Project Description

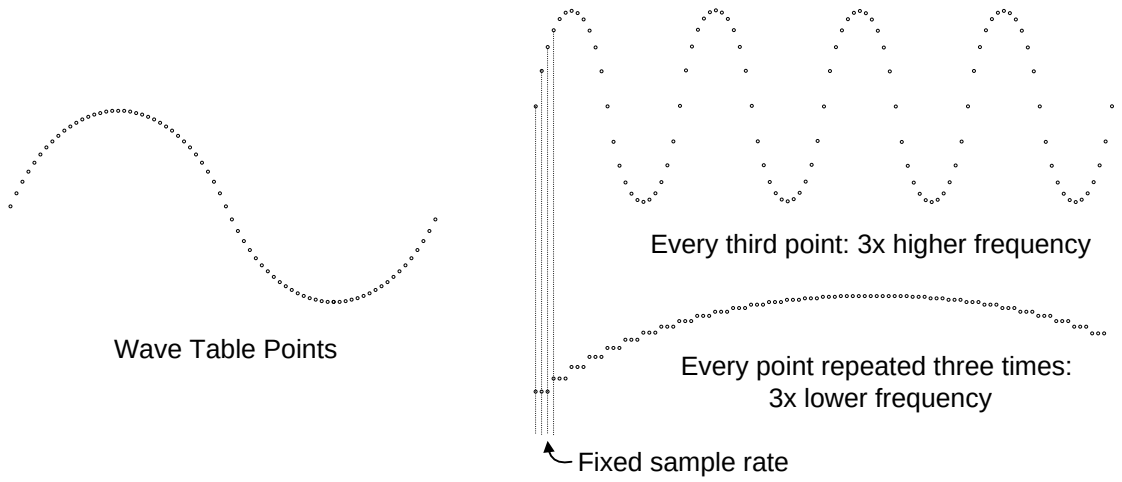
Waveform synthesis using a wave table

Periodic digital waveforms with arbitrary frequency content can be produced using a table of stored data that describes only a portion of the total waveform. The memorized waveform segment, commonly referred to as a wave-table, is typically comprised of a single period of a periodic wave from 0^o to 360^o (or, if the wave is symmetric, just 90^o must be stored since only the two MSBs must change to define quadrant). The wavetable is typically "power of 2" sized, so that as addresses are incremented beyond the wavetable's upper boundary, the lower bits that form the address alias back into the wavetable address space.

The type or shape of waveform stored in memory is typically matched to the requirements of the wave to be produced. For example, a sine wave of any frequency can be produced from wavetable data describing a sine-wave, but it would be difficult to produce the abrupt transitions of a saw-tooth waveform from a sine-wave wavetable.

Wavetable data can be used to produce a waveform in two different ways. Every point in the wavetable can be accessed in succession using a ***variable access rate***, or the points in the wavetable can be accessed using a ***constant access rate***, but with not every stored point being accessed in exact succession. Using the variable access rate method, if 1024 points are stored in a wave-table, and that data must be used to produce a 100Hz sine wave, then data points would be accessed at rate of 100Hz x 1024 or about 100KHz. If that same wave-table data were used to produce a 1KHz waveform, then data points would be accessed at a rate of about 1MHz. Using this method, a simple counter with a variable clock is used to create the wavetable address.

Using the constant access rate method, new data values are read from the wavetable at a fixed frequency, but each and every point isn't necessarily read in sequence – some points may be skipped over to create higher frequency outputs, or the same point may be read multiple times to create lower frequency outputs. For example, assume data in a 1024-point wavetable is accessed at a constant 100KHz rate. If all points are read in succession at 100KHz, a waveform of 100KHz/1024 or about 100Hz would be produced. If every third point were read at the same rate (so two points are skipped between each point that is read), then a waveform of 100KHz/(1024/3) or about 300Hz would be produced. If each point were read three times in a row before moving to the next address, then a waveform of (100Hz/1024*3) or about 33Hz would result.

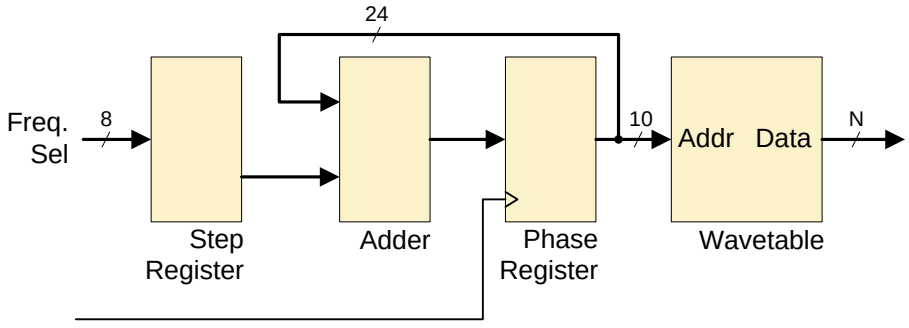


In the constant access rate method, the wavetable access frequency f_c remains constant, so new addresses into the wavetable must be computed – they can no longer be created by a variable-clock counter. New wavetable addresses are computed by adding some integer to the current address. The higher the number added to the address, the more points are skipped and the higher the resulting output frequency. In the simplest case, an N-bit register holds the current wavetable address, and an integer (0, 1, 2, 3, 11, 19 or whatever) is added to the address register each time a new point is accessed. But simply adding the same integer (M) would limit the output frequencies f_o that can be created to integer multiples of the access frequency f_c divided by the number of points in the wavetable (N):

$$f_o = \frac{M \cdot f_c}{N}$$

A better method is to allow different integers to be added to the address between each access. For example, with a 1024-point wavetable and a constant time base of 100KHz, taking every point in succession would result in a 100Hz waveform (100KHz / 1K points). Taking every other point would result in a 200Hz waveform (100KHz/500 points), and taking every third point would result in a 300Hz output. But what if an output of 250Hz is needed? That could be accomplished by adding 2, then 3, then 2, then 3, and so on. What about 255Hz?

To generate wavetable addresses suitable for a wide variety of output frequencies from a fixed wavetable, a "phase accumulator" can be used. A phase accumulator is a register that contains more bits than are needed for a wavetable address, and an adder to add some amount to the accumulator each time a new address is needed. Only the upper, most-significant bits of the accumulator are used as a wavetable address – the lower bits "accumulate" information from adding new offsets to the wavetable address. As new numbers are added into the phase accumulator, the upper bits (the wavetable address) may or may not change, but information is always accumulated in the lower bits. For example, a typical accumulator might be 24 bits, but only the upper 10 bits are needed to address a 1024-entry wavetable. If an added amount is less than 2^{14}, then the upper 10-bits (the wavetable address bits) may not change with that addition, but that added amount accumulates in the lower-order bits. This accumulated information can contribute to the upper bits changing with the next addition.



The amount (M) added into the accumulator changes based on the desired output frequency. The equation below shows the relationship between the output frequency f_o, the addend M, the time base clock frequency f_c, and the largest number an n-bit accumulator can hold (2^n).

$$f_o = \frac{M \cdot f_c}{2^n}$$

Using a 16-bit accumulator and a 44.1KHz clock frequency, the equation simplifies to:

$$f_o = \frac{M \cdot 44{,}100}{65{,}536} \quad \text{or} \quad f_o = M \cdot 673$$

If we use a 14-bit number for M (0 to 16,384), then we can define frequencies from 0 to about 11KHz. As an example, let's say M = 1486, giving an output frequency f_o of 1KHz. This means if we add 1486 into the accumulator at a rate of 44.1KHz, the upper 10 bits of the 16-bit accumulator will pass through their range (0 to 1024) at a rate of 1KHz. Since these 10 bits are used as the address into a 1024-point sine wavetable, a 1KHz sine-wave will be produced. If we later change M to 2972, then the upper 10 bits will go through their range at a rate of 2KHz and a 2KHz sine wave will be produced.

Note that if M = 64 (the largest 6-bit binary number), then after 1024 successive additions each point in the wavetable will be visited exactly once. At 44.1KHz, 1024 additions results in an output frequency of 44100/1024, or 43Hz (which agrees with the formula $f_o = M \cdot 673$ above). At frequencies below 43Hz, the amount M added into the accumulator isn't large enough to cause the wavetable address to change with each addition, and the same wavetable point will be accessed on successive reads. At frequencies above 43Hz, the wavetable address may be changed by an amount greater than 1 and therefore some wavetable points will be skipped. At any given output frequency f_o, the number of samples or points from the wavetable (p) used to create one period is given by

$$p = \frac{2^n}{M}, \text{ or equivalently, } p = \frac{f_c}{f_o}$$

where 2^n is the largest number the accumulator can hold, M is the addend required to create the output frequency f_o, and f_c is the wavetable access frequency. Nyquist criteria require that p be greater than 2 to create a minimally recognizable output waveform, but in practice, the larger p is, the better (typically, at least 10 points per period is considered a comfortable minimum). In our example, if our goal is to create waveforms up to 10KHz using a 1024-point wavetable, a 16-bit accumulator and a 44.1KHz access frequency, we end up with

$$p = \frac{2^{16}}{14859},$$

or just 4 points per period at10KHz. Although this is above Nyquist, it will result in a 10KHz waveform with lots of quantization noise. To get more points per waveform, we must increase the access frequency f_c into the wavetable.

Output to PWM

Waveform data from the lookup table can be sent directly to the PWM circuit for conversion into an analog signal. To get the best possible analog output, the PWM should be run at the highest possible frequency. This is to separate the PWM carrier frequency (the "window" frequency) from the encoded analog waveform frequency by the widest possible margin, to make output filtering more efficient. Given a maximum system clock f_{sysclk} and n bits of data in the wavetable, the maximum PWM frequency is

$$f_{PWM} = \frac{f_{sysclk}}{2^n}$$

If the wavetable stores 8-bit data and we have a maximum clock frequency of 50MHz, then maximum f_{PWM} of 195KHz is possible. If the synthesizer produces data above this frequency, the PWM cannot consume it; if the synthesizer produces data below this frequency, then the potential to create higher fidelity waveforms is being wasted.

Project Requirements

Problem 1

Create a wavetable-based synthesizer that can produce sine waves in the 100Hz – 10KHz range. Connect the synthesizer to your PWM circuit (and to the analog filter), and use a scope to validate the output. Run the PWM at the maximum possible window frequency (without multiplying the clock to a higher frequency than is directly available on your board).

You will find a .csv file (comma-separated value) file on the website that contains 1024 8-bit entries – you can make that into a ROM in Verilog or VHDL for inclusion in your project.

When validating your output, be sure 100Hz, 10KHz, and some intermediate frequency sine waves are free of distortion and have minimal noise.

Extra Credit

Keeping your PWM window frequency the same, use a 10-bit wavetable instead of an 8-bit table. You will need a system clock frequency that is four times faster to account for the extra sample bits, which may mean you need to run your synthesizer faster as well. You can get higher clock frequencies by using the clock management tiles in the FPGA.

Working Design Submission

I am submitting my own work in this project. I know penalties will be assessed if I submit work for credit that is not my own.

Point Scale: 4 – Exemplary; 3 – Complete; 2 – Incomplete; 1 – Minor Effort; 0 – Not Submitted

Print Name _____ Sign Name _____ Date _____

#	***Deliverable***	Wt	Pts	Score	Lab Assistant Signature	Date	Wks Late
P1	Circuit demonstration	20					
	All source files present; well commented; well partitioned; easy to follow	6					
	Verbal questions answered well	6					
EC	Circuit demonstration	5					
	All source files present; well commented; well partitioned; easy to follow	1					
	Verbal questions answered well	1					

Optional Report	Wt	Pts	Score	Lab Assistant Signature	Date	Wks Late
Design intent/problem clear	1					
Good diagrams, well written partition description	1					
Well written technical descriptions	1					
Signals/blocks identified and described	1					
Good state & other diagrams	1					

Non-Working Design Submission

I am submitting my own work in this project. I know penalties will be assessed if I submit work for credit that is not my own.

Point Scale: 4 – Exemplary; 3 – Complete; 2 – Incomplete; 1 – Minor Effort; 0 – Not Submitted

Print Name	Sign Name	Date

List and describe what is working:

List and describe what is not working:

State what you would do if you continued on this design:

Required Report	Wt	Pts	Score	Lab Assistant Signature	Date	Wks Late
Design intent/problem clear	3					
High-level design-to specs and design context clear	2					
Design status clear (working and non-working)	2					
Good background materials	5					
Good diagrams, well written partition description	3					
Well written technical descriptions	5					
Signals and blocks identified and described	2					
Good state diagrams & other diagrams	3					
Good discussion of what went right and wrong	3					
Meaningful plan of action for continued work	2					
Meaningful conclusion	2					