# Build your own FM transmitter with an FPGA

Nina Engelhardt

December 7, 2014

## The things we want to work our magic on

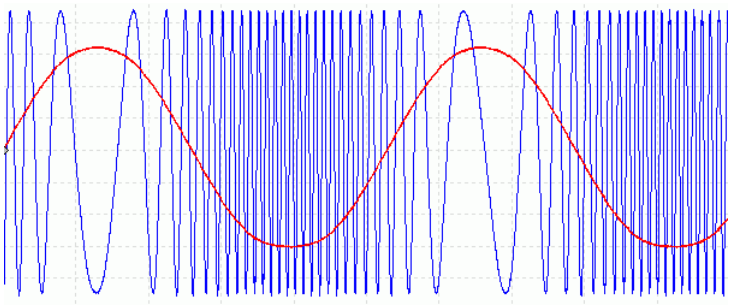
*Sound:* variation of amplitude of pressure waves at frequencies between 20Hz and 20KHz.

*FM Radio:* uses carrier wave of much higher frequency (87-106MHz), encodes sound amplitude as variation in frequency around the central carrier frequency. Each channel has a 150KHz band allocated to it, split into 75KHz above and 75KHz below the base frequency.

## So how does that work?

Let's examine what would happen if you were to tune your radio to the frequency 90MHz and somewhere close by, someone were to transmit a pure A (440Hz) on this frequency at the loudest possible setting (making use of the full bandwith):[1]

A tone at 440Hz has a period of 2.3 miliseconds. This means you will see the frequency change from 90 075 KHz to 89 925 KHz and back to 90 075 KHz again every 2.3 ms. On an oscilloscope, it looks a bit like the blue line here:



The frequency variation is greatly exaggerated. Red represents the encoded sound wave.

We will approximate this signal by a square wave, since our FPGA is a digital circuit and thus it is much easier to generate 0 and 1 than any intermediate value. Also, we are

[1] You would presumably wince.

already stretching the limits of the tiny board a bit just by switching between 0 and 1 that often, so never mind trying to influence how we get from one to the other.

The main frequency, our carrier signal, will be unaffected by this, but we will be generating a number of parasite frequencies in other bands. This is a bit impolite, but we'll keep it inside this room. Shhh. No one will notice.[2]

# 1 Generating the carrier signal

We've already learned how to make something switch back and forth between 1 and 0 every X seconds in the previous example of a counter. Let's extend this approach to generate our carrier wave.

The first thing we need is a faster clock, since at the default clock rate of 32MHz we won't be able to generate anything close to the FM spectrum. Inside the FPGA, there is a special component called "Digital Clock Manager" (DCM) that can be used to generate other clock frequencies. This component has already been instantiated for you in a class called `ClockGen` at the top of your file. The new clock runs at 300MHz.

You will be making a new counter inside the class `PhaseAccumulator` that's been prepared for you. The output signal has been created for you already, and you will define the inputs yourself later. The output should be, as before, the highest bit of the counter.

Create the counter as in the exercise before, simply adding 1 each cycle.

If we call $n$ the number of bits in the counter, at what frequency will the highest bit switch? Express the frequency in terms of $n$ and the clock frequency $f_{clk}$.

$f(n) =$

Is it possible to choose a bitwidth for the counter that will result in our desired carrier frequency, 90MHz?

What if instead of adding 1 each cycle, we add a number $m$? Express the output frequency in terms of $f_{clk}$, $n$ and $m$.

$f(n, m) =$

Which of the three parameters $n$, $m$, and $f_{clk}$ can be changed at runtime? Why?

At this point there are many possible choices for $n$ and $m$. We'll pick them later, because we have some additional requirements to think of.

[2] They won't. Our transmitter is not very strong, it doesn't even reach the other side of the room.

## 2 Making the frequency vary (frequency modulation)

Ultimately, this component needs to be able to produce 256 discrete frequencies linearly covering a 150KHz range. (This corresponds to the 256 possible values of an 8-bit music sample.)

How far apart are two neighboring frequencies in this range?

$\Delta_f =$

Given a certain frequency $f_0$ determined by a given set of parameters $f_{clk}$, $m$ and $n$, and respecting which parameters you determined to be fixed at runtime, and which are modifiable, what is the closest distinct output frequency you can generate?

$f_1 =$

We want the difference between two possible values of the output frequency to be the same as the distance $\Delta_f$ between two neighboring frequencies of the sample range determined above.

Express this as an equation using $\Delta_f$, $f_0$ and $f_1$:

Replace $\Delta_f$, $f_0$ and $f_1$ in this equation with their previously determined expressions and solve for $n$:

Our counter can't have fractional bits. What is the closest integer value for $n$ ($f_{clk}$ is 300MHz)?

$n =$

Given this value of $n$, what value of $m$ would generate the central carrier wave frequency $f_{center}$? Express it using $n$, $f_{clk}$ and $f_{center}$:

$m_{center} =$

This will correspond to a sample value of 0.

Given a sample value $s$ in the range $[-128, 127]$, express $m$ using $m_{center}$, and $s$.

$m =$

# 3 Implementation

## 3.1 Defining the phase accumulator

Implement the counter with your chosen value of $n$ and the input $m$ in the module `PhaseAccumulator`. You need to define the variables `m` and `n`. Think carefully about how: things that change at runtime should be signals, things that are fixed at construction time should be integer parameters of the class constructor (no hardcoding please!). In addition, the variable `m` needs to be settable from the exterior of the module, to change the frequency according to the sound sample currently being played.

## 3.2 Using the phase accumulator

Inside the module `FMTransmitter`, towards the bottom, insert the values for the variables `transmit_freq` (the central carrier wave frequency around which we will modulate, in Hz) and `phaseacc_nbits` (the width of the counter inside the phase accumulator that you determined above).

Instantiate the phase accumulator after the definition of the variables. To instantiate a module inside another module, you need to add it to the special attribute `submodules` of the outer module. You can either do this using the `+=` operator if you do not need to access the submodule again, or give it a name: `self.submodules.name = Module()`. In this case you can then access the submodule as `self.name`.

Finally, connect the inputs and outputs of the phase accumulator submodule. Calculate $m$ according to the formula you have determined above. The value of the current sample can be found in a signal named `sample` that has already been defined for you. (You can find it with the UART instantiation a few lines above.) Connect the ouput of the phase accumulator to the antenna signal requested from the platform.

## 3.3 Testing your design

Build your design and program it from the lab computer as before. If you tune the radio to the chosen center frequency of 90MHz, you should now receive silence instead of static.

Open a terminal and launch the command `minicom`. Access the configuration screen (C-a o) and select "Serial Port Setup". Configure the serial link at speed 115200 baud and parity 8N1 (8 bit data, no parity bit, 1 stop bit). With this configuration, it takes 10 symbols (bits) to transfer one sample: start bit, 8 bits of data, stop bit. At a speed of 115200 baud (baud = symbols per second), this means 11520 samples will be transferred per second. We have prepared a sound file encoded in signed 8 bit with a sample rate of 11520Hz. Use the command C-a s (send file) to transmit the file to the FPGA. If everything went well, you should hear a familiar song[3] play on the radio!

[3]It's not a rickroll, I promise.