Caravel Frame and SoC

Version 6.1.0

Caravel Frame and SoC documentation
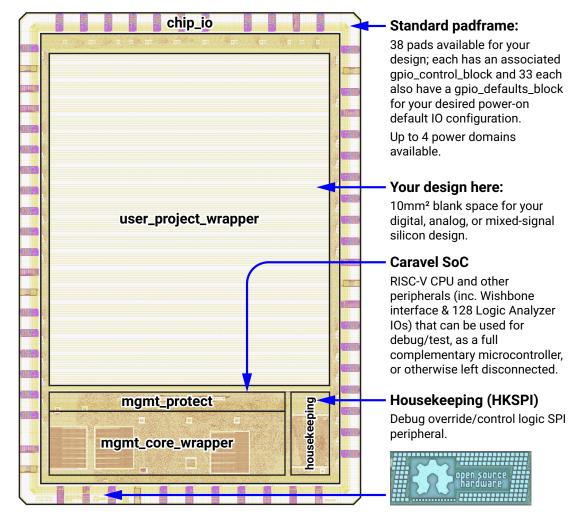
Table of Contents

GPIO (General Purpose I/O)

Logic Analyzer   59

UART (RS232 Serial Interface)   60

Wishbone Interface   61

SPI Controller   14

Counter/Timer   14

Housekeeping and HKSPI   64

Interrupts (IRQs)   74

Registers and Memory Map   75

Analog Connections   78

Advanced Guides   80

Caravel Overview

Caravel die floorplan

Standard padframe:

38 pads available for your design; each has an associated gpio_control_block and 33 each also have a gpio_defaults_block for your desired power-on default IO configuration.

Up to 4 power domains available.

Your design here:

10mm² blank space for your digital, analog, or mixed-signal silicon design.

Caravel SoC

RISC-V CPU and other peripherals (inc. Wishbone interface & 128 Logic Analyzer IOs) that can be used for debug/test, as a full complementary microcontroller, or otherwise left disconnected.

Housekeeping (HKSPI)

Debug override/control logic SPI peripheral.

Designing and fabricating an ASIC with Efabless chipIgnite ⬈ requires that you use an existing supported template (or "frame"), and our free and open-source **Caravel** is recommended as the most popular, feature-rich, and production-ready template. The chipIgnite submission process automatically integrates your user project design area into the frame, so you don't have to worry about it.

With **Caravel** , you have a ready-to-use chip harness for creating your own ASIC design and getting it fabricated for prototype or production purposes. It includes a **standardized padring** , blank silicon **design area of 10mm** ^{2} and optional **on-chip SoC** (microcontroller/management/test framework). You can use it whether you are creating a proprietary/private chip, one for commercial purposes, or an open-source design.

> **ⓘ Hint**
>
> While this documentation describes **Caravel** , it also covers **Caravan** (our frame with some dedicated bare analog pads, suitable for prototyping) and **Caravel Mini** . All 3 are very similar, with some specific distinctions marked where appropriate.
>
> For more information about the different frames and options, see: Which chipIgnite template should I use? ↗

How do I use this guide?

It is intended that you can work through each of the following sections in order to learn the basics of Caravel and how to use it, before reaching more advanced topics.

This guide covers general use of Caravel as a chip padring/harness for essential support of your design, as well as programming the Caravel SoC as a microcontroller and interfacing it with your design, and more-advanced concepts like analog connections.

Features of Caravel

As a padframe, Caravel offers easy-to-connect standard features such as GPIO pins, power delivery (optionally with multple power domains), clocking, and reset logic (both a dedicated reset pin, and a POR circuit). These are well-characterized and qualified to reduce the overhead of the designer.

As an SoC , Caravel provides a wide range of optional functionality on-chip that can either be largely ignored or can be used for any of bringing up your design, debugging and providing a diagnostic/maintenance interface, controlling different peripherals that your design can leverage, or even as a full microcontroller (including CPU with external firmware interface, basic UART, and SPI controller).

Caravel Padframe General Features

These features are universally available to any chip based on the Caravel frame, even without specific use of the **Management SoC** .

Caravel user_project_wrapper

Caravel includes a **10mm² "user project wrapper" design area** . The `user_project_wrapper` is what a user submits as a fixed-area GDS macro. It is then automatically integrated (by the Efabless chipIgnite submission process) into the rest of the Caravel chip harness to produce the final GDS that is submitted for fabrication. This design area, in the 130nm node, is enough for on the order of 6 million transistors or 1 million logic gate primitives .

The designer can choose whether the design they include in the user project wrapper will interface with the Caravel SoC, the padframe GPIOs, the built-in clock sources, the PDN (Power Delivery Network) or any combination of these. For advanced users ordering bare dice and implementing a highly-specialized design, the user design may even use none of these features (though this is not recommended).

The user project wrapper provides **support for digital, analog, and mixed-signal projects** .

See: user_project_wrapper

Housekeeping SPI

The chip's power-on state defaults to assigning 4 GPIOs as a **Housekeeping SPI (HKSPI)** interface for an external SPI controller to assert debugging control over certain base configuration, debugging, and clocking of the chip.

This also includes **SPI pass-through** , able to be driven via HKSPI to take control (for reading/writing) of a firmware SPI ROM connected to the Management SoC.

See: Housekeeping and HKSPI

38 GPIOs

Caravel provides **38 GPIO pins** that can be used interchangeably by the user's own digital logic, the Caravel CPU, or in some cases as analog connections:

- These 38 can be configured (and reconfigured via CPU firmware or HKSPI) to function as outputs, bidirectional, or inputs (including optional pull-up or pull-down).

- They also have built-in ESD protection, level shifting, and buffering, thus simplifying the job of the designer.

- 33 support power-on default configuration specified in silicon by the designer, while the remaining 5 have Caravel-dedicated power-on functions (that can be overridden by CPU firmware or HKSPI).

- 29 optionally support direct pad connections for analog signals [1].

- **For Caravan** : 11 are "bare analog" pads without GPIO circuitry and without ESD protection.

- Some offer additional Caravel SoC "management" functions (such as UART and additional debug functions).

See: General Purpose Input/Output (GPIO)

Footnotes

[1]
Caravel direct analog pad connections include ESD protection which typically limits full swing signals to about 50MHz.

Dedicated clock input and DLL/DCO configurable clocking

Caravel and/or the user design can optionally receive (and modify) a clock signal via a dedicated clock input pin that includes circuitry for multiplying/dividing the input clock frequency.

> **ⓘ Note**
>
> The DLL/DCO is inactive by default, passing the optional dedicated clock input directly through to the user project wrapper. If the DLL/DCO is to be used, it must be explicitly enabled via

HKSPI or firmware running on the **Management SoC** . For more information, see the section on **Clocking, DLL and DCO** .

See: Clocking and DLL/DCO

POR (Power-On Reset) module

See: Power-up/boot process

Four power domains

Caravel provides **4 power domains** : two intended as nominal 1.8V digital supplies, two intended as analog supplies in the range 1.8V to 5.5V. It also includes vddio for setting the desired external digital logic level (as a reference voltage in the range 1.8V-5.5V) for compatibility with a wide range of device.

Caravel Management SoC Features

The Management SoC's RISC-V CPU (RV32I) is built into the die area, adjacent the user project wrapper, and can be interfaced with your design, as well as externally to the chip.

The SoC is generated using Litex and includes the following peripherals and capabilities that can be optionally enabled/disabled on subsets of the GPIO pins, to make the SoC useful both as a general-purpose microcontroller and a specialized test/debug interface for your design…

VexRiscv RISC-V CPU core

The CPU core is a VexRiscv minimal+debug configuration. Intended for use either as a microcontroller, general-purpose CPU, control or debug interface to the user design. It can be considered as a lightweight single-core bare-metal microcontroller, programmable in C or RISC-V assembly (RV32I , 32-bit instructions specifically), and it comprises:

- **Dedicated firmware ROM SPI master** for XIP loading of firmware code from an external SPI memory into a local 16-word (64-byte) instruction cache.

- **1.5kByte** local SRAM for stack, scratch space/variables, or small high-speed in-memory executable subroutines.

- Interrupt and exception handling .

- **Dedicated power domain** .

- **GPIO control** : Ability to reconfigure the 38 GPIOs (**27 for Caravan**), including taking over GPIOs as "management mode", either to activate SoC peripherals that have external interfaces, or for firmware to directly use some GPIOs.

- **Single management GPIO pin** . See: Management GPIO pin

See: Firmware, Flash SPI, and Programming Guide

The CPU core also has **access to a range of other SoC peripherals** as described below.

> **ℹ Note**
>
> If you don't intend to make use of the Management SoC at all, you can simply choose to not connect to its ports in the users project wrapper, and you can optionally tie its `RESETb` signal low externally to hold it in reset.

Logic Analyzer interface

The **Logic Analyzer** comprises 128 internal IO pins that can optionally be connected with your design in the user project wrapper.

See: Logic Analyzer

Wishbone master

The user project wrapper has an incoming **Wishbone master** interface from the CPU. This includes the wb_clk_i signal. It is implemented as a 32-bit Classic-Wishbone-based memory map expansion of the CPU for addresses in the range `0x30000000` – `0x300FFFFF` .

See: Wishbone Interface

UART

The SoC includes a UART that is accessible only to the CPU. It can be enabled to take over two specific GPIO pins for transmit and/or receive, and it has the following features:

· Fixed baud rate proportional to 9,600 baud at a 10MHz core clock (i.e. 19,200 baud if the Caravel core clock is set to 20MHz).

· Fixed 8N1 configuration.

· 16-byte FIFO for each of transmit and receive.

· TX/RX runs independently of the CPU.

· CPU can poll the FIFO state or enable an IRQ for the UART.

See: UART (RS232 Serial Interface)

SPI Controller

SPI master for direct control by user firmware.

See: SPI Controller

6 user IRQs
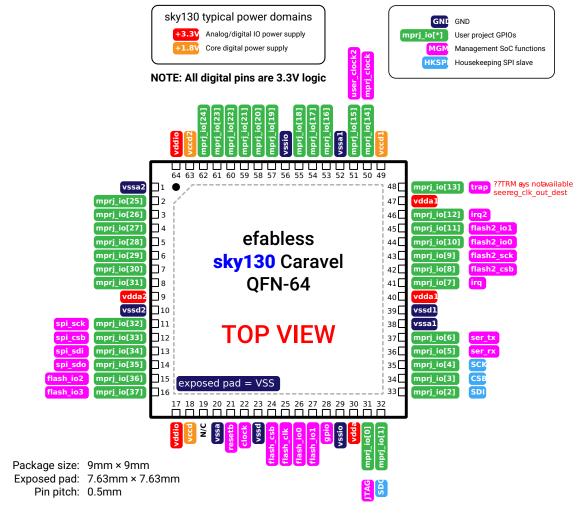
· 3 internally-driven by the user project.

· 2 externally-driven (optional) by GPIO pins.

· 1 internally-driven by SoC peripherals.

See: Interrupts (IRQs)

Counter-timer

See: Counter/Timer

Caravel Pinouts

Caravel QFN64 pinout

All chips fabricated using the Caravel harness have a standard pinout with some pins dedicated to the SoC (i.e. the CPU), some for power, most for user-configurable GPIOs, and a few with shared functions.

Caravel chips can be ordered in a 64-pin QFN package, or as bare dice (unpackaged, bare silicon chips).

Older generations of chipIgnite and the Open MPW shuttles also supplied WLCSP packaged parts. These are no longer available with the standard chipIgnite prototyping service.

Caravel pins and functions

Caravel QFN-64 pin descriptions

QFN64 #	User function	User type	User description	Mgmt function	Mgmt type	Mgmt description
1				`vssa2`	Ground	**User Project Wrapper** analog ground 2
2	`mprj_io[25]`	Digital I/O or Analog	GPIO[**25**] [1] or analog_io[**18**] [4]			
3	`mprj_io[26]`	Digital I/O or Analog	GPIO[**26**] [1] or analog_io[**19**] [4]			
4	`mprj_io[27]`	Digital I/O or Analog	GPIO[**27**] [1] or analog_io[**20**] [4]			
5	`mprj_io[28]`	Digital I/O or Analog	GPIO[**28**] [1] or analog_io[**21**] [4]			
6	`mprj_io[29]`	Digital I/O or Analog	GPIO[**29**] [1] or analog_io[**22**] [4]			

QFN64 #	User function	User type	User description	Mgmt function	Mgmt type	Mgmt description
7	`mprj_io[30]`	Digital I/O or Analog	`GPIO[ 30 ]` [1] or `analog_io[ 23 ]` [4]			[5]
8	`mprj_io[31]`	Digital I/O or Analog	`GPIO[ 31 ]` [1] or `analog_io[ 24 ]` [4]			[5]
9				`vdda2`	3.3V Power	**User Project Wrapper** analog power supply 2
10				`vssd2`	Ground	**User Project Wrapper** digital ground 2
11	`mprj_io[32]`	Digital I/O or Analog	`GPIO[ 32 ]` [2] or `analog_io[ 25 ]` [4]	`spi_sck`	Digital out	**SPI Controller** clock
12	`mprj_io[33]`	Digital I/O or Analog	`GPIO[ 33 ]` [2] or `analog_io[ 26 ]` [4]	`spi_csb`	Digital out	**SPI Controller** chip select (**active-low**)
13	`mprj_io[34]`	Digital I/O or Analog	`GPIO[ 34 ]` [2] or `analog_io[ 27 ]` [4]	`spi_sdi`	Digital in	**SPI Controller** data input
14	`mprj_io[35]`	Digital I/O or Analog	`GPIO[ 35 ]` [2] or `analog_io[ 28 ]` [4]	`spi_sdo`	Digital out	**SPI Controller** data output

QFN64 #	User function	User type	User description	Mgmt function	Mgmt type	Mgmt description
15	`mprj_io[36]`	Digital I/O	GPIO[**36**] [2]	`flash_io2`	Digital I/O	[6]
16	`mprj_io[37]`	Digital I/O	GPIO[**37**] [2]	`flash_io3`	Digital I/O	[6]
17				`vddio`	3.3V Power	ESD and padframe power supply [7]
18				`vccd`	1.8V Power	Management SoC digital power supply
19	N/C	–	No connection	N/C	–	No connection
20				`vssa`	Ground	Management SoC analog ground
21				`resetb`	Digital in	Management SoC system reset (active-low)
22				`clock`	Digital in	External CMOS 3.3V clock source
23				`vssd`	Ground	Management SoC digital ground
24				`flash_csb`		

QFN64 #	User function	User type	User description	Mgmt function	Mgmt type	Mgmt description
					Digital out	Firmware Flash SPI chip select (active-low)
25				flash_clk	Digital out	Firmware Flash SPI clock
26				flash_io[0]	Digital out	Firmware Flash SPI serial data out
27				flash_io[1]	Digital in	Firmware Flash SPI serial data in
28				gpio	Digital I/O	Management GPIO pin / user power enable
29				vssio	Ground	ESD and padframe ground [7]
30				vdda	3.3V Power	Management SoC analog power supply
31	mprj_io[0]	Digital I/O	GPIO[0] [3]	debug	Digital I/O	CPU debug port
32	mprj_io[1]	Digital I/O	GPIO[1] [3]	SDO	Digital out	HKSPI data output

QFN64 #	User function	User type	User description	Mgmt function	Mgmt type	Mgmt description
33	`mprj_io[2]`	Digital I/O	GPIO[**2**] [3]	`SDI`	Digital in	HKSPI data input
34	`mprj_io[3]`	Digital I/O	GPIO[**3**] [3]	`CSB`	Digital in	HKSPI chip select (active-low)
35	`mprj_io[4]`	Digital I/O	GPIO[**4**] [3]	`SCK`	Digital in	HKSPI clock
36	`mprj_io[5]`	Digital I/O	GPIO[**5**] [3]	`ser_rx`	Digital in	UART receive channel
37	`mprj_io[6]`	Digital I/O	GPIO[**6**] [3]	`ser_tx`	Digital out	UART transmit channel
38				`vssa1`	Ground	User Project Wrapper analog ground 1
39				`vssd1`	Ground	User Project Wrapper digital ground 1
40				`vdda1`	3.3V Power	User Project Wrapper analog power supply 1
41	`mprj_io[7]`	Digital I/O or Analog	GPIO[**7**] [2] or analog_io[**0**] [4]	`irq`	Digital in	External interrupt request
42	`mprj_io[8]`		GPIO[**8**] [2] or analog_io[**1**] [4]	`flash2_csb`		

QFN64 #	User function	User type	User description	Mgmt function	Mgmt type	Mgmt description
		Digital I/O or Analog			Digital out	User SPI pass-thru enable (active-low)
43	`mprj_io[9]`	Digital I/O or Analog	GPIO[9] [2] or analog_io[2] [4]	`flash2_sck`	Digital out	User SPI pass-thru clock
44	`mprj_io[10]`	Digital I/O or Analog	GPIO[10] [2] or analog_io[3] [4]	`flash2_io[0]`	DO ??	User SPI pass-thru data out
45	`mprj_io[11]`	Digital I/O or Analog	GPIO[11] [2] or analog_io[4] [4]	`flash2_io[1]`	DI ??	User SPI pass-thru data in
46	`mprj_io[12]`	Digital I/O or Analog	GPIO[12] [2] or analog_io[5] [4]	`irq2`	DI ??	External interrupt request IRQ
47				`vdda1`	3.3V Power	User Project Wrapper analog power supply 1
48	`mprj_io[13]`	Digital I/O or Analog	GPIO[13] [2] or analog_io[6] [4]	`trap`	Digital ??	[8]
49				`vccd1`	1.8V Power	User Project Wrapper digital power supply 1

QFN64 #	User function	User type	User description	Mgmt function	Mgmt type	Mgmt description
50	mprj_io[14]	Digital I/O or Analog	GPIO[14] [2] or analog_io[7] [4]	mprj_clock	Digital out	Clock monitoring output for wb_clk_i
51	mprj_io[15]	Digital I/O or Analog	GPIO[15] [2] or analog_io[8] [4]	mprj_clock2	Digital out	Clock monitoring output for user_clock2
52				vssa1	Ground	User Project Wrapper analog ground 1
53	mprj_io[16]	Digital I/O or Analog	GPIO[16] [1] or analog_io[9] [4]			
54	mprj_io[17]	Digital I/O or Analog	GPIO[17] [1] or analog_io[10] [4]			
55	mprj_io[18]	Digital I/O or Analog	GPIO[18] [1] or analog_io[11] [4]			
56				vssio	Ground	ESD and padframe ground [7]
57	mprj_io[19]	Digital I/O or Analog	GPIO[19] [1] or analog_io[12] [4]			
58	mprj_io[20]					

QFN64 #	User function	User type	User description	Mgmt function	Mgmt type	Mgmt description
		Digital I/O or Analog	GPIO[20] [1] or analog_io[13] [4]			
59	mprj_io[21]	Digital I/O or Analog	GPIO[21] [1] or analog_io[14] [4]			
60	mprj_io[22]	Digital I/O or Analog	GPIO[22] [1] or analog_io[15] [4]			
61	mprj_io[23]	Digital I/O or Analog	GPIO[23] [1] or analog_io[16] [4]			
62	mprj_io[24]	Digital I/O or Analog	GPIO[24] [1] or analog_io[17]			
63				vccd2	1.8V Power	User Project Wrapper digital power supply 2
64				vddio	3.3V Power	ESD and padframe power supply [7]

[1] (1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10 , 11 , 12 , 13 , 14 , 15 , 16 , 17 , 18)

GPIOs are General purpose configurable digital I/O with pullup/pulldown, input/output/bidirectional, enable/disable, and slew rate control. GPIO pins are shared between the user project area and the management SoC: any configured in "USER" mode are directly connected/controlled via logic

in the user_project_wrapper ; any configured in "MGMT" mode are directly under control of the Management SoC, *plus* their respective "Mgmt function" (if any) can optionally also be enabled. The power-on mode configuration of most GPIO pins is mask-programmed, defined by user_defines configuration during tapeout. **NOTE** : Some GPIOs can be configured for direct "analog" connections [4]

[2] (1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10 , 11 , 12 , 13 , 14 , 15 , 16 , 17)
GPIO function same as above [1], also optionally supporting activation of an alternate management function (i.e. a specific peripheral device in the management SoC that can optionally be activated when the pin is in "MGMT" mode).

[3] (1 , 2 , 3 , 4 , 5 , 6 , 7)
GPIO function same as above [2], but always powers up initially in "MGMT" mode , *and* with its respective management function activated by default (to ensure Housekeeping can always be made available). Unlike [1] and [2], these pins cannot be overridden by user_defines configuration but can still be temporarily overridden by firmware or HKSPI .

[4] (1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10 , 11 , 12 , 13 , 14 , 15 , 16 , 17 , 18 , 19 , 20 , 21 , 22 , 23 , 24 , 25 , 26 , 27 , 28 , 29 , 30)
"mrpj_io" pins, where specified, can also be configured for "analog_io" mode. In this case, digital buffers are disabled, thus enabling custom analog circuits in the User Project Wrapper to make direct connections to the respective bare pad. See: Analog Connections ; and note the internal " `analog_io[#]` " numbering differences.

[5] (1 , 2)
Caravan can only provide its two clock monitoring pins via mprj_io 30 and 31 (instead of 14 and 15) as Caravan repurposes `mprj_io[24:14]` as bare analog pads only, with no built-in digital configuration options. This alternate function mapping to 30/31 only exists on Caravan, not on Caravel.

[6] (1 , 2)
For more information on QSPI and the two additional flash IO pins, see: https://github.com/efabless/caravel/blob/27cbe49c90ba5362ad52c9968dd98e035c30c74f/verilog/rtl/housekeeping.v#L776-L793 ↗

[7] (1 , 2 , 3 , 4)
`vddio` sets the digital I/O 'high' voltage level, automatically handling level shifting. `vddio` (supply) and `vssio` (ground) are also connected to pad clamping diodes for ESD protection. `vddio` is nominally 3.3V; see also: Specifications and Ratings .

[8]

Caravel Registers TRM says this is not available; See: `reg_clk_out_dest`

Caravel bare die pinout

Caravel bare dice have bond pads in a standard padring and are numbered starting at 1 on the top of the left-hand edge, incrementing counter-clockwise up to pad 63.

Caravel WLCSP pinout

Wafer-level chip-scale packaging is no longer offered by Efabless for standard chipIgnite orders, but may be specially-ordered and customized for large-volume production orders.

Older generations of Caravel chip already fabricated as WLCSP instead of QFN or bare dice had the following pinout:

Getting Started with Caravel

Essentials

Should this come before or after 'Minimal example'?

Can describe the essentials of Caravel itself? Is this already covered sufficiently in "Features of Caravel"?

Can include high-level steps, e.g. create your digital logic, connect it to physical harness, harden it to a silicon layout, ensure timing is met for realiable operation and expected clock speed(s), create firmware (if applicable), simulate (RTL & GL), submit for precheck, submit for final tapeout integration, submit for fabrication.

Different approach is to list the things a user would want to accomplish, e.g. you have a need for inputs and outputs that comply with what Caravel's physical harness offers, and optionally want to leverage SoC features, allow CPU to control/supervise aspects of your design, or just be used for bring-up and debug/diagnostics.

Minimal example

Note that all of this (except user_defines) is already implemented for a real example in CUP. Suggest the user try the example (in full) first. Almost out of scope: Should be covered better by CUP doco.

Implementation: Design/functionality:

- Explain Verilog is the most common and readily-accessible way to think about this and get started.

- Describe a digital macro with inputs and outputs and optionally a clock source. Note that other hardening methods exist besides 1 macro.

- Connect it to UPW pins. Avoid lower 5 for simple designs. Point out that other pins may coexist with other SoC features, but all are optional. Note power connections inside the 'guard' also.

- Specify user_defines for connected pins.

Hardening:

- Describe the macro's physical attributes in config.json.

- Harden the macro – ensure timing is met (advanced users can adjust as required – OpenLane is out of scope, but what about SDC port naming??)

- Minimal instantiation of the macro in UPW config.json.

- Harden UPW – check timing again.

- Check for any other warnings.

Firmware, simulation/verification: …

Submission: …

Block diagram

user_project_wrapper

caravel_user_project template

Power-up/boot process

Clocking and DLL/DCO

Caravel has a dedicated clock input pin. While it is not mandatory to make use of this for any design submitted to chipIgnite, an external clock source is (in most cases) required in order for most parts of the Caravel SoC to be used.
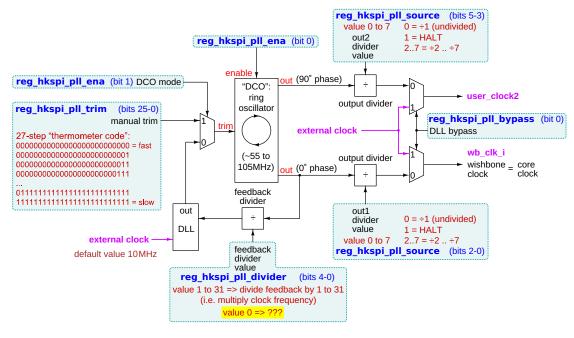
The Caravel SoC's RISC-V CPU derives its core clock from the clock pin, as do the following peripherals that can be used by the CPU:

· UART

· SPI Controller

· Counter/Timer

Note that HKSPI can be used without using the Caravel clock pin, as it has its own dedicated SCK clock input.

At power-on (and during any reset state) the clock signal present on the clock pin is passed, unmodified, directly to the CPU and other dependent peripherals, and is also available as an incoming signal in the user project area via both wb_clk_i and user_clock2.

The CPU or HKSPI may modify this clock path, e.g. to enable the DLL (and hence multiply and divide it to derive a different clock frequency), or to switch to using Caravel's internal ring oscillator (DCO) as the clock source:

Caravel clocking diagram

While the Caravel SoC uses specifically wb_clk_i , if *your own user project* requires a clock source then you might choose to use any of:

· wb_clk_i – recommended for synchronous interfacing with the Caravel CPU (e.g. via Wishbone Interface or Logic Analyzer).

· user_clock2

· Any GPIO input pin

· A Caravel-controlled Logic Analyzer pin

· Your own internal oscillator, e.g. a ring oscillator

Of the options given, wb_clk_i and user_clock2 are preferred because they have known timing characteristics and can either be derived from Caravel's internal ring oscillator (DCO) or from Caravel's external dedicated clock input pin – in both cases optionally also being modified by the DLL/divider circuits as described. In turn this means that they can be controlled, to a degree, by firmware running on the CPU.

DLL (Delay-Locked Loop)

The Caravel DLL is like a PLL as found in an FPGA. The Caravel frame has a dedicated " clock " input pin.

DCO (Digitally-Controlled Oscillator)

This is an internal ring oscillator with a fixed base frequency and which can be "trimmed" by up to 26 steps to control its actual output frequency. It is used by the DLL to generate a reasonably stable multiple of the input clock source, but can also be used simply as a direct clock source instead of the clock input pin. In this mode it can optionally be divided by two independent integer dividers (to produce each of wb_clk_i and user_clock2).

Note that the actual internal DCO frequency is PVT -dependent.

wb_clk_i

`wb_clk_i` is the core clock used by the Caravel CPU and related peripherals. It is also available inside the User Project Wrapper .

user_clock2

When the DLL is enabled, the clock source feeding `user_clock2` 's divider is 90 degrees out of phase. That is, `user_clock2` lags wb_clk_i by a quarter-cycle.

Clock monitoring

It's possible to monitor wb_clk_i and/or user_clock2 via GPIO pins, which can help with debugging your clock source and DLL/DCO behavior.

Caravel can use GPIOs 15 and 14 for this purpose, while Caravan (for chips fabricated as of June 2023) can use GPIOs 31 and 30 instead.

Enabling this feature requires setting the respective GPIO(s) to GPIO_MODE_MGMT_STD_OUTPUT mode, and then setting one or two bits of the clock monitor register: HKSPI register 0x1B (in firmware, this is `reg_clk_out_dest` or address `0x26200004`).

- For Caravel: Monitoring of wb_clk_i and user_clock2 is via GPIOs 15 and 14 respectively. These are respectively enabled by writing `1` to bits 2 and/or 1 of the clock monitor register.

- For Caravan: Monitoring of wb_clk_i and user_clock2 is via GPIOs 31 and 30 respectively. These are respectively enabled by writing `1` to bits 4 and/or 3 of the clock monitor register.

Firmware, Flash SPI, and Programming Guide

Use of the Caravel CPU and other SoC peripherals is optional. Complex user designs can be made that ignore the CPU and SoC completely, but they offer several other advantages also. You might want to use the CPU and Management SoC for any of:

· Bring-up and debugging, especially in a prototype where you need to be able to define and inspect internal signals, or otherwise alter aspects of how your design interfaces with the rest of the system and the outside world.

· As a general-purpose microcontroller, whether in combination with your design, or otherwise separately.

· As a supervisor to ensure proper operation of your design, especially to enable user project access to GPIO[**4:0**] (i.e. GPIOs reserved for HKSPI and debugging) if specifically required.

· Diagnostics and maintenance in the field.

If you want the CPU to remain idle/unused, you should strap the resetb pin low (i.e. ensure the CPU is always held in reset).

Otherwise, use of the CPU (or SoC typically) requires that the CPU is able to execute firmware code. This section covers:

· Hardware required to host firmware code that the CPU will execute.

· Writing C code that is compatible with the RISC-V CPU in general.

· References to specific C API calls that access hardware and functions of the SoC.

· Installing and using a compiler that targets the Caravel RISC-V CPU.

· Simulating execution of your C code on the CPU, and its interaction with the rest of the chip and with your user design.

For details on how to actually deploy firmware code to a Caravel-based development board (with your fabricated chip) and test it, see: Caravel Chip Bring-up and the Caravel Evaluation Board .

Firmware Flash SPI interface

Writing and compiling basic firmware for the Caravel RISC-V CPU

Writing and simulating testbenches with Verilog or Cocotb

Other important programming information

Caravel Full-chip Simulation

General Purpose Input/Output (GPIO)

In the Caravel context, "General Purpose Input/Output" (or GPIO) most often refers to the 38 User GPIO pins , but might also refer to the single extra Management GPIO pin .

> ❗ **See also**
>
> **Caravel Pinouts**
>
> List of all the Caravel chip external pins (including GPIO pins), named by function.

User GPIO pins

Caravel provides 38 external GPIO pins, typically called `mprj_io[37]` down to `mprj_io[0]` , sometimes referred to as `GPIO[37:0]` or simply `IO[37:0]` . These are available for use by the user design and logic, and/or by the Caravel Management SoC (including CPU, but also other dedicated periphrals that can be enabled on certain pins). While all are digital-capable, some can also carry analog signals (for advanced designs that implement their own analog circuits/devices).

· All 38 GPIOs have digital control circuitry that allows them to be configured after power-on for different modes via registers accessible via firmware or HKSPI .

· The lowest-numbered 5 of the 38 always power up in a fixed Caravel management mode: `IO[4:1]` are enabled for exclusive use as the HKSPI interface; `IO[0]` is enabled as a debug pin. Therefore, these are not immediately usable by the user design area, until they are reconfigured at run-time (if desired), in which case the respective HKSPI/debug functions are disabled (until explicitly re-enabled by firmware, or until the next power-on, only).

· The remaining 33 of 38 must have their default power-on modes masked-programmed, i.e. a simple " `user_defines` " configuration file defines how they will be hard-set during fabrication. This ensures they have a known desired mode immediately after power-up, though they can still be reconfigured at run-time.

· NOTE: The GPIO configuration modes are retained across SoC resets, but not across power cycling.

· Out of the above 33 pins, 29 (`IO[35:7]`) can also be configured to support direct analog connections , thus disabling their digital control circuitry for designers who specifically want to include their own (or ready-made) analog circuits in the user project area.

Each of the 38 GPIOs provides multiple ports into the user project area to enable direct connections to the user's design/logic, and meet all supported use cases:

· All 38 provide `io_in[*]` (input digital signal paths), `io_out[*]` (output digital signal paths), and `io_oeb[*]` ("Output Enable Bar " for setting the signal directions). Note that, at any given moment, whichever of these paths are actually usable or meaningful depends on what mode the pin is configured for. For example, `io_oeb` has no effect on changing the pin direction if the pin is already configured as an output.

· The 29 that are analog-capable also provide `analog_io[*]` ports.

> **ⓘ Note**

`io_in` always has an active input buffer that feeds the digital logic state of the pin back into the user project area depending on the voltage present on the pad (does it definitely? Are there cases where this is not true?)

Management GPIO pin

User GPIO configuration by firmware or HKSPI

User GPIO power-on configuration by user_defines

Standard GPIO configuration mode constants

USER modes

The presence of `..._USER_...` in the name of a GPIO mode constant indicates that this mode will activate the pin's respective User Project Wrapper edge ports, meaning the user project can be directly connected to the GPIO's digital paths. In other words, the user project has exclusive access to the pin when one of these modes are used.

Standard GPIO "USER" mode constants

Named constant	Value	Description
GPIO_MODE_USER_STD_INPUT_NOPULL	0x0402	(TBC! TO BE CONFIRMED or COMPLETED)
GPIO_MODE_USER_STD_INPUT_PULLDOWN	0x0c00	(TBC! TO BE CONFIRMED or COMPLETED)
GPIO_MODE_USER_STD_INPUT_PULLUP	0x0800	(TBC! TO BE CONFIRMED or COMPLETED)
GPIO_MODE_USER_STD_OUTPUT	0x1808	(TBC! TO BE CONFIRMED or COMPLETED)
GPIO_MODE_USER_STD_BIDIRECTIONAL	0x1800	(TBC! TO BE CONFIRMED or COMPLETED)
GPIO_MODE_USER_STD_OUT_MONITORED	0x1802	(TBC! TO BE CONFIRMED or COMPLETED)
GPIO_MODE_USER_STD_ANALOG	0x000a	(TBC! TO BE CONFIRMED or COMPLETED)

MGMT modes

The presence of `..._MGMT_...` in the name of a GPIO mode constant indicates that this mode will give the Management SoC exclusive access to the pin, thus deactivating the pin's respective

User Project Wrapper edge ports. In other words, the Management SoC will be able to read/write/control the pin, while the user project will not.

There are two exceptions, however:

- There is a |upw| input buffer always attached to the pin pad, so long as the GPIO is not configured for one of the ANALOG modes. This means the user project is always able to sense the digital state of the pin, including if it is being used in any USER or MGMT input/output mode. This also means that the |soc| could directly drive GPIOs in a way that loop back into the user project.

- Connections to the analog paths of User Project Wrapper edge ports are always physically direct and cannot be disconnected. If your user project makes such connections, be careful about configuring the GPIO for any non- `ANALOG` mode (whence the GPIO's digital circuitry will be active simultaneously).

Standard GPIO "MGMT" mode constants

Named constant	Value	Description
`GPIO_MODE_MGMT_STD_INPUT_NOPULL`	0x0403	(TBC! TO BE CONFIRMED or COMPLETED)
`GPIO_MODE_MGMT_STD_INPUT_PULLDOWN`	0x0c01	(TBC! TO BE CONFIRMED or COMPLETED)
`GPIO_MODE_MGMT_STD_INPUT_PULLUP`	0x0801	(TBC! TO BE CONFIRMED or COMPLETED)
`GPIO_MODE_MGMT_STD_OUTPUT`	0x1809	(TBC! TO BE CONFIRMED or COMPLETED)
`GPIO_MODE_MGMT_STD_BIDIRECTIONAL`	0x1801	How is the OEB handled in this mode?
`GPIO_MODE_MGMT_STD_ANALOG`	0x000b	(TBC! TO BE CONFIRMED or COMPLETED)

Custom modes

You can set your own configuration bitfield. DEFINE HOW.

io_oeb conventions

GPIO pin ports map

Logic Analyzer

UART (RS232 Serial Interface)

Wishbone Interface

SPI Controller

> **ⓘ Tip**
>
> Not to be confused with the firmware SPI bus or HKSPI .

Counter/Timer

Housekeeping and HKSPI

Housekeeping (HK) describes a subset of SoC control registers which – besides being addressable by the Caravel CPU – have been made externally-accessible through a "Housekeeping SPI" (HKSPI) interface. This interface coexists on four of the Caravel GPIO pins (`mprj_io[4:1]`) and is always enabled at power-on (but can be deactivated at run-time). Importantly, this means any simple external SPI controller can always take control over certain blocks of the chip frame/SoC. This feature is typically used for bring-up debugging purposes, and could be used for diagnostic/maintenance purposes in a field application.

With Housekeeping, you can externally access certain SoC registers (some read-only, some read/write) to inspect some aspects of the SoC state or otherwise control its behaviour, including to:

· Verify the chipIgnite product ID and read your chip's unique Project ID .

· Alter the chip's core clock paths/speed via DLL and DCO .

· Redirect the internal clock signals out via GPIO pins.

· Reset the Caravel RISC-V CPU.

· Reconfigure GPIO pin functions.

· Take over and optionally reprogram a firmware SPI Flash ROM chip connected to the Caravel CPU.

"HKSPI" is an SPI responder that can be accessed from an external controller (e.g. the Caravel Eval Board) through a standard 4-pin SPI serial interface. The SPI implementation is mode 0 ⬈ , with new data on `SDI` captured on the `SCK` rising edge, and output data presented on the falling edge of `SCK` (to be sampled on the next `SCK` rising edge). The SPI pins are shared with user area GPIO.
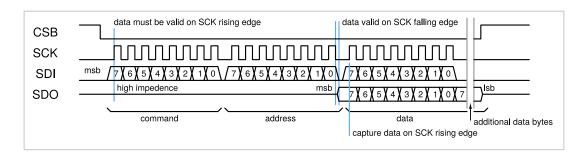
Housekeeping SPI pins

Housekeeping SPI external interface pins

GPIO pin	HKSPI pin	Dir	Function
`mprj_io[1]`	SDO	Output	Serial data out, clocked out on falling edge of `SCK`
`mprj_io[2]`	SDI	Input	Serial data in, clocked in on rising edge of `SCK`
`mprj_io[3]`	CSB	Input	"Chip Select bar" (falling edge starts an HKSPI transaction)
`mprj_io[4]`	SCK	Input	Serial clock.

Housekeeping SPI protocol definition

All input is in groups of 8 bits. Each byte is input MSB (most-significant-bit) first.

Every command sequence requires one command word (8 bits), followed by one address word (8 bits), followed by one or more data words (8 bits each), according to the data transfer modes described in Housekeeping SPI modes .

Housekeeping SPI signalling

Addresses are read in sequence from lower values to higher values.

Therefore groups of bits larger than 8 should be grouped such that the lowest bits are at the highest address. Any bits additional to an 8-bit boundary should be at the lowest address.

Data is captured from the register map in bytes on the falling edge of the last SCK before a data byte transfer. Multi-byte transfers should ensure that data do not change between byte reads.

`CSB` pin must be low to enable an SPI transmission. Data are clocked by pin `SCK` , with data valid on the rising edge of `SCK` . Output data is received on the `SDO` line. `SDO` is held high-impedance when `CSB` is high and at all times other than the transfer of data bits on a read command. `SDO` outputs become active on the falling edge of `SCK` , such that data are written and read on the same `SCK` rising edge.

After `CSB` is set low, the SPI is always in the "command" state, awaiting a new command.

The first transferred byte is the command word, interpreted according to the Housekeeping SPI command word definitions .

Housekeeping SPI command word definitions

Word	Meaning
`00000000`	No operation

Word	Meaning
`10000000`	Write in streaming mode
`01000000`	Read in streaming mode
`11000000`	Simultaneous Read/Write in streaming mode
`11000100`	Pass-through (Management) Read/Write in streaming mode
`11000110`	Pass-through (User) Read/Write in streaming mode
`10nnn000`	Write in n-byte mode (up to 7 bytes)
`01nnn000`	Read in n-byte mode (up to 7 bytes)
`11nnn000`	Simultaneous Read/Write in n-byte mode (up to 7 bytes)

> ℹ️ **Note**
>
> All other words are reserved and act as no-operation if not defined by the SPI responder module.

Housekeeping SPI modes

The two basic modes of operation are **streaming mode** and **n-byte mode** .

In **streaming mode** operation, the data is sent or received continuously, one byte at a time, with the internal address incrementing for each byte. Streaming mode operation continues until `CSB` is raised to end the transfer.

In **n-byte mode** operation, the number of bytes to be read and/or written is encoded in the command word, and may have a value from 1 to 7 (note that a value of zero implies streaming mode). After `n` bytes have been read and/or written, the SPI returns to waiting for the next command. No toggling of CSB is required to end the command or to initiate the following command.

Housekeeping SPI Pass-through mode

The pass-through mode puts the CPU into immediate reset, then sets `FLASH_CSB` low to initiate a data transfer to the CPU's attached SPI flash. After the pass-through command byte has been issued, all subsequent SPI signaling on `SDI` and `SCK` are applied directly to the SPI flash (pins `FLASH_IO0` and `FLASH_CLK` , respectively), and the SPI flash data output (pin `FLASH_IO1`) is applied directly to `SDO` , until the `CSB` pin is raised. When `CSB` is raised, the `FLASH_CSB` is also raised, terminating the data transfer to the SPI flash. The CPU is brought out of reset, and starts executing instructions at the program start address.

This mode allows the SPI flash to be programmed from the same SPI communication channel as the housekeeping SPI, without the need for additional wiring to the SPI flash chip.

There are two pass-through modes, as stated in the Housekeeping SPI command word definitions :

- **Pass-through (Management)** mode is to the primary SPI flash used by the Management SoC (Firmware Flash SPI), as described above.

- **Pass-through (User)** mode is to `mprj_io[11:8]` . Consider a user design in the User Project Wrapper that uses these pins as its own implementation of an SPI controller and maps IOs 8-11 respectively to each of `flash2_csb` , `flash2_sck` , `flash2_io0` , and `flash2_io1` – Pass-through (User) mode can take over these pins and control an SPI device connected via these pins.

Assuming SPI memory chips are connected to each of the interfaces described above, the pass-through modes allow a controller external to the Caravel chip to control/read/erase/program either SPI memory chip from a host computer without requiring a separate external bus. Both pass-through modes only connect to I/O pins 0 and 1 of the SPI interface, and so must operate only in the 4-pin (single-data-rate) SPI mode. The user project may, of course, elect to operate its own SPI implementation in QSPI mode by incorporating two additional pins into its design (for SPI I/O pins 2 and 3).

Housekeeping SPI addresses

The purpose of the housekeeping SPI is to give access to certain system values and controls independently of the CPU. The housekeeping SPI can be accessed even when the CPU is in full reset. Some control registers in the housekeeping SPI affect the behaviour of the CPU in a way that can be potentially detrimental to the CPU operation, such as adjusting the trim value of the digital frequency-locked loop generating the CPU core clock.

While both the CPU and HKSPI can access the same registers that control/inspect certain SoC functions, the addresses are different between the two interfaces. Namely, accessing these registers via HKSPI uses an 8-bit address only, while accessing them via the CPU uses 32-bit addresses scattered through the range `0x26000000` – `0x262FFFFF` with no correlation between the addresses of the two interfaces.

Register Address	msb 7	6	5	4	3	2	1	lsb 0	comments
0x00	SPI status and control								unused/ undefined
0x01	unused				manufacturer_ID[11:8] (= 0x4)				read-only
0x02	manufacturer_ID[7:0] (= 0x56)								read-only
0x03	product_ID (= 0x10)								read-only
0x04– 0x07	user_project_ID (unique value per project)								read-only
0x08	unused						PLL DCO enable	PLL enable	default 0x02
0x09	unused							PLL bypass	default 0x01
0x0A	unused							CPU IRQ	default 0x00
0x0B	unused							CPU reset	default 0x00
0x0C	unused							CPU trap	read-only
0x0D– 0x10	DCO trim (26 bits) (= 0x3ffefff)								default 0x3ffefff
0x11	unused		PLL output divider 2		PLL output divider				default 0x12
0x12	unused				PLL feedback divider				default 0x04

Housekeeping SPI register map

Housekeeping SPI registers

Name	Register address	Description
manufacturer_ID	`0x01` (*low 4 bits*) and `0x02`	The 12-bit manufacturer ID for efabless is `0x456`
product_ID	`0x03`	The product ID for the Caravel harness chip is 0x10
user_project_ID	`0x04` to `0x07`	The 4-byte (32bit) user project ID is metal-mask programmed on each project before tapeout, with a unique number given to each user project.
PLL enable	`0x08` *bit 0*	This bit enables the digital frequency-locked-loop clock multiplier. The enable should be applied prior to turning off the PLL bypass to allow the PLL time to stabilize before using it to drive the CPU clock.
PLL DCO enable	`0x08` *bit 1*	The PLL can be run in DCO mode, in which the feedback loop to the driving clock is removed, and the system operates in free-running mode, driven by the ring oscillator which can be tuned between approximately 90 to 200MHz by setting the trim bits (check PLL trim) (NEED TO UPDATE THIS TO MATCH LEO'S RECENT CHARACTERIZATION and do some more char)
PLL bypass	`0x09` *bit 0*	When enabled, the PLL bypass switches the clock source of the CPU from the PLL output to the external CMOS clock (pin `C9`). The default value is `0x1` (CPU clock source is the external CMOS clock).
CPU IRQ	`0x0A` *bit 0*	This is a dedicated manual interrupt driving the CPU IRQ channel 6. The bit is not self-resetting, so while the rising edge will trigger an interrupt, the signal must be manually set to zero before it can trigger another interrupt.

CPU reset	`0x0B` *bit 0*	The CPU reset bit puts the entire CPU into a reset state. This bit is not self-resetting and must be set back to zero manually to clear the reset state
CPU trap	`0x0C` *bit 0*	If the CPU has stopped after encountering an error, it will raise the trap signal. The trap signal can be configured to be read from a GPIO pin, but as the GPIO state is potentially unknowable, the housekeeping SPI can be used to determine the true trap state.
PLL trim	`0x0D` *(all bits)* to `0x10` *(lower two bits)*	The 26-bit trim value can adjust the DCO frequency over a factor of about two from the slowest (trim value `0x3ffffff`) to the fastest (trim value `0x0`). Default value is `0x3ffefff` (1 step higher than the slowest trim). Note that this is a thermometer-code trim, where each bit provides an additional (approximately) 250ps delay (on top of a fixed delay of 4.67ns). The fastest output frequency is approximately 215MHz while the slowest output frequency is approximately 90MHz (check PLL trim) (NEED TO UPDATE THIS TO MATCH LEO'S RECENT CHARACTERIZATION and do some more char)
PLL output divider	`0x11` *bits 2-0*	The PLL output can be divided down by an integer divider to provide the core clock frequency. This 3-bit divider can generate a clock divided by 2 to 7. Values 0 and 1 both pass the undivided PLL clock directly to the core (and should not be used, as the processor does not operate at these frequencies).
PLL output divider (2)	`0x11` *bits 5-3*	The PLL 90-degree phase output is passed through an independent 3-bit integer clock divider and provided to the user project space as a secondary clock. Values 0 and 1 both pass the undivided PLL clock, while values 2 to 7 pass the clock divided by 2 to 7, respectively.
PLL feedback divider	`0x12` *bits 4-0*	The PLL operates by comparing the input clock (pin `C9`) rate to the rate of the PLL clock divided by the feedback divider value (when running in PLL mode, not DCO mode). The feedback divider must be set such that the external clock rate multiplied by the feedback divider value falls between 90 and 214 MHz (preferably centered on this range, or approximately 150 MHz) (check PLL trim) (NEED TO UPDATE THIS, and the calculation below, TO MATCH LEO'S

RECENT CHARACTERIZATION and do some more char) . For example, when using an 8 MHz external clock, the divider should be set to 19 (`19 * 8 = 152`). The DCO range and the number of bits of the feedback divider implies that the external clock should be no slower than around 4 to 5 MHz.

Interrupts (IRQs)

Registers and Memory Map

High-level memory map

Register list

Analog Connections

NOTES:

- Allowing a pin to switch between digital and analog modes should be possible, but probably difficult. Might really only work well for OpenFrame, given how long it takes the CPU to signal everything in the GPIOs/chip.

- Comment on io_oeb and io_out when pads are configured for analog

Advanced Guides

Caravel CPU code execution

Executing code from RAM

Custom ISRs

Power-on behavior

Management Core wrapper

The Caravel "Management Core Wrapper" is designed in a way to allow the implementation of different management cores. Early versions of the Google-sponsored "Open MPW" Caravel SoCs used a PicoRV32 core, while the current version officially offered by Efabless is "Caravel V6.0" (in use since Open MPW-6) and is generated using Litex with a VexRiscv CPU core.

The common parts of the management core wrapper include:

· Housekeeping and HKSPI.

· GPIO configuration blocks (power-on defaults and runtime-reprogrammable).

· User Project Wrapper interface pins, namely: Logic Analyzer; IRQs; Wishbone (inc. `wb_rst_i` , and `wb_clk_i` and `user_clock2` clock sources); and power rings (for a PDN of up to 4 power domains).

· Management protection.

· Clocking module (DLL/DCO).

· POR (Power-On Reset) module.

Building Caravel using Litex

Misc

- CPU as a testbench, e.g. using the CPU to drive **inputs** into UPW on GPIO pins.

- Reset behaviour, sources, options, e.g. is it a good idea to rely solely on *wb_rst_i* ?

Caravel Chip Bring-up and the Caravel Evaluation Board

Debug Port

The Caravel CPU supports debugging via the debug pin. It can be accessed through a dedicated UART port configured as a Wishbone master. The baud rate for the port is fixed; nominally i tis 9600 baud given a 10MHz SoC core clock.

See the following reference for more information: VexRiscv DebugPlugin ⬀

Caravan Specifics

This is a summary of all things that are common and different between Caravan and the normal Caravel.

Common features

Caravan differences

Caravel Mini Specifics

This is a summary of all things that are common and different between Caravel Mini and the normal Caravel.

Common features

Caravel Mini differences

Supplementary Figures

Specifications and Ratings

Absolute maximum ratings

Type	minimum	typical	maximum	units
Supply voltage (VDDIO)	1.8	3.3	5.0	V
Core digital supply voltage (VCCD)	1.62	1.8	1.98	V
Junction temperature	-40	27	100	$^{\circ}C$
V_{OH}	$0.8 \cdot {VDDIO}$		V	
V_{OL}			0.4	V
Management area power		TBD		mW
Storage area power		TBD		mW
GPIO voltage	0		VDDIO	V
GPIO frequency	0		50	MHz
Management clock	0		40	MHz

Schematics and PCB Design

KiCad schematic and footprint

PCB design guidelines

Caravel Eval Board and M.2 card

Glossary

Active-low

A signal whose named intent/mode is considered to be asserted when the signal is low (i.e. logic 0, or GND), and deasserted when the signal is high (i.e. logic 1, or positive). Sometimes also referred to as "inverted logic" or "negative logic". Example: A device with an input signal name like `reset_n` (note the `_n` suffix) is considered to actively be in the "reset" state when the signal is low, and otherwise running normally (and not in reset) when the signal is high. A similar convention is `resetb` where the `b` suffix means " Bar ".

Bar

Used as a suffix (e.g. "Output Enable Bar"), this typically means the signal is " active-low ". In the name of a signal, this is often indicated by a `b` or `_b` suffix (e.g. `oeb` might mean "Output Enable Bar"), where the name would normally be rendered in a schematic with a horizontal line (or "bar") over the signal name.

bring-up

The process of getting a chip to operate for the first time after receiving it as a fabricated device. During this time, many measurements and experiments may be required, as well as debugging. This includes to verify that the core features of the frame/SoC are functioning as expected, and that your design is able to respond in a set of your own expectations.

Caravel Eval Board

The bring-up/development/evaluation board for chipIgnite/Caravel chips. Typically one board is supplied with every chipIgnite order that includes QFN-packaged Caravel chips. For more information, see https://github.com/efabless/caravel_board ⬈ and note that you can purchase a demo board from the Efabless Store ⬈ – the demo board includes 1 Caravel demo chip.

Caravel SoC

Define me

crt0

Initial "C Runtime" bootstrapping routines. Code built into the assembly process of a C program that is executed before the `main()` function is called. Responsible for loading the initial system/memory state, including initializing any global/static variables and optionally loading read-only data.

DLL

Delay-Locked Loop. Very similar to a PLL . In Caravel, there is a DLL which is an all-digital SoC peripheral that can be used to generate new clock frequencies from an internal or external clock source.

ESD

Electrostatic Discharge. Often used in the context of "ESD Protection", i.e. dedicated circuits that help protect integrated circuits by suppressing high voltage spikes on chip pins, that can resulting from static electric charges that build up during the handling of parts.

GDS

Define me

Hardening

The process of generating a final silicon layout (and hence GDS file) from potentially multiple parts, including synthesis of higher-level descriptions of digital logic.

ISA

Instruction Set Architecture.

Litex

Define me

Management Area

Define me

Management Core

Define me

Management Core Wrapper

Define me

Management SoC

Define me

PDN

Power Delivery Network.

PLL

Phase-Locked Loop. A device commonly used in FPGAs and in Caravel to derive a new clock frequency/phase from a supplied clock source. Typically allows for a clock source to be multiplied in frequency by an integer value, and then divided by a second integer value to produce a new clock frequency. Sometimes may offer multiple multipliers/dividers in order to produce multiple clocks. Compare: DLL

POR

Power-On Reset. A circuit that ensures a stable reset sequence during chip power-on, thus ensuring a stable system state if a dedicated external reset is not otherwise implemented.

Project ID

Every unique silicon layout (e.g. customer project) fabricated with Efabless chipIgnite has a unique 32-bit "Project ID" assigned by Efabless and included in the silicon layout. The Project ID is accessible by the Caravel SoC (and via HKSPI) as a read-only 32-bit value, but is also present as "GDS art" text in the padring, rendered as 8 hex digits. Most Project IDs are of the pattern `YYMMhhhh` where `hhhh` is a random value assigned by Efabless at the initialization of the project, and `YYMM` is the shuttle number (e.g. `2409`) and itself is formed of the last two digits of the shuttle year and the month number. An example Project ID (as a hex string) is `240476A0` which is Tiny Tapeout 6 ⬈ , on the April 2024 shuttle.

Note that when using the SoC or HKSPI to read the 32-bit value of the Project ID, some shuttles had the project ID bits in reverse order, e.g. `240476A0` (which in binary is `0010_0100_0000_0100_0111_0110_1010_0000`) would be read as `056E2024` (which is the binary string in reverse: `0000_0101_0110_1110_0010_0000_0010_0100`).

PVT

Short for "Process, Voltage, Temperature" and typically used in the context "PVT-dependent", meaning that the exact behaviour/characteristics of something is affected (or otherwise likely to deviate from typical stated figures) by virtue of: variations that naturally occur in the fabrication process; variations in precise voltages in the circuit; and variations in ambient temperature.

QFN

Quad Flat No-leads IC package ⬈ . A plastic-encapsulated chip package with pin pads around all 4 sides.

RV32I

A particular variant of the RISC-V ISA , denoting a RISC-V core that uses 32-bit addressing and instructions (no compressed instructions), with basic integer operations (no hardware multiply/divide or floating-point).

SoC

System on a Chip. A combination of chip modules that provide a system of functionality, often including a CPU and other useful peripheral devices implemented in silicon.

SPI

Serial Peripheral Interface ⬈ . A common 4-wire interface for simple serial communication with a peripheral device, driven by a controller. Often used between chips, and capable of multi-megabit-per-second transfers.

UPW

Short for User Project Wrapper .

User Project Wrapper

The design area reserved for a user project. It has a fixed location and dimensions within the overall Caravel chip die area, and fixed pin placements around all 4 edges that a user design must connect to in order to interface with the Caravel SoC and/or GPIOs.

Typically the User Project Wrapper also has a PDN that is generated by the hardening flow.

VexRiscv

Define me

WLCSP

Wafer-Level redistribution Chip-Scale Package ↗ . A minimal chip package usually with a "redistribution" layer that attaches bare bond pads of a silicon die to ball grid array (BGA) solder balls via tiny wires.

XIP

Execute In Place: Code is directly loaded and executed from an external memory as needed, without the need for user-driven caching control, buffering, translation, logic, etc.