

Zucker SOC

 [machdyne.com](https://machdyne.com)

 View license


★ 10 stars    2 forks

☆ Star

 Watch

<> Code


⦿ Issues

 Pull requests

▶ Actions

 Projects

 Security

 Insights

 main ▾

...



inc remove unused pll ...

last week  49

[View code](#)

# Zucker SOC

## Overview

Zucker is an experimental System-on-a-Chip (SOC) designed for Lone Dynamics FPGA computers that provides a RISC-V CPU ([PicoRV32](#)), a simple GPU, memory controllers, a keyboard controller and a UART. This repo also contains firmware, a minimal OS and example applications.

Zucker was created as a demo platform and a starting point for developing gateway and apps on the [Riegel](#) FPGA computer. The goal of Zucker is to allow FPGA computers to be used as stand-alone [timeless](#) personal computer systems when attached to a keyboard and a monitor.

## Supported Boards

- [Riegel](#)

- [Eis](#)
- [Bonbon](#)
- [Keks](#)
- [Brot](#)
- [Kolibri](#)
- [Schoko](#)

## Getting Started

---

### Building the Gateway and Firmware

Building Zucker requires [Yosys](#), [nextpnr-ice40](#), [IceStorm](#) and a [RV32I toolchain](#).

After everything is installed, to build the SOC:

```
make BOARD=<board>
```

For example:

```
make BOARD=eis
```

This will build the firmware and FPGA configuration image and write them to `output/soc.bin` . It will also build the demo apps.

### Flashing the MMOD

Once the gateway, firmware and demo apps are built, you can use [ldprog](#) to write everything to the MMOD:

```
make BOARD=<board> flash
```

For example:

```
make BOARD=eis flash
```

### Serial Console

The default configuration assumes that a UART PMOD is connected to PMODB (or PMODA if there's only one PMOD on the device).

If for example your USB-UART PMOD is on `/dev/ttyUSB0` you can access the serial console using minicom:

```
$ minicom -D /dev/ttyUSB0 -b 115200
```

Ensure that hardware flow control is *enabled*. In minicom this is under CTRL-A O, Serial port setup, Hardware Flow Control.

## Boot Process

---

0. USB Bootloader (optional)
1. Zucker Bootloader (ZBL)
2. LIX
3. Apps

### USB Bootloader

See the [Riegel](#) repo for details on setting up a USB bootloader. Eis, Keks and Bonbon are programmable over USB with the onboard RP2040, and don't need an FPGA-based USB bootloader.

### Zucker Bootloader (ZBL)

The ZBL bootloader firmware is inside the FPGA configuration image. Its primary purpose is to load the next stage (LIX) from the MMOD flash. ZBL can also perform some basic system diagnostics.

The source code for ZBL is located in [firmware.c](#) and it's called by [boot\\_picov32.S](#).

### LIX

LIX is a minimal OS and second stage bootloader. LIX is capable of loading and booting apps from a FAT-formatted SD card. LIX is programmed onto the flash MMOD after the FPGA configuration image.

The source code for LIX is located in [apps/lix](#).

### Commands

Type `help` to see a list of commands.

## User Apps

LIX loads user apps into memory at 0x40100000.

See apps/hello for an example application. You can create your own apps by making a copy of the apps/hello directory.

After compiling your app you can copy it to an SD card and run it from LIX:

```
lix> run myapps/hello.bin
```

You can also upload apps to LIX over the UART using the [xfer](#) utility.

If a file named BOOT.BIN is located in the root directory of the SD card it will be loaded into main memory and run automatically at boot time.

## Lisp

LIX also includes a simple Lisp interpreter.

Some usage examples:

```
lix> (+ 1 2 3 4 5)
[num:15.000000]
lix> (map (lambda (x) (* 2 x)) (list 1 2 3 4))
[list: [num:2.000000] [num:4.000000] [num:6.000000] [num:8.000000] ]
```

Display the environment:

```
lix> (dump)
```

Load a LISP program from the SD card:

```
lix> (load myfiles/lisp/hello.l)
```

## Technical Details

---

### PMODs

The default configuration assumes that a USB-UART PMOD is connected to PMODB (or PMODA if the device only has one PMOD).

## Memory Map

Begin	End	Size	Description
00000000	000017ff	5120 - 6144	BRAM (ZBL firmware)
10000000	100007cf	2000	BRAM (video text memory)
20000000	2fffffff	38.4KB - 512KB	SRAM/SPRAM/BSRAM (framebuffer)
40000000	4fffffff	8MB - 64MB	HRAM/QQSPI/SDRAM (main memory)
80000000	8fffffff	-	MMOD (read-only flash memory)
a0000000	affffffff	-	Cartridge memory (Keks PMODA)
e0000000	effffffff	-	RPMEM
f0000000	f0000000	1	UART0 data register
f0000004	f0000004	1	UART0 control register
f0001000	f0001000	1	LED control register
f0001100	f0001103	4	RTC seconds counter register
f0002000	f0002000	1	SD card SPI register
f0003000	f0003000	1	PS/2 data register
f0003004	f0003004	1	PS/2 control register
f0004000	f0004000	1	UART1 data register
f0004004	f0004004	1	UART1 control register
f0005000	f0005000	1	left gamepad
f0005004	f0005004	1	right gamepad

## Video Graphics

☰ README.md

whether or not pixel doubling is enabled.

Each byte in the framebuffer contains two pixels: XRGBXRGB. X is an unused bit.

The remaining SRAM is currently unused and available.

## Video Text

There is a 80x25 character buffer in BRAM at 0x10000000.

Writing an ASCII character to this video text memory will display it on the screen.

Text and graphics can be displayed at the same time.

## License

Zucker is released under the Lone Dynamics Open License. This repo contains code from the [PicoRV32](#) project, you can find its license in the [rtl/cpu/picorv32](#) directory.

## Releases

No releases published

## Packages

No packages published

## Contributors 3



inc Philip



cathiele Carsten Thiele



machdyne Machdyne

## Languages

● C 64.2%   ● Verilog 32.8%   ● Makefile 1.8%   ● Common Lisp 0.6%   ● SystemVerilog 0.3%  
● Assembly 0.2%   ● Python 0.1%