

FPGAs: BUILD YOUR OWN CUSTOM SOUND CHIP

BEN EVERARD

Mine Bitcoins or make silly noises: anything's possible when you have the power to design your own chips.

WHY DO THIS?

- Make funky noises
- Create hardware customised for your exact needs
- Peek into the innards of chips and learn about IC design

Field Programmable Gate Arrays (FPGAs) are chips that can be programmed to perform different actions. However, they're programmable in a different way to CPUs and microcontrollers. An FPGA contains a series of logic circuits that can be programmed to connect to each other in different ways, so when programming an FPGA, you don't describe a series of steps like you would in Python or C but the circuit that you want to implement. In many ways, you can think of an FPGA as being a bit like a breadboard for prototyping a your own chips. Because of this, you can program FPGAs to perform some tasks much quicker than they could be done on a CPU (such as mine bitcoins).

One common use of FPGAs is to re-implement processors and other chips from old computers such as games consoles. It's important to remember that

the FPGA doesn't emulate the design of a chip like a games console emulator may when running on a normal CPU: it actually implements it in

“One common use of FPGAs is to re-implement processors from old computers.”

hardware. This has a massive effect on the speed at which it performs, because all the parts of the circuit are running in parallel not being simulated one at a time like they would be on a CPU.

There are a few FPGAs available, and a number of different prototyping boards for working them into your projects. Some need specialist equipment to program, while others can simply be plugged into a

USB device. In this tutorial, we're going to use a Papilio One 500K, which we bought from the Gadget Factory. In many ways, you can think of this as the Arduino of FPGAs. It's not the most powerful chip available, but it's enough to get started, and there are some easy-to-use tools to help beginners get started on it. The Gadget Factory also sells the Papilio Pro, which is more powerful than the model we're using (in FPGA terms, a device is more powerful if it has more components inside it, thereby enabling you to create larger and more complex circuits).

Gadget Factory also produces a range of wings, which are add-on hardware to enable things like audio and VGA output. Since the FPGA can be programmed to include the driver circuits for these, they're far simpler than they would be for microcontroller boards.

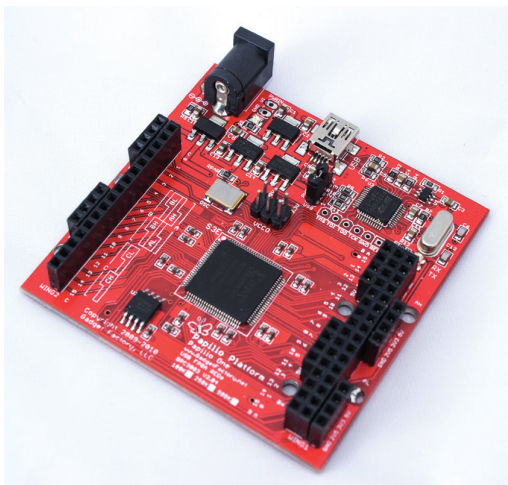
The language of hardware

The two common languages used to program FPGAs are Verilog and VHDL, but diving straight into the code isn't the easiest way to get started with the Papilio. Instead, you can start with circuits that other people have built. There aren't huge numbers of these, but there are a few. One of the most useful for Papilio users is the Zylín CPU (ZPU).

This core is also openly licensed (under a BSD-style licence). So, while running open source code on a ZPU, you're using fully open software on a fully open processing unit on a fully open board. There's enough freedom there to keep anyone's inner Stallman happy.

ZPUs aren't powerful enough to run a regular computer, but in microcontroller terms, the ZPU performs quite well. It's a 32-bit processor that runs at 96MHz. That makes it quite a bit faster than most Arduinos (about the same processing power as the Arduino Due). This means it's useful for embedded applications where you don't need a full OS stack, just a bit of processing power to control inputs and outputs. One of the most popular uses of the ZPU on the Papilio has been in making music synthesisers. The ZPU can control what's going on, while additional peripherals can generate the noises. That's what we're going to look at here.

GadgetFactory has modified the Arduino IDE to work with the ZPU and the Papilio One FPGA. We'll use that for our projects in this tutorial, so you'll need to get it from <http://forum.gadgetfactory.net/index.php?files/file/8-zap-zpuino-arduino-papilio-ide>. Installing this is simply a case of unzipping it. Inside



The Papilio FPGA board is one of the easiest ways to get started with FPGAs.

there's an executable called **zap**, which will start the IDE. If you've used the Arduino IDE before, you can use Zap in exactly the same way, but it does have a few features that are specially for the Papilio FPGA. You may have noticed that there's an extra menu called Papilio, and it's here that we'll get started.

Create a new project (known as a sketch in the Arduino terminology). Go to Papilio > New Papilio Project, and give it a name.

The programming language used is a dialect of C++. The two main functions are **setup()** and **loop()**. The **setup()** function is run when the Papilio first starts, then **loop()** runs continuously. In the new project you've just created, these will contain the code to simply turn pin 0 off and on a few times. Actually, the default code contains a typo: it uses a variable that's not declared. To correct it, change the loop function to:

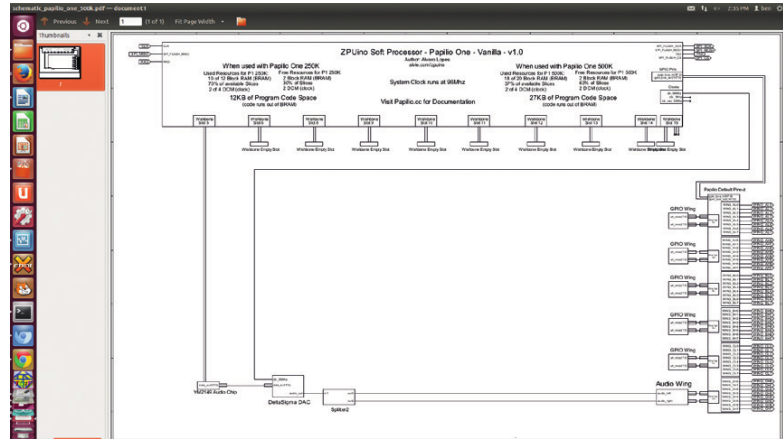
```
void loop() {
    digitalWrite(0, HIGH); // turn the LED on (HIGH is the voltage level)
    delay(1000);          // wait for a second
    digitalWrite(0, LOW); // turn the LED off by making the voltage LOW
    delay(1000);         // wait for a second
}
```

Here, in the two places where the **led** variable was used, we've just put in the value 0 instead. This simple function sets pin 0 to **high**, then waits a second (the **delay** function takes the number of milliseconds as it's argument), then puts the pin low, waits another second and repeats.

This code isn't for the FPGA, remember, but for the ZPU processor. So, before you can use this code, you have to load a BIT file containing the processor onto the board. To do this, click on the following link, which should be in the comments at the top of the page:

sketchdir://500K/papilio_one_500k.bit

This will write a ZPU processor to the FPGA. Once it's finished, it'll display the following line in the console



at the bottom of the ZAP window:

DONE = 0

Once this is finished, you can upload your code to the processor by clicking on the arrow icon in the top-left corner of the window.

There won't be any instant sign that it's worked, so you need to connect an LED to the appropriate pin along with a resistor to stop it drawing too much current. Any colour LED will do, but it needs to be connected the right way around (one of the legs will be a little shorter and there will be a flat side on the base. This is the negative leg). To stop it drawing too much current, you'll also need a resistor (any value between 220 and 1,000 ohms will be fine).

First, unplug the Papilio. You need to connect the positive leg of the LED to pin AL_0 on the Papilio One. If you look at the board with the USB port on the top side, this is the pin in the middle of the three in the bottom-right corner. The negative leg of the LED should connect to the resistor, and then the resistor should connect to the ground. You'll need a breadboard or some other circuit-building hardware to link everything together.

Once all this is connected, plug the Papilio back in and the LED should start to blink on and off.

All of the Papilio example sketches come with schematics showing the chip design inside the BIT file, so you can see what's connected to which pins.

Expanding the Papilio

There are three official 'MegaWings' made by the Gadget Factory to add features to the Papilio (all are compatible with all versions of the board).

- The LogicStart MegaWing adds a four-character seven-segment display, a VGA port, audio jack, 12-bit eight-channel SPI ADC, five direction micro joystick, eight LEDs and eight slide switches.
- The Arcade MegaWing adds a VGA port, a sound jack, PS/2 ports and two DB 9 joystick ports (for Commodore and Atari joysticks).
- The RetroCade MegaWing adds two stereo audio jacks, MIDI in, out and through, microSD card, micro joystick, 2x16 character LCD display, 16 analogue inputs and 16 digital inputs.

The first of these is designed for people who want to explore the general possibilities of the Papilio while the second and third provide functions for Arcade machines and music synths respectively. There are also a wide range of single-function wings available. Head to www.gadgetfactory.net/papilio to see all the options.

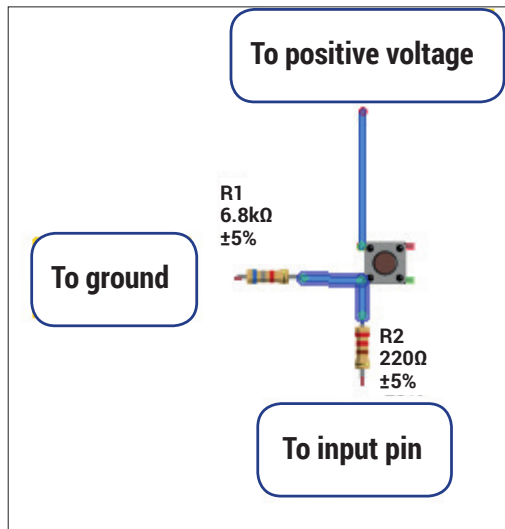
Chip tunes

The ZPU in an FPGA is a way to get a slightly more powerful Arduino, but the Papilio can be far more than just a souped-up microcontroller. The ZPU doesn't take up all of the space in the Papilio One 500K. You can use the remaining space to implement additional features that you might find useful.

Go to Papilio > Papilio Examples > Audio YM2149 Simple in the Zap menu. This will open a new sketch that includes an implementation of a YM2149 audio chip. The music geeks among you will know this as the audio chip from the Atari ST (and several other games systems). This example can be used to play music files from these old systems to generate some retro sounds.

To see the schematic for the FPGA used in this sketch, click on sketchdir://schematic_papilio_one_500k.pdf. You'll see that this contains three

Figure 1. These pulldown resistors ensure that the pin reads low when the button isn't pressed.



things in addition to the ZPU and the output pins: a YM2149, a sigma-delta DAC and an Audio splitter. The first takes the data file and converts it into audio data, the second takes the audio data and creates the signal, and the final one takes a mono audio signal and splits it so that it works with stereo headphones.

The ZPU has 10 empty wishbone slots. These are the places that additional peripherals can connect to the wishbone bus of the processor. In this example, the YM2149 is connected to wishbone slot 5. This is set up in the code with:

```
ymplayer.setup(&ym2149,5);
```

The program then reads data from the SmallFS filesystem. This is included as part of the sketch that's uploaded, and it contains the files in the **smallfs** folder in the sketch folder (these are saved in sketchbook in your home directory). If you have other YMD music files, you can put these in there, and play them, though there's not much space. ZAP does come with one that's loaded by default called **music.ymd**.

Add-on bits of Papilio hardware are known as wings, and these slot directly into the headers on the board. There are a few available; because the FPGA can be reconfigured to include any driver hardware,

Designing chips

In this tutorial we've only looked at using designs that have already been made and compiled for the Papilio. It is possible to create your own and upload these. If you want to understand what's going on at a deep level, there's a free book on the Spartan E3 (the FPGA in the Papilio) and VHDL (a hardware definition language) on GitHub at <https://github.com/hamsternz/IntroToSpartanFPGABook>.

Alternatively, The Gadget Factory has produced a schematics library. This enables you to work at a much higher level and link together components such as the sine wave generator. It is still quite technical, but easier than working with raw VHDL. There are some video tutorials on their website to help you get started: www.gadgetfactory.net/learn.

most of the wings are simple connectors to the hardware (sometimes with a high- or low-pass filter). This sketch is designed to work with an audio wing that includes a low-pass filter and an audio jack. However, this isn't essential. It is possible to connect a speaker directly between the output pins and the ground (though it's best to add a 220Ω resistor to prevent too much current being drawn).

The speaker should be connected to pin 8 on the side with the single strip of connectors (CL 0).

With this in place, upload everything to the Papilio as you did before. First, click on the appropriate BIT file link, and once this has finished, click on the icon to upload the sketch. You should hear the music playing. If it's a bit quiet, you can increase the volume by changing 11 to 15 in the following lines:

```
ym2149.V1.setVolume(15);
ym2149.V2.setVolume(15);
ym2149.V3.setVolume(15);
```

You'll need to reupload the sketch for this change to take effect.

You can't arrest me, I'm a rock star

You may have noticed that there are also examples to play Atari MOD files and Commodore SID files. However, these are too big to fit on the Papilio One. If you want to use an FPGA to emulate audio chips, the Papilio Pro is a better choice because it has more space for these more complex designs.

In the next example, we'll keep going with music, but steer away from these more complex designs. Instead, let's get back to basics with a sine wave generator. The one included with ZAP is designed for testing circuits, but it also works for generating tones.

In ZAP, go to Papilio > Papilio Examples > Bency_Waveform_Generator to open the sample project, then go to File > Save As to create a copy.

The software is set up to generate a sine wave at 2.4 MHz, which is a far higher frequency than humans can hear, so before uploading it, you'll need to change this to something a little more audible. Let's go for middle C (or 261.6Hz). Change the line in **setup()** to:

```
setFreq(0.0002616);
```

Burn the bit file to the FPGA, then upload the sketch. You should have a nice clean tone that gets a annoying after a while. To make the tone a little more interesting, we can add a slight vibrato effect by changing the sketch to the following:

```
#define MYBASE IO_SLOT(5)
#define MYREG(x) REGISTER(MYBASE,x)
```

```
float frequency= 0.0002616;
float frequencyStep = 0.0000002;
int numberOfWarbles = 10;
```

```
void setup() {
  setFreq(frequency);
}
```

```
void setFreq(float freq)
```

```

{
  unsigned long long phase = freq * 44507433.119;
  MYREG(0) = phase;
}

void loop() {
  for(int i=1;i<numberOfWarbles;i++){
    setFreq(frequency + frequencyStep*i);
    delay(10);
  }
}

```

Most of this is taken from the example sketch, with just a little added to take some values out to variables (to make them easier to change), and a for loop that changes the frequency slightly. This will vary the frequency by gradually increasing it, then dropping it back down. A more sophisticated version would also drop the frequency back down gradually, but we'll leave that up to you to implement.

This produces a slightly more interesting tone, but it still just keeps playing it on and on.

The ZPU has plenty of input/output pins that we can use, and the ZAP environment gives us a way of controlling the sine wave generator based on these inputs. We're going to use this to create a simple keyboard. To save space, we'll only use five keys (four to play notes and one to control the vibrato), but you can use the same techniques to build up to a more complex instrument. The banks of IO pins are handily arranged in blocks of eight, and five of them are completely free (three of the sixth are used up by the wave form generator, but the remaining ones could be used for control).

The code is similar to the example above, with some input pins set up, and some if statement to control the variables depending on which ones are pressed.

```

#define MYBASE IO_SLOT(5)
#define MYREG(x) REGISTER(MYBASE,x)

```

```

float frequency = 0.0;
float freqC = 0.0002616;
float freqD = 0.0002936;
float freqE = 0.0003296;
float freqF = 0.0003492;
float frequencyStep = 0.0000002;
int numberOfWarbles = 10;

```

```

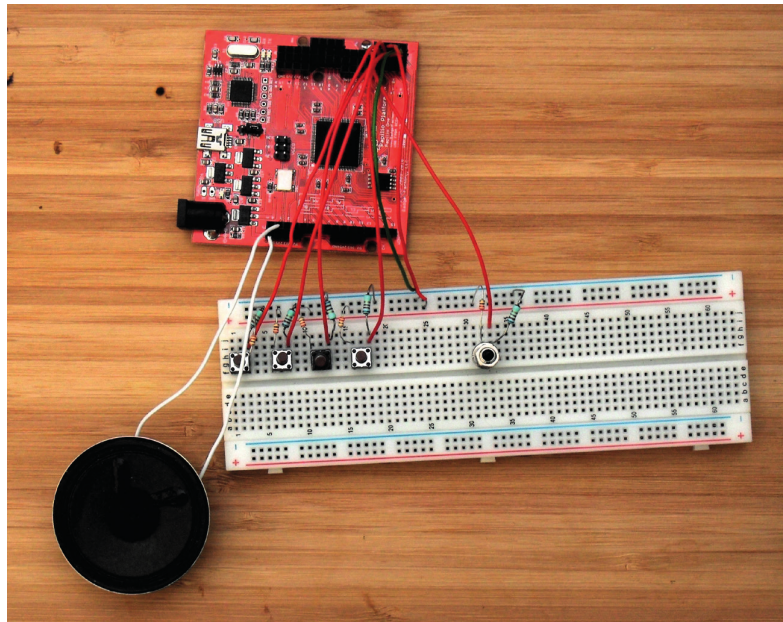
void setup() {
  // put your setup code here, to run once:
  pinMode(0, INPUT);
  pinMode(1, INPUT);
  pinMode(2, INPUT);
  pinMode(3, INPUT);
  pinMode(4, INPUT);
  setFreq(frequency); //Sets in Mhz
}

```

```

void setFreq(float freq) //sets in Mhz
{

```




```

  unsigned long long phase = freq * 44507433.119;
  MYREG(0) = phase;
}

void loop() {
  // put your main code here, to run repeatedly:
  for(int i=1;i<numberOfWarbles;i++){
    setFreq(frequency + frequencyStep*i);
    delay(10);
  }
  frequency = 0.0;
  if(digitalRead(0)==HIGH){ frequency = freqC; }
  if(digitalRead(1)==HIGH){ frequency = freqD; }
  if(digitalRead(2)==HIGH){ frequency = freqE; }
  if(digitalRead(3)==HIGH){ frequency = freqF; }
  frequencyStep = 0.0000002;
  if(digitalRead(4)==HIGH){ frequencyStep = 0.000002;}
}

```

The buttons can't be directly connected to the pins, as they need a pair of resistors in order to correctly control the voltages. See figure 1 for details of how these should be connected. The five input pins on the Papilio are in the middle row at the bottom of the pin bank on the right-hand side if you hold the Papilio with the USB port facing upwards.

We haven't even scratched the surface of what's possible with the Papilio and other FPGAs. The ability to create custom hardware in chips enables you to easily create things that are quite difficult with microcontrollers, and this is only one use for them. At the moment the library of components to go into your designs is still quite small, but it's already exciting. As they become more popular with the hobbyist community, you can expect to see a growing range of easy-to-use schematics. 

Just in case you missed it, hardware wrangler Ben has written a book about Learning Python with the Raspberry Pi. It's called *Learning Python With Raspberry Pi*.

Our little synth will probably never produce any top 10 hits, but it has a sound unique to us.