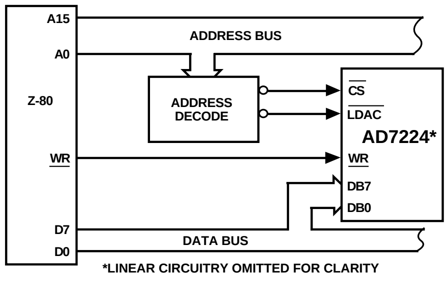


Figure 10. AD7224 to Z-80 Interface

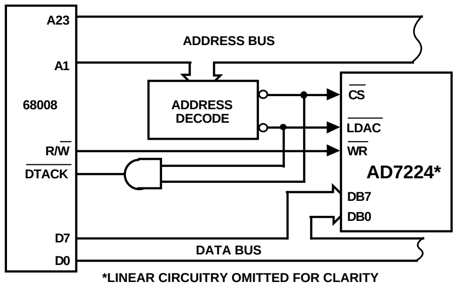
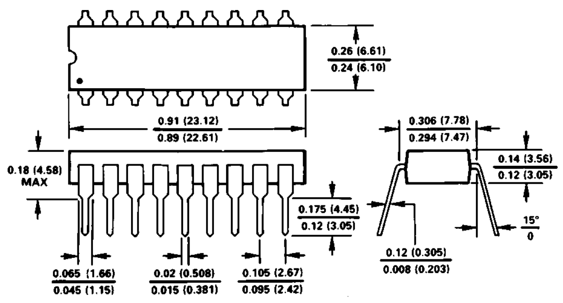


Figure 11. AD7224 to 68008 Interface

OUTLINE DIMENSIONS

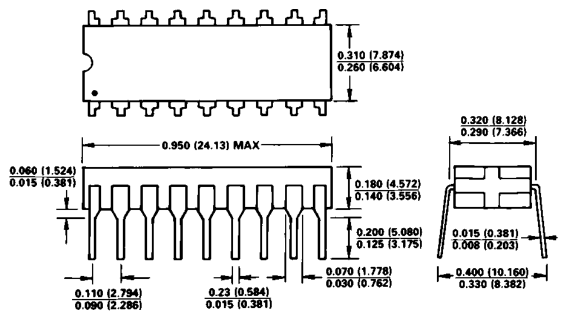
Dimensions shown in inches and (mm).

18-Pin Plastic (Suffix N)



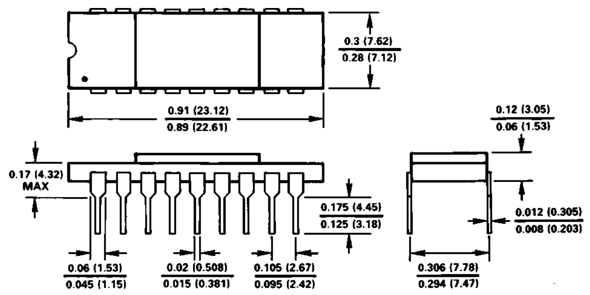
NOTES:
1. LEAD NO. 1 IDENTIFIED BY DOT OR NOTCH.
2. CERAMIC DIP LEADS WILL BE EITHER GOLD OR TIN PLATED IN ACCORDANCE WITH MIL-M-38510 REQUIREMENTS.

18-Pin Cerdip (Suffix Q)



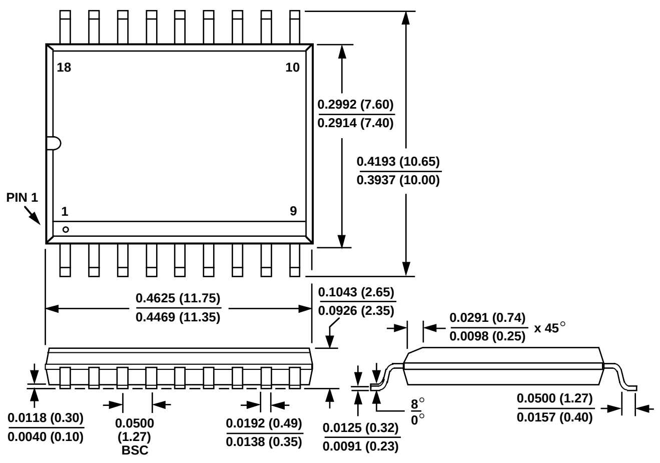
NOTES:
1. LEAD NO. 1 IDENTIFIED BY DOT OR NOTCH.
2. CERAMIC DIP LEADS WILL BE EITHER GOLD OR TIN PLATED IN ACCORDANCE WITH MIL-M-38510 REQUIREMENTS.

18-Pin Ceramic (Suffix D)

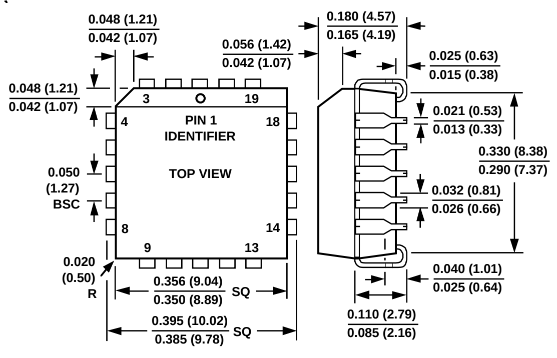


NOTES:
1. LEAD NO. 1 IDENTIFIED BY DOT OR NOTCH.
2. CERAMIC DIP LEADS WILL BE EITHER GOLD OR TIN PLATED IN ACCORDANCE WITH MIL-M-38510 REQUIREMENTS.

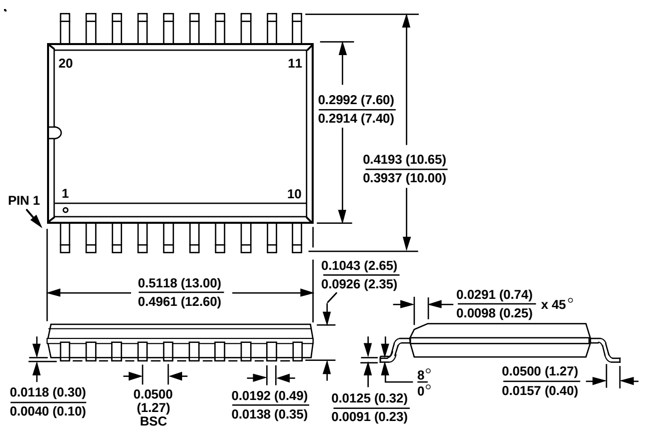
18-Lead SOIC (R-18)



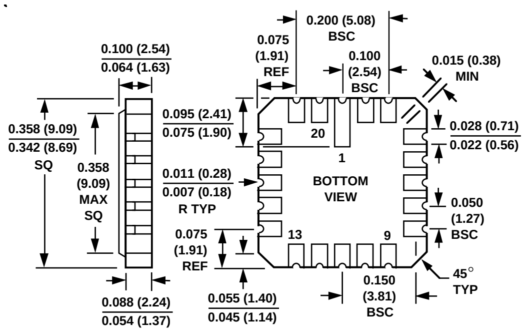
PLCC Package P-20A



20-Lead SOIC (R-20)



LCCC Package E-20A



C836a–10–10/84

PRINTED IN U.S.A.