













 FilipPaul	lepsi obrazek simulace ...	8 minutes ago	 History
..			
	.vscode	project	21 days ago
	include	project	21 days ago
	lib	comment	2 hours ago
	pictures	No commit message	9 minutes ago
	src	comments	2 hours ago
	test	project	21 days ago
	.gitignore	project	21 days ago
	platformio.ini	project	21 days ago
	project.simu	comments	2 hours ago
	projectSimV2.simu	readme	26 minutes ago
	project_backup.simu	No commit message	9 minutes ago
	readme.md	lepsi obrazek simulace	8 minutes ago

readme.md 

Project DE-2

Authors: [Kristýna Pijáčková](#) ; [Filip Paul](#)

Project description

Function description R-2R ladder Digital to Analog Converter (DAC). Application of analog signal generator using at least 8-bit DAC; several preset signal types; display; 4x3 keypad control; (possibility of DTMF, Dual-Tone Multiple Frequencies tone generation); sending interesting information about the status of the application to the UART.

Project objectives

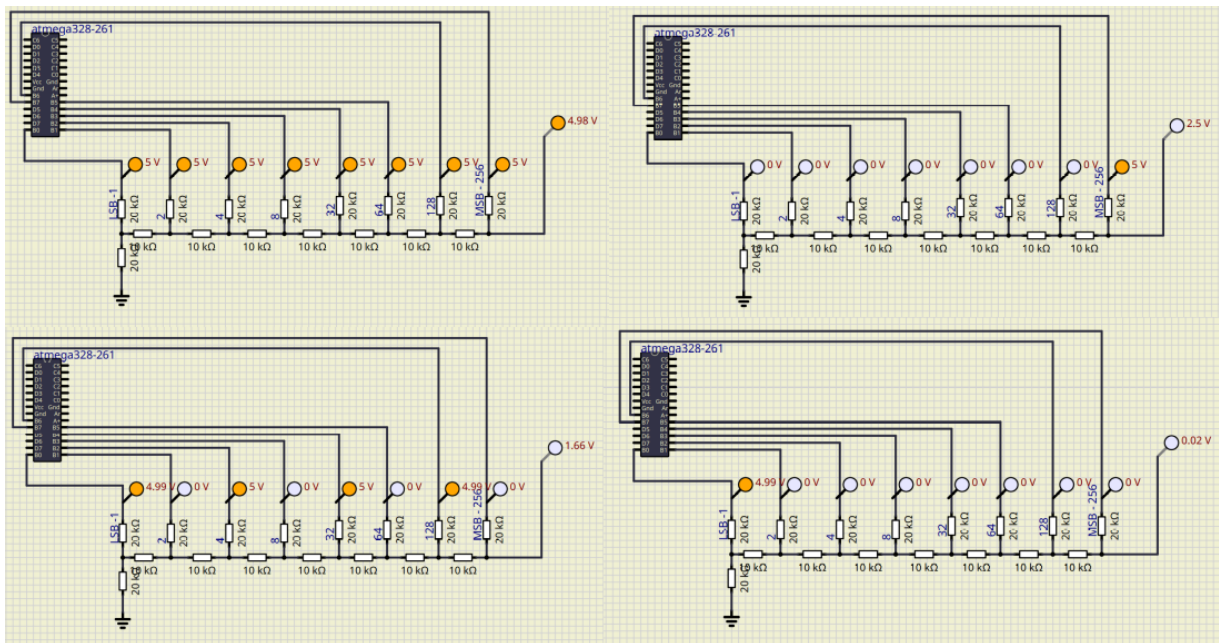
Application of analog signal generator using 8-bit R-2R DAC with several preset signal types. User interface: 4x3 keypad control, 2x16 LCD, UART.

Hardware description

The brain of the generator is the 8-bit microcontroller Atmega238p. This MCU is soldered with few more components (voltage regulators, 16MHz oscillator, USB interface etc.) on the premade board called Arduino Uno. ATMEGA 328p has 21 GPIO pins. These pins can be accessed by registers PORTB, PORTC and PORTD. In our case we will use PORTD pins to control LCD with Hitachi HD44780 driver, PORTB as input to 8-bit R-2R DAC and PORTC to scan 4x3 keypad matrix

R-2R 8-bit DAC

R-2R is basically a voltage divider created from resistors of 2 values. Working principle is pictured on following figure. The figure pictures four different examples - it can be seen which outputs of the AVR are activated (5V input into the resistors) and which are not, the actual output voltage can be at the most right probe.



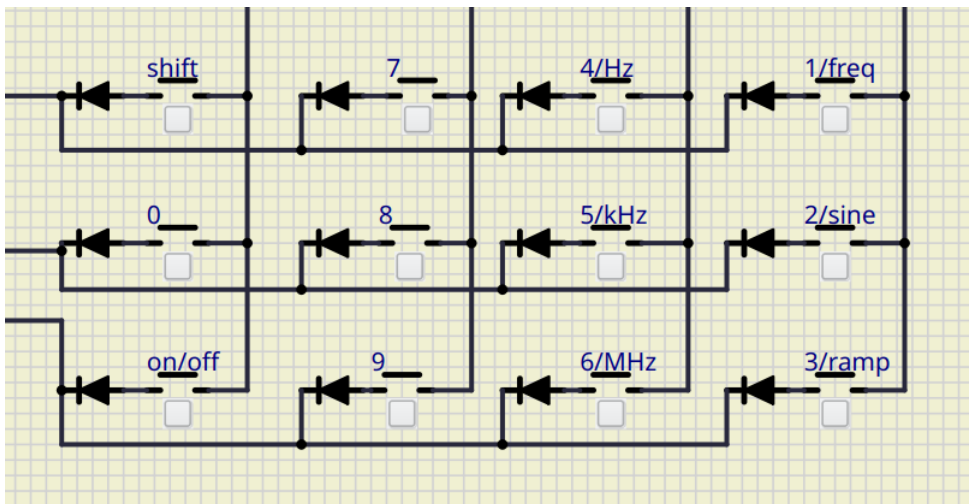
The output of this DAC is given by the following equation

$$V_{out} = 5V \cdot \frac{value}{2^8}$$

The DAC output is the output of the Generator with three possible output waveforms - sine, square and ramp in frequency range 15Hz to 1000Hz.

4x3 keypad matrix

Key pad matrix is an array of buttons, where each button is connected to the 2 GPIOs, as can be seen at fig. below. Connecting buttons in this pattern reduces the number of used GPIOs. With this method, we can use only 7 (3 rows x 4 columns) pins to control 12 buttons. If we want to get the position of the pressed button, we need to set all columns as inputs with the internal pullup resistors and rows as output with HIGH logic level. The next step is to loop through all rows in the following pattern: ROW(i) = LOW, check all columns, if the column is LOW, then the button has position row(i) x column(LOW). If none of the columns is LOW set ROW(i) back to HIGH and go to the next row.



Code description

User Interface

For purpose of controlling the 4x3 matrix, we have written a library called "myMatrix". This library contain these functions:

function	input	output	Description
initMatrix()	none	none	set columns as inputs with pullup resistors, and rows as output HIGH
scanMatrix()	none	uint8_t pos	returns position in form of integer of pushed button EX: 23 -> row 2 x column 3; returns 0 if nothing was pressed
posToConstChar(uint8_t pos, bool shift)	uint8_t pos bool shift	const char* button_name	returns button name according to the position and also takes into account if shift button was pressed

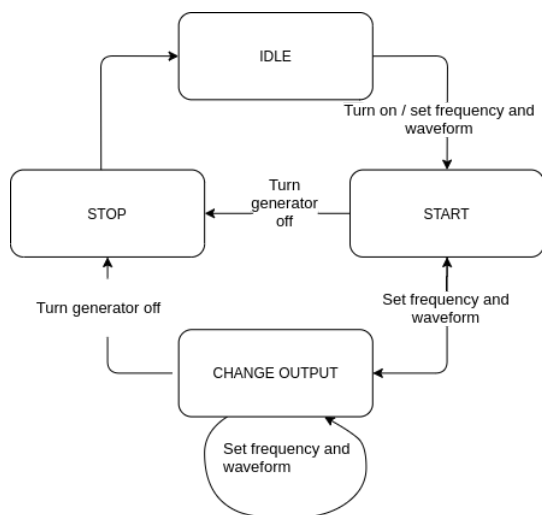
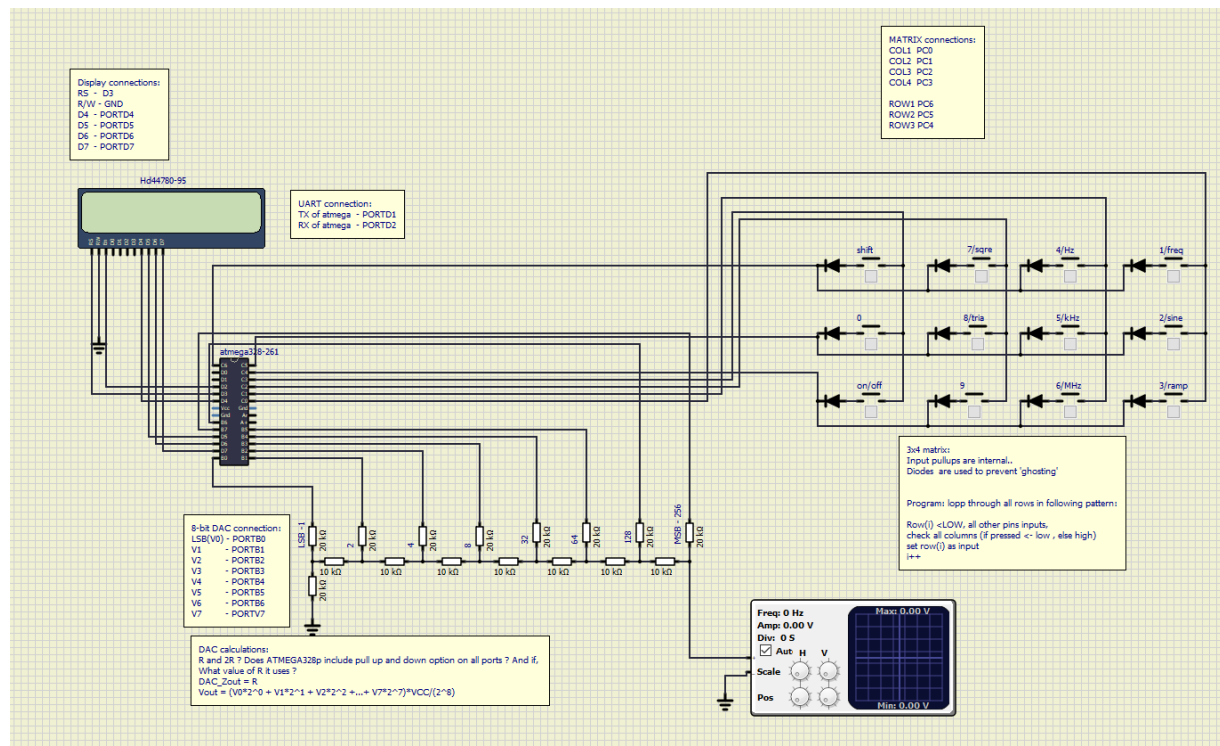
To scan matrix. function scanMatrix() is called in regular interval given by Timer1 overflow each 33ms. The display, UART and parameters like frequency, output status and waveform are then updated according to the pushed button.

Please note: for UART transmission it is necessary to uncomment the relevant lines, because with UART simul IDE sometimes crashes unexpectedly.

Waveform generator

The generator is realized in the main.c file using counter frequency 7812Hz given by the Timer0, which is set to 128us. After user enters wanted frequency and waveform, output frequency is given by calculating stepsize, which helps to speed up counting of a counter, which regularizes the frequency. A value between 0 and 255 is sent to PORTB, which is set as output and is connected to the 8bit R-2R DAC. Oscilloscope connected to the output of the DAC allows to watch the different waveforms.

Schematics and diagram



Results discussion

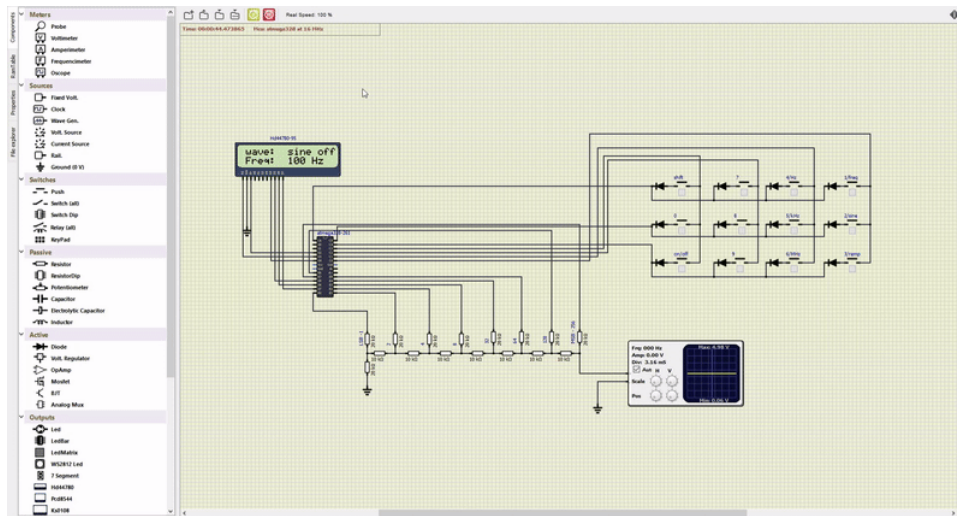
We created a simple Waveform Generator by using the 8-bit microcontroller Atmega238p and R-2R 8-bit digital to analog converter. There are three waveforms to choose from - sine, square and ramp. The frequency they can be set to is in range from 15Hz to 1kHz. The generator also includes a keypad, containing ON/OFF button, 0-9 and waveform/frequency keys and shift button to switch between numerical keys and keys holding waveforms and frequency multipliers (Hz, kHz, MHz). It further includes a LCD display which informs the user of the current situation - setting the frequency/waveform, status of the generator - on/off, etc... The code is also written so, that it contains lines of code to send data to uart, however the Simul IDE would freeze unexpectedly when these lines were included, they were therefor commented so that the simulation can run "smoothly".

This generator was called "simple" since there is a huge potential to enlarge it and fine-tune it. E.g. the frequency range could be much wider, just by using either larger look-up table, or by using the 16us counter. Much larger look-up table, however, did not allow us to create a .HEX file and usage of 16us counter messed up the LCD completely. Further a triangle function could be easily added (and is included in the code, but commented out), however we found it best worked with another look-up table for triangle, but unfortunately by adding it, we would not be able to compile the code. Also the frequency might be slightly off at some values as the step size is a rounded number (as C does not work well with floating points - <https://hackaday.com/2016/08/10/fun-audio-waveform-generator-is-more-than-the-sum-of-its-parts/?fbclid=IwAR0Nk2zbO9rXzHuYk9N9j1dR9Zuk-eaXMND-rjAb6OhCiWJ3W1Z2zMQk1-Y> gives you the idea, how it might be solved by using shifting.)

Most challenging however, was unfortunately the work with SimulIDE... when running the same simulation with same .HEX file in one version the program would have problems working with keypad and lcd, but was fine with the generator, another version at different laptop was alright with keypad and lcd, but did not show anything at the output of the generator, which we found rather strange. Also, the same .HEX file does not run on simulation which contains text boxes for schematics description in it, but does without them and adding additional probe to measure voltage at the wires significantly worsend flattering of the oscilloscope.

Eventhough the listing of what could be improved might seem to be quite long, these all are just notes and our generator works just fine. After all, see for yourself at the included gif below... :)"

Simulation



Reference links

<https://hackaday.com/2016/08/10/fun-audio-waveform-generator-is-more-than-the-sum-of-its-parts/?fbclid=IwAR0Nk2zbO9rXzHuYk9N9j1dR9Zuk-eaXMND-rjAb6OhCiWJ3W1Z2zMQk1-Y> https://en.wikipedia.org/wiki/Resistor_ladder