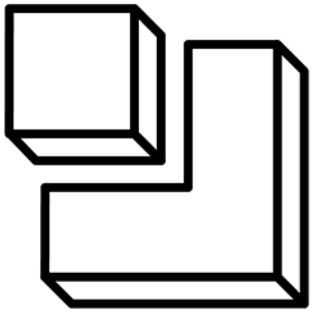
HACKADAY

Advanced Hackaday Supercon 2019 Badge Hacking

Piotr Esden-Tempski, Sophi Kravitz, Sylvain Munaut, Mike Walters and friends

Version 0.1, 13 Nov 2019



1 BIT SQUARED

Table of Contents

Overview

Course Objectives

This lab manual is a workbook that accompanies Hackaday Supercon FPGA Badge workshop.

Our objective is for attendees to get as much hands-on time with hardware as possible during our classes. Lectures serve as introductions to topics, but the walk-throughs and challenges in this manual will lead you to deeper understanding and higher retention of important information. We recommend recording as much as possible in this manual, as it may become a resource for your future hardware hacking adventures.

This training is based on the SecuringHardware.com model developed by Joe FitzPatrick. If you enjoy this wokrshop make sure to check out the list of workshops SecuringHardware.org offer. 1BitSquared also adapts and develops their own workshops for the hardware they manufacture and distribute. For example the iCEBreaker FPGA development board. You can find the WTFPga and iCEBreaker workshops on GitHub at https://github.com/icebreaker-fpga.

Chapter 1. Introduction

The Supercon 2019 Badge is centered around the Lattice Semiconductor ECP5 FPGA. As far as we know this is the very first time that a conference decided to put an FPGA on their official badge! On top of that they chose an FPGA that is supported by the Open-Source FPGA flow that runs on any OS you might choose making it a very accessible FPGA platform. If you like those ideas make sure to hack on your badge, spred the word what you did and that you like that idea. Maybe other conferences will get inspired too.

This workshop is a self-guided hands-on crash-course of how to design your own digital designs in verilog.

The objective of this workshop is to do something cool with FPGAs in only two hours. In order to introduce such a huge topic in such a short time, LOTS of details will be glossed over. Two hours from now you're likely to have more questions about FPGAs than when you started - but at least you'll know the important questions to ask if you choose to learn more.

If you have any questions or run into trouble let any of the helpers know, we are here to help. If you are working on this outside of the Supercon workshops join the hackaday.io Supercon Chat [https://hackaday.io/messages/room/280647] or the 1BitSquared discord [https://1bitsquared.com/pages/chat].

1.1. What We Won't Learn

In order to introduce Verilog and FPGAs in such a short time, we're going to skip over several things that will be important when you build your own FPGA-based designs, but are not necessary to kickstart your tinkering:

1. **IP Cores**: FPGA vendors pre-build or automatically generate code to let you easily interface your FPGA to interfaces like RAM, network, or PCIe. We'll stick to LEDs and switches today.
2. **Simulation**: Didn't work right the first time? Simulation lets you look at all the signals in your design without having to use hardware or potentially expensive observation equipment.
3. **Testbenches**: For effective simulation, you need to write even more Verilog code to stimulate the inputs to your system.

1.2. Hardware Overview

Besides the FPGA part and if you grew up in the 90' you will recognize the shape. It is one of the most iconic handheld game consoles. That is still not enough to make the board interesting though. We will need some more chips and connectors to make it an interesting device.

The badge hardware consists of the following components:

- Lattice Semiconductor ECP5 FPGA: LFE5U-45F

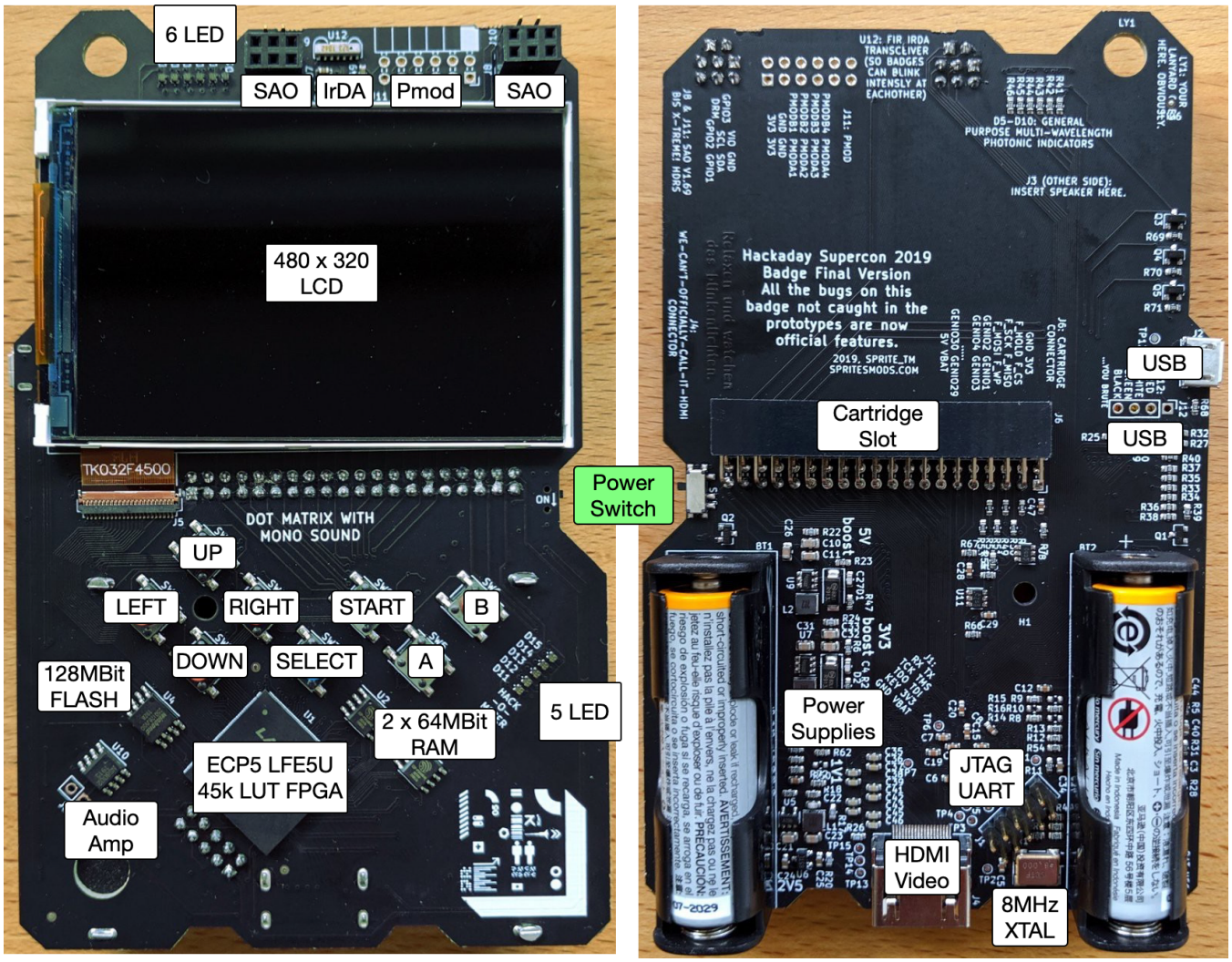
- Battery Power supply
- 128MBit (16MByte) Serial Flash
- Two 64MBit (8MByte) Serial RAM
- 320 x 480 resolution LCD
- Mono Audio amplifier
- Eight Buttons
- Eleven LEDs
- IrDA transceiver

And a bunch of connections to the outside world:

- Micro USB Connector
- HDMI Video Connector
- Cartridge Port
- Two SAO Connectors
- PMOD Connector
- Speaker Connector
- JTAG/UART Connector

In this workshop we will focus on the following parts of the badge:

- FPGA (duh)
- Buttons
- LEDs
- Pmod connector


6 LED
SAO
IrDA
Pmod
SAO
480 x 320
LCD
TK032F4500
DOT MATRIX WITH
MONO SOUND
UP
LEFT
RIGHT
START
B
DOWN
SELECT
A
128MBit
FLASH
2 x 64MBit
RAM
5 LED
ECP5 LFE5U
45k LUT FPGA
Audio
Amp
Power
Switch
Hackaday Supercon 2019
Badge Final Version
All the bugs on this
badge not caught in the
prototypes are now
official features.
2019, SPRITE_TM
SPRITESMODS.COM
Cartridge
Slot
USB
USB
Power
Supplies
JTAG
UART
HDMI
Video
8MHz
XTAL

Chapter 2. Setting Up

2.1. Get the toolchain

If you did not already you will need to install the FPGA toolchain first.

You can download precompiled toolchain from: https://github.com/xobs/ecp5-toolchain/releases

Download the version for your OS and make sure the `bin` subdirectory is in your $PATH environment variable. When you are done with that test the installation by making sure you can run `yosys`, `nextpnr-ecp5` from your command line.

For additional information about tool installations you can also refer to the badge documentation on github [https://github.com/Spritetm/hadbadge2019_fpgasoc#how-to-use].

You still might need a few additional tools installed on your system. Here is the list of the tools you might need to install too:

- git
- make

let us know if we missed any other dependencies here :D

NOTE	On Linux you might need to add a udev rule so that you get the needed permissions to access your badges USB interface. You will need to open `/etc/udev/rules.d/59-hadb19.rules` in a text editor and add the following rules in that file. When you change the rules make sure to unplug and plug in your badge again.

```
ATTR{idVendor}=="1d50", ATTR{idProduct}=="614a", MODE="660", GROUP="plugdev",
TAG+="uaccess"
ATTR{idVendor}=="1d50", ATTR{idProduct}=="614b", MODE="660", GROUP="plugdev",
TAG+="uaccess"
```

2.2. Get the workshop repository

You probably already have it on your drive as you are reading this, but for good measure here is how you obtain the workshop git repository:

```
git clone --recursive https://github.com/esden/hadbadge2019_workshops.git
cd hadbadge2019_workshops/advanced
```


Chapter 3. Your first digital design

Now that everything is set up let's dive into it and build our first digital design.

3.1. Configuring an FPGA

Now that we are familiar with the hardware as-is, let's walk through the process of synthesizing an FPGA design of your own and uploading it to the badge. We will be using the amazing open source tools called prjtrellis, nextpnr and Yosys.

1. Open the command line terminal:
    a. Enter the badge advanced workshop directory directory by typing `cd icebreaker-workshop/stopwatch`
    b. If it's not already connected, plug your badge board into your laptop with a USB cable while holding the `SELECT` (`SW7`) button. The badge should power up with a splash screen saying that you are in the DFU (Device Firmware Upgrade) mode.
    c. Run the build and upload process by executing make in the directory by typing: `make prog`
    d. Confirm that you were able to replace the default configuration with a new one by checking to see if the behavior has changed.
    e. Test the buttons and switches. Does this new configuration do anything useful?

3.2. Reading Verilog

Now that we know how to use the tools to configure our FPGA, let's start by examining some simple Verilog code. Programming languages give you different ways of storing and passing data between blocks of code. Hardware Description Languages (HDLs) allow you to write code that defines how things are connected.

1. Open `stopwatch.v` (in the repository we cloned previously) in the text editor of your choosing.
    a. Our `module` definition comes first, and defines all the inputs and outputs to our system. Can you locate them on your board?
    b. Next are `wire` definitions. Wires are used to directly connect inputs to one or more outputs.
    c. Next are parallel `assign` statements. All of these assignments are always happening, concurrently.
    d. Next are always blocks. These are blocks of statements that happen sequentially, and are triggered by the *sensitivity list* contained in the following `@( )`.
    e. Finally we can instantiate modules. There is one already instantiated and drive the 7-segment display.
2. Now let's try and map our board's functionality to the Verilog that makes it happen.

a. What happened when you pressed buttons?
b. Can you find the pushbuttons in the `module` definition? What are they called?
c. Can you find an assignment that uses each of the pushbuttons? What are they assigned to?
d. Can you follow the assignments to an output?
e. Do you notice anything interesting about the order of the `assign` statements?

You should be able to trace the `btn[1]` and `btn[3]` push button inputs, through a pair of wires, to a pair of LED outputs. Note that these aren't sequential commands. All of these things happen at once. It doesn't actually matter what order the `assign` statements occur.

3.3. Making Assignments and Combinatorial Logic

Let's start with some minor changes to our Verilog, then configure our board. We will be changing the way the four buttons (btn[0-3] aka *UP*, *DOWN*, *LEFT*, *RIGHT*) are affecting the five LED above the LCD screen (led[0-4]):

1. Change the assignments for `led[0]`..`led[4]` so that
   a. `led[0]` is on when buttons 1 and 2 are pressed.
   b. `led[1]` is on when buttons 1 and 3 are pressed.
   c. `led[2]` is on when buttons 2 and 3 are pressed.
   d. `led[3]` is on when the button 0 is pressed. For this one use the nbtn[0] that has inverted logic instead of the btn[0].
   e. `led[4]` is on when any of the four buttons is pressed.
2. Next, we need to create a new configuration for our FPGA. Brace yourself - it will be really quick! ;-)
   a. Power off your badge, hold the *SELECT* button (`SW7`), and power the badge on again. Make sure the DFU bootloader splash screen is visible.
   b. In the command line terminal that we opened earlier, type `make prog` or use the up arrow on your keyboard to recall the previous command, then press `enter`.
3. You should see some text scroll by and the new design should be uploaded and running within a few seconds. If we were using proprietary tools (Vivado or Quartus) as we did in the early versions of the WTFPGA workshop V1 and V2, that this workshop is the descendant of, the synthesis would take ~~8 minutes depending on the computer used. We used these 8 minutes to talk about the synthesis process itself. Even though we don't have to wait that long, let's talk about what the software is doing and what tools are used to accomplish those steps. It is a bit different from a software compiler.
   a. First, the software will **synthesize** the design - turn the Verilog code into basic logical blocks, optimizing it in the process using yosys. (http://www.clifford.at/yosys/)
   b. Next, the tools will **implement** the design. This takes the optimized list of registers and

assignments, and **places** them into the logical blocks available on the specific FPGA we have configured, then **routes** the connections between them using the tool called nextpnr. (https://github.com/YosysHQ/nextpnr)

c. When that completes, the fully laid out implementation needs to be packaged into a format for programming the FPGA. There are a number of options, but we will use a .bit bitstream file for programming over USB DFU interface using ecppack from the prjtrellis tool collection. (https://github.com/SymbiFlow/prjtrellis)

d. Hopefully everything will go as planned. If you have issues, look in the console for possible build errors. If you have trouble, ask for help!

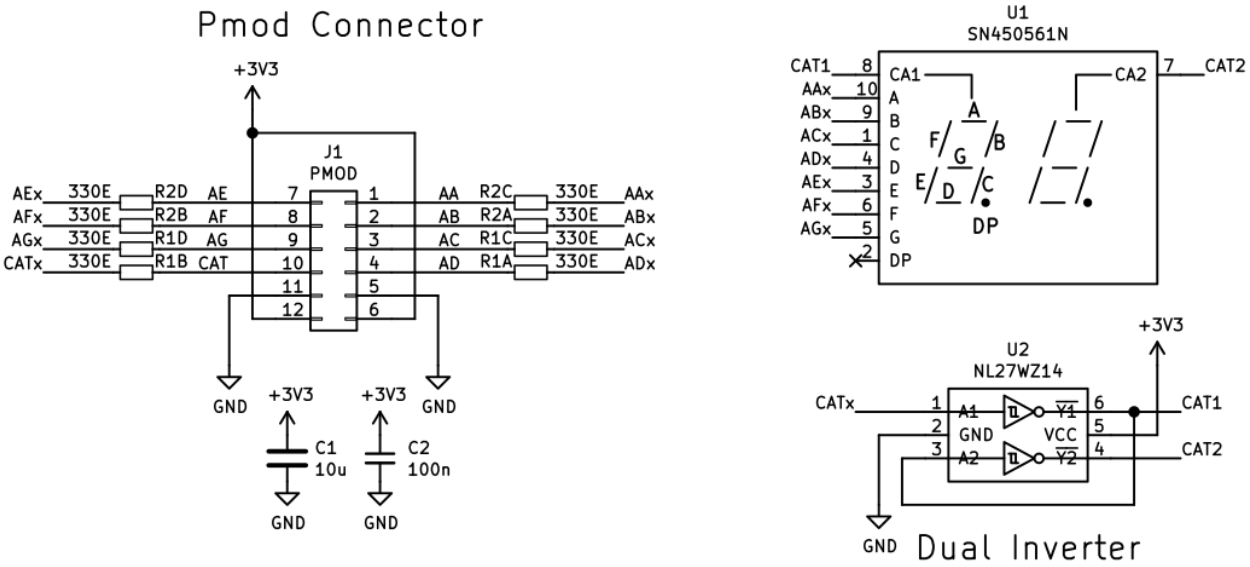
e. Finally, the .bit file needs to be sent to the FPGA over USB. When this happens, the demo configuration will be cleared and the new design will take its place using the dfu-util tool.

f. Test your system. Did it do what you expected?

3.4. Add missing 7-segment digits

If you have not done that already, this is the time to solder on the missing Pmod connector on the top edge of your badge. After you do that you should plug in the 7-segment display Pmod.

The 7-segment display Pmod module that is attached to your badge has the following schematic.

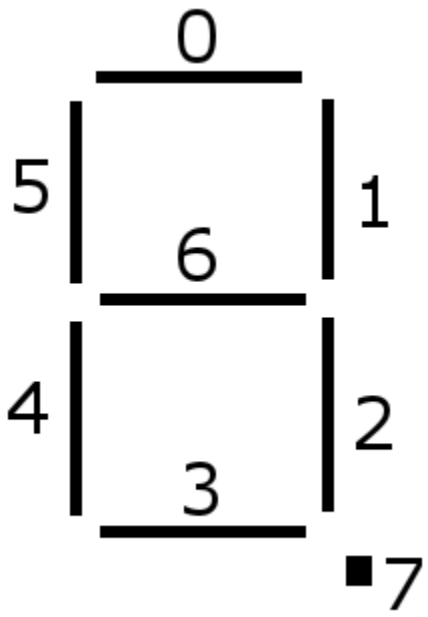


As you can see in the verilog code the pins of the badge pmod get assigned a vector of wires called `seven_segment`.

These vectors are being fed through the module `seven_seg_ctrl` that switches very quickly between the two digits to create the impression that both segments are illuminated at the same time. That module in turn uses the `seven_seg_hex` module. The module `seven_seg_hex` is converting a 4-bit binary

number into a 7-bit seven segment control vector.

The mapping of the segment vector bits to the segments looks like this:



1. For example, hex '1' looks like `7'b0000110`. We can express this as: `4'h1: dout = 7'b0000110;` which roughly translates to:

`4'`	when our 4 bits
`h1:`	equal hex 0x01
`dout =`	assign a value to segout
`7'b`	of seven bit
`0000110;`	leds 1 and 2 illuminated

   a. Figure out what you need to set for each of the hex values using the diagram.
2. We are missing the definitions for digits `3` and `8`. Go ahead and add them based on the above diagram.
3. Now `make prog` your design. Does it work?

3.5. Switch to decimal counting

The module `bcd8_increment` reads a 8-bit BCD (Binary Coded Decimal) number (a decimal digit in each nibble) and increments it by one. Replace the line assign `display_value_inc = display_value + 1;` with an instance of `bcd8_increment` so that the stop watch counts in decimal instead of hexadecimal.

(See the instances of `seven_seg_ctrl` for how to instantiate a module.)

3.6. Add RESET button

Add an if-statement to the `always @(posedge clk)` block in the top module that will reset `display_value` to zero when the `btn[0]` is pressed.

3.7. Add START/STOP buttons

While there is so much more to combinational logic than we actually touched on, let's move on to a new concept - registers. Assignments are excellent at connecting blocks together, but they're more similar to passing parameters to a function than actual variables. Registers allow you to capture and store data for repeated or later use.

Add a (1 bit) `running` register (initialized to zero), and change the code that increments `display_value` to only apply the increment when running is `1`.

Add if-statements to the `always @(posedge clk)` block in the top module that will set `running` to `1` when `btn[3]` is pressed, and reset running to `0` when `btn[1]` is pressed. Now these two buttons function as START and STOP buttons for the stop watch.

Also change the RESET functionality so that `running` is reset to `0` when the `btn[0]` is pressed.

3.8. Add lap time measurement

Finally let's also add lap time measurement: pressing the center button on the board should display the current time for two seconds while we keep counting in the background.

For this, we need to add a 8 bit register `lap_value` and an 5 bit register `lap_timeout`.

`lap_timeout` should be decremented in every `clkdiv_pulse` cycle until it reaches zero. The seven segment display should show the value of `lap_value` instead of `display_value` when `lap_timeout` has a nonzero value.

Pressing the `btn[2]` should set `lap_timeout` to `20` and copy the value from `display_value` to `lap_value`.

NOTE	The syntax `x = a ? b : c` can be used to assign value `b` to `x` when `a` is nonzero, and value `c` otherwise.

3.9. Exploring More

You've now completed a basic calculator, that uses most of the core concepts used to design nearly all silicon devices in use today. If time permits, here are a few additional things you can explore or

try with this board:

- What happens in overflow and carry situations? Can you make something different happen?
- Can you display something other than numbers?
- Examine the `had19_proto3.lpf` file. This contains all the mappings of the FPGA's pins to the names you use in your code.
- Add additional math functions to your stopwatch, like switching to displaying minutes and seconds when the seconds counter rolls over.
- Modify the design to flash the LEDs.
- Modify the design to PWM fade the LEDs.
- Add display dimming.
- Add support for the 7-segment display to the Supercon Badge SOC so you can write to it from the C code?

3.10. Getting back to the original badge SOC

If you want to revert your badge to the original state. Follow these steps:

Repower your badge while holding the SELECT (`SW7`) button, to enter the DFU mode.

```
git clone https://github.com/Spritetm/hadbadge2019_fpgasoc.git
cd hadbadge2019_fpgasoc/soc
make dfu_flash_all
```

This should bring your badge back to the same state as before the workshop.

Chapter 4. Final notes

I hope you had fun with this workshop. There is so much more to learn about the badge itself as well as FPGA development.

Make sure to take part in the badge hacking that is happening all the time during Supercon.

Shameless plug: If you want to have some more FPGA hardware to play with give the iCEBreaker [https://1bitsquared.com/products/icebreaker] a look. It is designed with FPGA beginners and developers in mind. It uses a smaller FPGA than the one on the badge but plenty for a lot of applications.

See you around! :D