Efabless Caravel “harness” SoC

Description:

The efabless Caravel chip is a ready-to-use test harness for creating designs with the Google/SkyWater 130nm open PDK. The Caravel harness comprises a small RISC-V microprocessor based on the simple 2-cycle PicoRV32 RISC-V core implementing the RV32IMC instruction set (see http://riscv.org/), a 32-bit wishbone bus, and an approximately 2.8 mm × 2.8 mm open area for the placement of user IP blocks.

Core:

The processor core is the PicoRV32 design (see http://github.com/cliffordwolf/picorv32). The full core description is available from the github site. The hardware implementation is the “large” variant, incorporating options IRQ, MUL, DIV, BARREL_SHIFTER, and COMPRESSED_ISA (16-bit instructions).

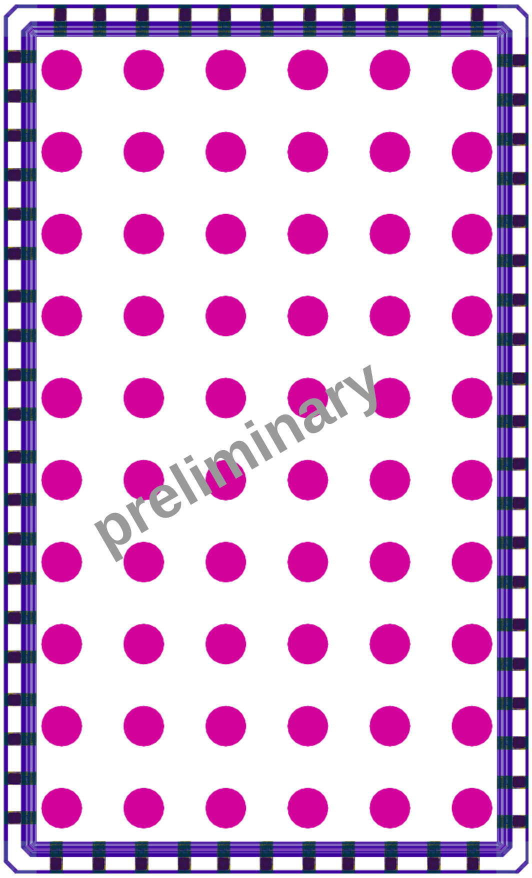
Core clock rate: (TBD) MHz maximum over all PVT conditions (likely around 50 MHz guaranteed)

Features:

Functions/features of the SoC include:

- 1 SPI flash controller
- 1 UART
- 1 SPI master
- 2 counter-timers
- 1 dedicated general-purpose input/output channel
- 27 shared general-purpose input/output channels
- 8k word (32768 bytes × 8 bits) on−board SRAM

- All-digital frequency-locked loop clock multiplier
- 128 bit logic analyzer



Caravel harness die (3.2mm × 5.3mm)

License:

The Caravel chip is an open-source design, licensed under the terms of Apache 2.0.

Repository:

The complete Caravel chip design may be obtained from the git repository located at https://github.com/efabless/caravel/.

Process:

The efabless Caravel harness chip is fabricated in SkyWater 0.13µm CMOS technology, with process specifications and data at https://github.com/google/skywater−pdk/.

Efabless Caravel “harness” SoC

Version:

This document corresponds to version 1 of the Caravel processor (October 2020).

Revision history:

Documentation revision 0 (October 14, 2020)

Documentation revision 1 (October 19, 2020)

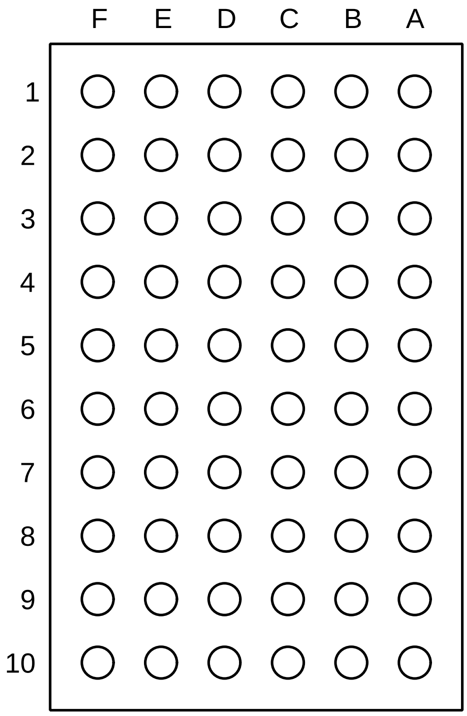
> Changed pinout to add four ground (bump bond) pads in the center, and to move the user area power supply padframe pads into the user area for easier connecting.

Documentation revision 2 (October 27, 2020)

> More updates corresponding to RTL-level changes in the description.

Pinout (6x10 WLCSP)

A1	mprj_io[23]
A2	vccd2
A3	mprj_io[25]
A4	mprj_io[27]
A5	mprj_io[29]
A6	mprj_io[31]
A7	mprj_io[32]
A8	mprj_io[35]
A9	mprj_io[37]
A10	vccd
B1	mprj_io[21]
B2	mprj_io[22]
B3	vssa2
B4	mprj_io[26]
B5	mprj_io[28]
B6	mprj_io[30]
B7	vssd2
B8	mprj_io[34]
B9	mprj_io[36]
B10	resetb
C1	mprj_io[19]
C2	mprj_io[20]
C3	mprj_io[24]
C4	vddio
C5	vssio/vssa/vssd
C6	vssio/vssa/vssd
C7	vdda2
C8	mprj_io[33]
C9	clock
C10	flash csb
D1	mprj_io[18]
D2	mprj_io[17]
D3	mprj_io[13]
D4	vdda1
D5	vssio/vssa/vssd
D6	vssio/vssa/vssd
D7	mprj_io[0]/JTAG
D8	flash clk
D9	flash io1
D10	flash io0
E1	mprj_io[16]
E2	mprj_io[14]
E3	mprj_io[11]/flash2 io1
E4	mprj_io[9]/flash2 sck
E5	mprj_io[7]/irq
E6	vssa1
E7	mprj_io[5]/ser_rx
E8	mprj_io[3]/CSB
E9	mprj_io[1]/SDO
E10	gpio
F1	mprj_io[15]
F2	vccd1
F3	mprj_io[12]
F4	mprj_io[10]/flash2 io0
F5	mprj_io[8]/flash2 csb
F6	vssd1
F7	mprj_io[6]/ser_tx
F8	mprj_io[4]/SCK
F9	mprj_io[2]/SDI
F10	vdda



Package as viewed from the bottom.

Pin Description (6x10 WLCSP)

Pin #	*Name*	*Type*	*Summary description*
A9, B9, A8, B8, C8, A7, A6, B6, A5, B5, A4, B4, A3, C3, A1, B2, B1, C2, C1, D1, D2, E1, F1, E2, D3, F3, E3, F4, E4, F5, E5, F7, E7, F8, E8, F9, E9, D7	mprj_io[37:0]	Digital I/O	General purpose configurable digital I/O with General purpose configurable digital I/O with pullup/pulldown, input or output, enable/disable, analog output, high voltage output, slew rate control. Shared between the user project area and the management SoC.
D8	flash clk	Digital out	Flash SPI clock
F5	flash csb	Digital out	Flash SPI chip select
E3, F4	flash io1:0	Digital I/O	Flash SPI data input/output
C9	clock	Digital in	External CMOS 3.3V clock source
B10	resetb	Digital in	SoC system reset (sense inverted)
E9	SDO	Digital out	Housekeeping serial interface data output
F9	SDI	Digital in	Housekeeping serial interface data input
E8	CSB	Digital in	Housekeeping serial interface chip select
F8	SCK	Digital in	Housekeeping serial interface clock
F7	ser_tx	Digital out	UART transmit channel
E7	ser_rx	Digital in	UART receive channel
E5	irq	Digital in	External interrupt
E10	gpio	Digital I/O	Management GPIO/user power enable
D7	JTAG	Digital I/O	JTAG system access
F5	flash2 csb	Digital out	User area QSPI flash enable (sense inverted)
E4	flash2 sck	Digital out	User area QSPI flash clock
E3, F4	flash2 io1:0	Digital I/O	User area QSPI flash data
F9	spi_sdo	Digital out	Serial interface master data output
F8	spi_sck	Digital out	Serial interface master clock
E8	spi_csb	Digital out	Serial interface master chip select
E9	spi_sdi	Digital in	Serial interface masterdata input
C4	vddio	3.3V Power	ESD and padframe power supply
F10	vdda	3.3V Power	Management area power supply
A10	vccd	1.8V Power	Management area digital power supply
C5, C6, D5, D6	vssio/vssa/vssd	Ground	ESD, padframe, and management area ground
D4	vdda1	3.3V Power	User area 1 power supply
F2	vccd1	1.8V Power	User area 1 digital power supply
E6	vssa1	Ground	User area 1 ground
F6	vssd1	Ground	User area 1 digital ground
C7	vdda2	3.3V Power	User area 2 power supply
A2	vccd2	1.8V Power	User area 2 digital power supply
B3	vssa2	Ground	User area 2 ground
B7	vssd2	Ground	User area 2 digital ground

Standard package: WLCSP (bump bond)
Package size: 3.2 mm × 5.3 mm
Bump pitch: 0.5 mm

General Purpose I/O

GPIO (pin E10)

The GPI pin is a single assignable general-purpose digital input or output that is available only to the management SoC and cannot be assigned to the user project area. On the test board provided with the completed user projects, this pin is used to enable the voltage regulators generating the user area power supplies.

The basic function of the GPIO is illustrated below. All writes to **`reg_gpio_data`** are registered. All reads from **reg_gpio_data** are immediate.

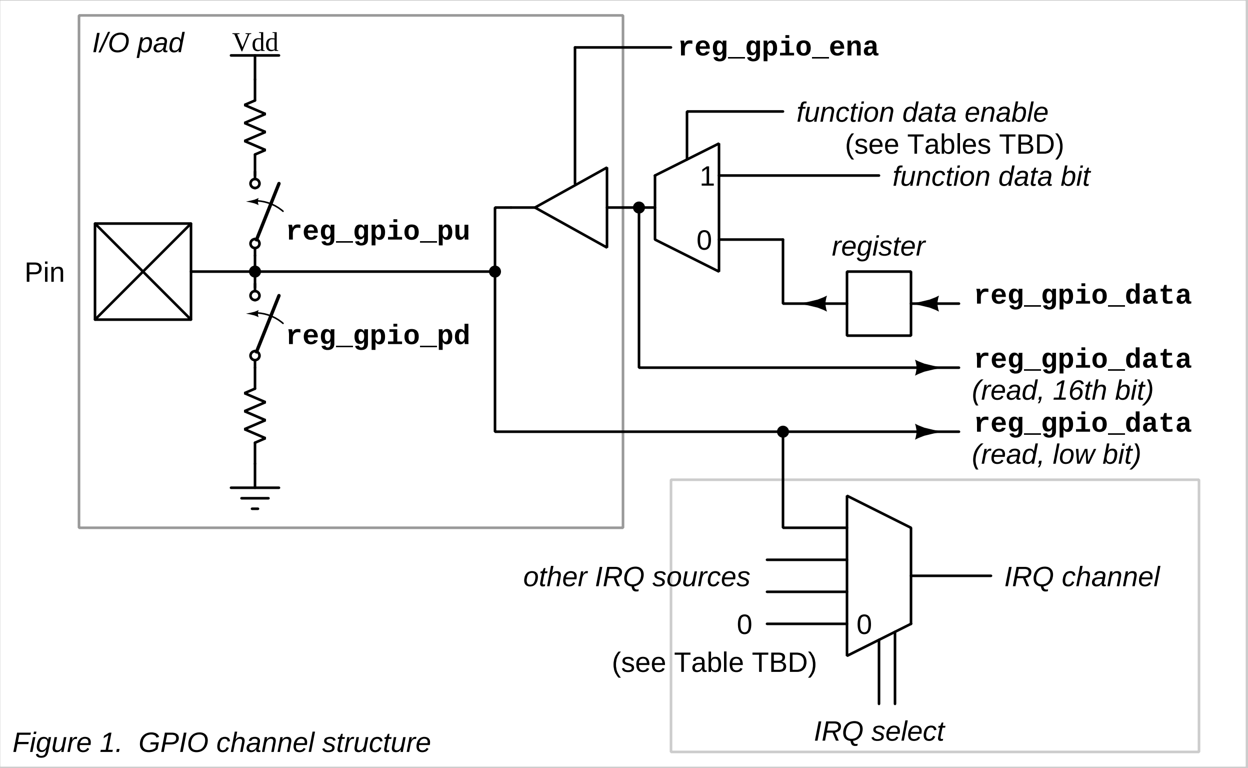


Figure 1. GPIO channel structure

GPIO memory address map:

C header name	*address*	*description*
`reg_gpio_data`	**`0x21000000`**	GPIO input/output (low bit) GPIO output readback (16th bit)
`reg_gpio_ena`	**`0x21000004`**	GPIO output enable (0 = output, 1 = input)
`reg_gpio_pu`	**`0x21000008`**	GPIO pullup enable (1 = pullup, 0 = none)
`reg_gpio_pd`	**`0x2100000c`**	GPIO pulldown enable (1 = pulldown, 0 = none)
`reg_pll_out_dest`	**`0x2f000000`**	PLL clock output destination (low bit)
`reg_trap_out_dest`	**`0x2f000004`**	Trap output destination (low bit)
`reg_irq7_source`	**`0x2f000008`**	IRQ 7 input source (low bit)

GPIO description, continued.

In the memory-mapped register descriptions below, each register is shown as 32 bits corresponding to the data bus width of the wishbone bus. Addresses, however, are in bytes. Depending on the instruction and data type, the entire 32-bit register can be read in one instruction, or one 16-bit word, or one 8-bit byte.

Table 1 **`reg_gpio_data`**

<table>
<tr><td colspan="8">0x21000003</td><td colspan="8">0x21000002</td><td colspan="8">0x21000001</td><td colspan="8">0x21000000</td><td>address</td></tr>
<tr><td colspan="16">GPIO output readback</td><td colspan="16">GPIO input/output</td><td>value</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>bit</td></tr>
</table>

Writing to the address low bit always sets the registered value at the GPIO.
Writing to address bit 16 has no effect.
Reading from the address low bit reads the value at the chip pin.
Reading from address bit 16 reads the value at the multiplexer output (see diagram).

Table 2 **`reg_gpio_ena`**

<table>
<tr><td colspan="8">0x21000007</td><td colspan="8">0x21000006</td><td colspan="8">0x21000005</td><td colspan="8">0x21000004</td><td>address</td></tr>
<tr><td colspan="16">(undefined, reads zero)</td><td colspan="16">GPIO output enable</td><td>value</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>bit</td></tr>
</table>

Bit 0 corresponds to the GPIO channel enable.
Bit value 1 indicates an output channel; 0 indicates an input.

Table 3 **`reg_gpio_pu`**

<table>
<tr><td colspan="8">0x2100000b</td><td colspan="8">0x2100000a</td><td colspan="8">0x21000009</td><td colspan="8">0x21000008</td><td>address</td></tr>
<tr><td colspan="16">(undefined, reads zero)</td><td colspan="16">GPIO pin pull-up</td><td>value</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>bit</td></tr>
</table>

Bit 0 corresponds to the GPIO channel pull-up state.
Bit value 1 indicates pullup is active; 0 indicates pullup inactive.

Table 4 **`reg_gpio_pd`**

<table>
<tr><td colspan="8">0x2100000f</td><td colspan="8">0x2100000e</td><td colspan="8">0x2100000d</td><td colspan="8">0x2100000c</td><td>address</td></tr>
<tr><td colspan="16">(undefined, reads zero)</td><td colspan="16">GPIO pin pull-down (inverted)</td><td>value</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>bit</td></tr>
</table>

Bit 0 corresponds to the GPIO channel pull-down state.
Bit value 1 indicates pullup is active; 0 indicates pulldown is inactive.

GPIO description, continued.

Table 5 **reg_pll_out_dest**

0x2f000003	0x2f000002	0x2f000001	**0x2f000000**	address
(undefined, reads zero)			PLL clock dest.	value
31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0	bit

The low bit of this register directs the output of the core clock to the GPIO channel, according to the following table:

Register byte **0x2f000000** value		Clock output directed to this channel
0	**0**	(none)
1	**1**	Core PLL clock to GPIO out

Note that a high rate core clock (*e.g.,* 80MHz) may be unable to generate a full swing on the GPIO output.

Table 6 **reg_trap_out_dest**

0x2f000007	0x2f000006	0x2f000005	**0x2f000004**	address
(undefined, reads zero)			trap signal dest.	value
31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0	bit

The low bit of this register directs the output of the processor trap signal to the GPIO channel, according to the following table:

Register byte **0x2f000004** value		Trap signal output directed to this channel
0	**0**	(none)
1	**1**	GPIO

Table 7 **reg_irq7_source**

0x2f00000b	0x2f00000a	0x2f000009	**0x2f000008**	address
(undefined, reads zero)			IRQ 7 source	value
31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0	bit

The low bit of this register directs the input of the GPIO to the processor's IRQ7 channel, according to the following table:

Register byte **0x2f000008** value		This channel directed to IRQ channel 7
0	**00**	(none)
1	**01**	GPIO

Housekeeping SPI

SDI (pin F9), CSB (pin E8), SCK (pin F8), and SDO (pin E9)

The “housekeeping” SPI is an SPI slave that can be accessed from a remote host through a standard 4-pin 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 general-purpose I/O.

SPI protocol definition

All input is in groups of 8 bits. Each byte is input msb 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 defined below.

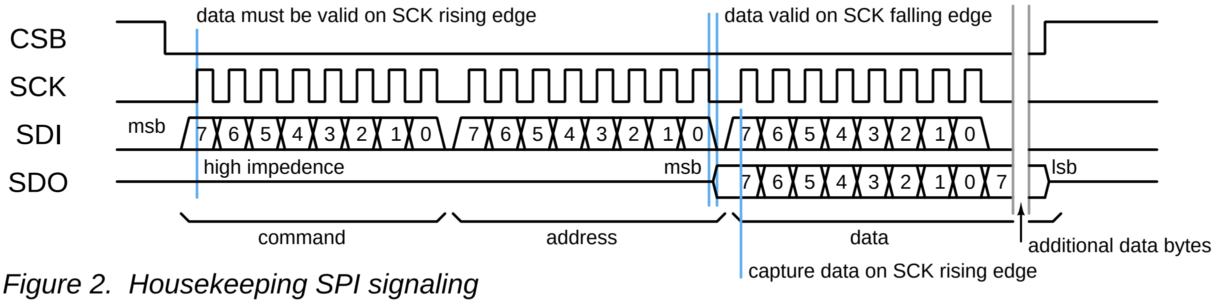


Figure 2. Housekeeping SPI signaling

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 are 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 are 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 Table 8 below.

Table 8 Housekeeping SPI command word definition

`00000000`	No operation
`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).

All other words are reserved and act as no-operation if not defined by the SPI slave module.

SPI protocol definition *(continued)*

The two basic modes of operation are "streaming mode" and "n-byte mode". In "streaming mode" operation, data are 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.

Pass-thru mode

The pass-thru mode puts the CPU into immediate reset, then sets FLASH_CSB low to initiate a data transfer to the QSPI flash. After the pass-thru command byte has been issued, all subsequent SPI signaling on SDI and SCK are applied directly to the QSPI flash (pins FLASH_IO0 and FLASH_CLK, respectively), and the QSPI 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 QSPI flash. The CPU is brought out of reset, and starts executing instructions at the program start address.

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

There are two pass-thru modes. The first one corresponds to the primary SPI flash used by the management SoC. The second one corresponds to a secondary optional SPI flash that can be defined in the user project. The pass-thru mode allows a communications chip external to the Caravel chip program either SPI flash chip from a host computer without requiring separate external access to the SPI flash. Both pass-thru modes only connect to I/O pins 0 and 1 of the SPI flash chips, and so must operate only in the 4-pin SPI mode. The user project may elect to operate the SPI flash in quad mode using a 6-pin interface.

Housekeeping SPI registers

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 behavior of the CPU in a way that potentially can be detrimental to the CPU operation, such as adjusting the trim value of the digital frequency-locked loop generating the CPU core clock.

Under normal working conditions, the SPI should not need to be accessed unless it is to adjust the clock speed of the CPU. All other functions are purely for test and debug.

The housekeeping SPI can be accessed by the CPU from a running program by enabling the SPI master, and enabling the bit that connects the internal SPI master directly to the housekeeping SPI. This configuration then allows a program to read, for example, the user project ID of the chip. See the SPI master description for details.

manufacturer_ID register address 0x01 low 4 bits and register address 0x02

The 12-bit manufacturer ID for efabless is 0x456

product_ID register address 0x03

The product ID for the Caravel harness chip is 0x10

Housekeeping SPI registers *(continued)*

user project ID register addresses 0x04 to 0x07

The 4-byte (32 bit) user project ID is metal-mask programmed on each project before tapeout, with a unique number given to each user project.

PLL enable register address 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 register address 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 200 MHz by setting the trim bits (see below).

PLL bypass register address 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 register address 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 register address 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 register address 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 register addresses 0x0D (all bits) to 0x10 (lower 2 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 (slow trim, -1). Note that this is a thermometer-code trim, where each bit provides an additional (approximately) 250 ps delay (on top of a fixed delay of 4.67 ns). The fastest output frequency is approximately 215 MHz while the slowest output frequency is approximately 90 MHz.

PLL output divider register address 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) register address 0x11 bit 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.

Housekeeping SPI registers *(continued)*

PLL feedback divider register address 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). 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.

Table 9 **Housekeeping SPI register map**

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

QSPI Flash interface flash io0–1 (pins D10 to D9), flash csb (pin C10), and flash clk (pin D8)

The QSPI flash controller is automatically enabled on power-up, and will immediately initiate a read sequence in single-bit mode with pin "flash io0" acting as SDI (data from flash to CPU) and pin "flash io1" acting as SDO (data from CPU to flash). Protocol is according to, *e.g.,* Cypress S25FL256L.

The initial SPI instruction sequence is as follows:

`0xFF`	Mode bit reset
`0xAB`	Release from deep power-down
`0x03`	Read w/3 byte address
`0x00`	Program start address (**`0x10000000`**) (3 bytes) (upper byte is ignored)
`0x00`	
`0x00`	

The QSPI flash continues to read bytes, either sequentially on the same command, or issuing a new read command to read from a new address.

The behavior of the QSPI flash controller can be modified by changing values in the register below:

Table 10 **`reg_spictrl`**

<table>
<tr><td colspan="8"><code>0x2d000003</code></td><td colspan="8"><code>0x2d000002</code></td><td colspan="8"><code>0x2d000001</code></td><td colspan="8"><code>0x2d000000</code></td><td>address</td></tr>
<tr><td></td><td colspan="8">(unused)</td><td colspan="7">(see below)</td><td colspan="4">(unused)</td><td colspan="4"></td><td colspan="2"></td><td colspan="6">(see below)</td><td>value</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>bit</td></tr>
</table>

mask bit	default	description
31	1	QSPI flash interface enable
22–20	0	Access mode (see table below)
19–16	8	Dummy clock cycle count
11–8	0	Bit-bang OE FLASH_IO3–FLASH_IO0
5	0	Bit-bang FLASH_CSB
4	0	Bit-bang FLASH_CLK
3–0	0	Bit-bang value FLASH_IO3–FLASH_IO0

Access mode bit selection (bits 22–20):

0	**`000`**	Single bit per clock
1	**`001`**	Single bit per clock (same as 0)

All additional modes (QSPI dual and quad modes) cannot be used, as the management SoC only has pins for data lines 0 and 1.

The SPI flash can be accessed by bit banging when the enable is off. To do this from the CPU, the entire routine to access the SPI flash must be read into SRAM and executed from the SRAM.

Interrupt IRQ (pin E5)

The interrupt pin triggers the CPU interrupt channel 5.

External clock clock (pin C9)

The external clock functions as the source clock for the entire processor. On start-up, the processor runs at the same rate as the external clock. The processor program may access the housekeeping SPI to set the processor into PLL mode or DCO free-running mode. In PLL mode, the external clock is multiplied up by the feedback divider value to obtain the core clock. In DCO mode, the processor is driven by a trimmed free-running ring oscillator.

UART ser tx (pin F7) and ser rx (pin E7)

The UART is a standard 2-pin serial interface that can communicate with most similar interfaces at a fixed baud rate. Although the UART operates independently of the CPU, data transfers are blocking operations which will generate CPU wait states until the data transfer is completed.

The behavior of the UART can be modified by changing values in the registers below:

Table 11 **`reg_uart_clkdiv`**

<table>
<tr><td colspan="8">0x20000003</td><td colspan="8">0x20000002</td><td colspan="8">0x20000001</td><td colspan="8">0x20000000</td><td>address</td></tr>
<tr><td colspan="32">UART clock divider</td><td>value</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>bit</td></tr>
</table>

The entire 32 bit word encodes the number of CPU core cycles to divide down to get the UART data bit rate (baud rate). The default value is 1.

Example: If the external crystal is 12.5MHz, then the core CPU clock runs at 100MHz.
To get 9600 baud, 100E6 / 9600 = 10417 (hex value 0x28b1).

Table 12 **`reg_uart_data`**

<table>
<tr><td colspan="8">0x20000007</td><td colspan="8">0x20000006</td><td colspan="8">0x20000005</td><td colspan="8">0x20000004</td><td>address</td></tr>
<tr><td colspan="24">(unused, value is 0x0)</td><td colspan="8"></td><td>value</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>bit</td></tr>
</table>

Writing a value to this register will immediately start a data transfer on the SER_TX pin. If a UART write operation is pending, then the CPU will be blocked with wait states until the transfer is complete before starting the new write operation. This makes the UART transmit a relatively expensive operation on the CPU, but avoids the necessity of buffering data and checking for buffer overflow. Reading a value from this register returns 255 (0xff) if no valid data byte is in the receive buffer, and returns the value of the receive buffer otherwise, and clears the receive buffer for additional reads. Note that there is no FIFO associated with the UART.

Table 13 **`reg_uart_enable`**

<table>
<tr><td colspan="8">0x2000000b</td><td colspan="8">0x2000000a</td><td colspan="8">0x20000009</td><td colspan="8">0x20000008</td><td>address</td></tr>
<tr><td colspan="24">(unused, value is 0x0)</td><td colspan="8"></td><td>value</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>bit</td></tr>
</table>

The UART must be enabled to run (default disabled)

SPI Master spi sdi (pin E9), spi csb (pin E8), spi sck (pin F8), and spi sdo (pin F9)

Table 14 **`reg_spi_config`**

<table>
<tr><td colspan="8">0x24000003</td><td colspan="8">0x24000002</td><td colspan="8">0x24000001</td><td colspan="8">0x24000000</td><td>address</td></tr>
<tr><td colspan="16">(undefined, reads zero)</td><td colspan="16">SPI master configuration</td><td>value</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>bit</td></tr>
</table>

Configuration bit defintions

Bit 15	Housekeeping	0 = SPI master connected to external pins 1 = SPI master connected directly to housekeeping SPI
Bit 14	SPI interrupt enable	0 = interrupt disabled 1 = interrupt enabled
Bit 13	SPI system enable	0 = SPI disabled 1 = SPI enabled
Bit 12	stream	0 = apply/release CSB separately for each byte 1 = apply CSB until stream bit is cleared (manually)
Bit 11	mode	0 = read and change data on opposite SCK edges 1 = read and change data on the same SCK edge
Bit 10	invert SCK	0 = normal SCK 1= inverted SCK
Bit 9	invert CSB	0 = normal CSB (low is active) 1 = inverted CSB (high is active)
Bit 8	MLB	0 = msb first 1 = lsb first
Bits 7–0	prescaler	count (in master clock cycles) of 1/2 SCK cycle (default value 2)

All configuration bits other than the prescaler default to value zero.

Table 15 **`reg_spi_data`**

<table>
<tr><td colspan="8">0x24000007</td><td colspan="8">0x24000006</td><td colspan="8">0x24000005</td><td colspan="8">0x24000004</td><td>address</td></tr>
<tr><td colspan="24">(undefined, reads zero)</td><td colspan="8">SPI data</td><td>value</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>bit</td></tr>
</table>

The byte at 0x24000004 holds the SPI data (either read or write)

Reading to and writing from the SPI master is simply a matter of setting the required values in the configuration register, and writing values to or reading from reg_spi_data. The protocol is similar to the UART. A write operation will stall the CPU if an incomplete SPI transmission is still in progress. Reading from the SPI will also stall the CPU if an incomplete SPI transmission is still in progress. There is no FIFO buffer for data. Therefore SPI reads and writes are relatively expensive operations that tie up the CPU, but will not lose or overwrite data. Note that there is no FIFO associated with the SPI master.

Counter-Timer 0

The counter/timer is a general-purpose 32-bit adder and subtractor that can be configured for a variety of timing functions including one-shot counts, continuous timing, and interval interrupts. At a core clock rate of 80MHz, the longest single time interval is 26.84 seconds.

Table 16 **`reg_timer0_config`**

<table>
<tr><td colspan="8">0x22000003</td><td colspan="8">0x22000002</td><td colspan="8">0x22000001</td><td colspan="8">0x22000000</td><td>address</td></tr>
<tr><td colspan="24">(undefined, reads zero)</td><td colspan="8">Timer config</td><td>value</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>bit</td></tr>
</table>

Timer configuration bit definitions

Bit 3	Counter/timer enable	1 = counter/timer enabled 0 = counter/timer disabled
Bit 2	Oneshot mode	1 = oneshot mode 0 = continuous mode
Bit 1	Updown	1 = count up 0 = count down
Bit 0	Interrupt enable	1 = interrupt enabled 0 = interrupt disabled

Table 17 **`reg_timer0_value`**

<table>
<tr><td colspan="8">0x22000007</td><td colspan="8">0x22000006</td><td colspan="8">0x22000005</td><td colspan="8">0x22000004</td><td>address</td></tr>
<tr><td colspan="32">Timer value</td><td>value</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>bit</td></tr>
</table>

The value in this register is the current value of the counter. Value is 32 bits. The register is read-write and can be used to reset the timer.

Table 18 **`reg_timer0_data`**

<table>
<tr><td colspan="8">0x2200000b</td><td colspan="8">0x2200000a</td><td colspan="8">0x22000009</td><td colspan="8">0x22000008</td><td>address</td></tr>
<tr><td colspan="32">Timer data</td><td>value</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>bit</td></tr>
</table>

The value in this register is the reset value for the comparator.

When enabled, the counter counts up or down from the value set in reg_timer_value at the time the counter is enabled. If counting up, the count continues until the counter reaches reg_timer_data. If counting down, the count continues until the counter reaches zero.

In continuous mode, the counter resets to zero if counting up, and resets to the value in reg_timer_data if counting down, and the count continues immediately. If the interrupt is enabled, the counter will generate an interrupt on every cycle.

In one-shot mode, the counter triggers an interrupt (IRQ channel 10; see next page) when it reaches the value of reg_timer_data (up count) or zero (down count), and stops.

Note: When the counter/timer is disabled, the reg_timer_value remains unchanged, which puts the timer in a hold state. When re-enabled, counting resumes. To reset the timer, write zero to the reg_timer_value register.

Counter-Timer 1

The second counter/timer is functionally identical to the first, with different memory mapped addresses for the controls, as shown in the tables below.

Table 19 **`reg_timer1_config`**

<table>
<tr><td colspan="8"><code>0x23000003</code></td><td colspan="8"><code>0x23000002</code></td><td colspan="8"><code>0x23000001</code></td><td colspan="8"><code>0x23000000</code></td><td>address</td></tr>
<tr><td colspan="24">(undefined, reads zero)</td><td colspan="8">Timer config</td><td>value</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>bit</td></tr>
</table>

Timer configuration bit definitions

Bit 3	Counter/timer enable	1 = counter/timer enabled 0 = counter/timer disabled
Bit 2	Oneshot mode	1 = oneshot mode 0 = continuous mode
Bit 1	Updown	1 = count up 0 = count down
Bit 0	Interrupt enable	1 = interrupt enabled 0 = interrupt disabled

Table 20 **`reg_timer1_value`**

<table>
<tr><td colspan="8"><code>0x23000007</code></td><td colspan="8"><code>0x23000006</code></td><td colspan="8"><code>0x23000005</code></td><td colspan="8"><code>0x23000004</code></td><td>address</td></tr>
<tr><td colspan="32">Timer value</td><td>value</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>bit</td></tr>
</table>

The value in this register is the current value of the counter. Value is 32 bits. The register is read-write and can be used to reset the timer.

Table 21 **`reg_timer1_data`**

<table>
<tr><td colspan="8"><code>0x2300000b</code></td><td colspan="8"><code>0x2300000a</code></td><td colspan="8"><code>0x23000009</code></td><td colspan="8"><code>0x23000008</code></td><td>address</td></tr>
<tr><td colspan="32">Timer data</td><td>value</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>bit</td></tr>
</table>

The value in this register is the reset value for the comparator.

When enabled, the counter counts up or down from the value set in reg_timer_value at the time the counter is enabled. If counting up, the count continues until the counter reaches reg_timer_data. If counting down, the count continues until the counter reaches zero.

In continuous mode, the counter resets to zero if counting up, and resets to the value in reg_timer_data if counting down, and the count continues immediately. If the interrupt is enabled, the counter will generate an interrupt on every cycle.

In one-shot mode, the counter triggers an interrupt (IRQ channel 11; see next page) when it reaches the value of reg_timer_data (up count) or zero (down count), and stops.

Note: When the counter/timer is disabled, the reg_timer_value remains unchanged, which puts the timer in a hold state. When re-enabled, counting resumes. To reset the timer, write zero to the reg_timer_value register.

Interrupts (IRQ)

The interrupt vector is set to memory addres 0 (bottom of SRAM). The program counter switches to this location when an interrupt is received. To enable interrupts, it is necessary to copy an interrupt handler to memory location 0. The PicoRV32 defines 32 IRQ channels, of which the Caravel chip uses only a handful, as described in the table below. All IRQ channels not in the list below always have value zero.

Table 19 CPU IRQ channel definitions

IRQ channel	*description*
4	UART data available
5	IRQ external pin (pin E5)
6	Housekeeping SPI IRQ
7	Assignable interrupt (see Table 7)
9	SPI master data available, when enabled (see Table 14)
10	Timer 0 expired, when enabled (see Table 16)
11	Timer 1 expired, when enabled (see Table 19)

The Caravel PicoRV32 implementation does not enable IRQ QREGS (see PicoRV32 description).

The handling of interrupts is beyond the scope of this document (see RISC-V instruction set description). All interrupts are masked and must be enabled in software.

Management area SRAM

The Caravel chip has an on-board memory of 256 words of width 32 bits. The memory is located at address 0 (zero). There are additional blocks of memory above this area, size and location TBD.

Storage area SRAM

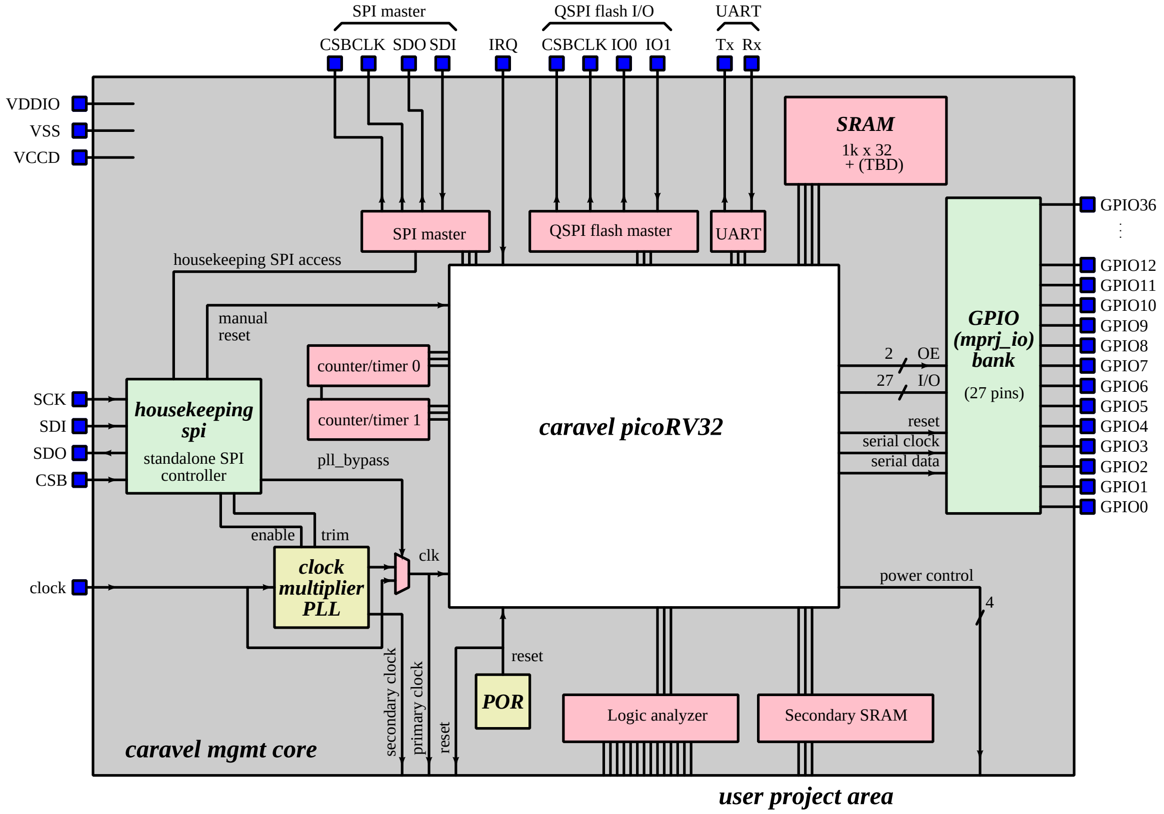
The Caravel chip has a “storage area” SRAM block that is auxiliary space that can be used by either the management SoC or the user project, through the wishbone bus interface. The storage area is connected into the user area 2 power supply, and so is nominally considered to be part of the user area.

The storage area may be used as an experimentation area for OpenRAM, so for any user project making use of this space, the user should notify efabless of their requirement for the size and configuration of the SRAM block.

Logic Analyzer

User area wishbone base

Caravel management SoC simplified block diagram



Programming

The RISC-V architecture has a **gcc** compiler. The best reference for getting the correct cross-compiler version is the PicoRV32 source at

`http://github.com/cliffordwolf/picorv32`.

Specifically, see the top-level **`README.md`** file section “Building a pure RV32I Toolchain.”

For programming examples specifically for the Caravel chip (assuming a correct installation of a RISC-V gcc toolchain as described above), see

`http://github.com/efabless/caravel`

The directory **`verilog/dv`** contains example source code to program the Ravenna chip along with the header file **`defs.h`** that defines the memory-mapped locations as described throughout this text.

The **`verilog/dv`** directory contains a **`Makefile`** that compiles hex files and runs simulations of a number of test programs that exercise various features of the chip.

Additional documentation exists on the same site for the provided demonstration circuit board and driver software.

Additional references

See **`http://riscv.org/`**
`http://riscv.org/software-status/`

Memory Mapped I/O summary by address

Address (bytes)	Function
0x00 00 00 00	Flash SPI / overlaid SRAM (4k words) start of memory block
0x00 00 3f ff	End of SRAM
0x10 00 00 00	Flash SPI start of program block Program to run starts here on reset.
0x10 ff ff ff	Maximum SPI flash addressable space (16MB) with QSPI 3-byte addressing
0x1f ff ff ff	Maximum SPI flash addressable space (32MB)
0x20 00 00 00	UART clock divider select (system clock freq. / baud rate)
0x20 00 00 04	UART data (returns `0xffffffff` if receiver buffer is empty)
0x20 00 00 08	UART enable
0x21 00 00 00	GPIO input/output (bit 16/bit 0) 1 general-purpose digital, management area only
0x21 00 00 04	GPIO output enable (1 = output, 0 = input)
0x21 00 00 08	GPIO pullup enable (1 = pullup, 0 = none)
0x21 00 00 0c	GPIO pulldown enable (1 = pulldown, 0 = none)
0x22 00 00 00	Counter/Timer 0 configuration register (lower 4 bits) bit 0 = enable (0 = hold, 1 = count) bit 1 = oneshot (0 = continuous count, 1 = one-shot count) bit 2 = updown (0 = count down, 1 = count up) bit 3 = irq enable (0 = disabled, 1 = trigger IRQ channel 10 on timeout)
0x22 00 00 04	Counter/Timer 0 current value Set or read the 32-bit current value.
0x22 00 00 08	Counter/Timer 0 reset value Set or read the 32-bit reset (down-count) or compare (up-count) value.
0x23 00 00 00	Counter/Timer 1 configuration register (lower 4 bits) bit 0 = enable (0 = hold, 1 = count) bit 1 = oneshot (0 = continuous count, 1 = one-shot count) bit 2 = updown (0 = count down, 1 = count up) bit 3 = irq enable (0 = disabled, 1 = trigger IRQ channel 11 on timeout)
0x23 00 00 04	Counter/Timer 1 current value Set or read the 32-bit current value.
0x23 00 00 08	Counter/Timer 1 reset value Set or read the 32-bit reset (down-count) or compare (up-count) value.
0x24 00 00 00	SPI master configuration register bits 0–7 = prescaler (core clock / (prescaler + 1) = SPI clock rate / 2) (default 2) bit 8 = mlb (0 = msb first, 1 = lsb first) (default 0) bit 9 = invcsb (0 = csb active low, 1 = csb active high) (default 0) bit 10 = invsck (0 = normal sck, 1 = inverted sck) (default 0) bit 11 = mode (0 = read/write on opposite sck edge, 1 = same edge) (default 0) bit 12 = stream (0 = raise csb after each byte, 1 = keep csb low until stream bit cleared) bit 13 = enable (0 = SPI master disabled, 1 = SPI master enabled) bit 14 = irq enable (0 = disabled, 1 = SPI read valid triggers interrupt channel 9) bit 15 = housekeeping (0 = disconnected, 1 = connected)
0x24 00 00 04	SPI master data register (low 8 bits) Write data to send to low byte or read received data from low byte.

Memory Mapped I/O summary by address *(continued)*

Address (bytes)	Function
0x25 00 00 00	Logic Analyzer Data 0
0x25 00 00 04	Logic Analyzer Data 1
0x25 00 00 08	Logic Analyzer Data 2
0x25 00 00 0c	Logic Analyzer Data 3
0x25 00 00 10	Logic Analyzer Enable 0
0x25 00 00 14	Logic Analyzer Enable 1
0x25 00 00 18	Logic Analyzer Enable 2
0x25 00 00 1c	Logic Analyzer Enable 3
0x26 00 00 00	User project area GPIO data (L)
0x26 00 00 04	User project area GPIO data (H)
0x26 00 00 08	User project area GPIO data transfer (bit 0, auto-zeroing)
0x26 00 00 0c	User project area GPIO mprj_io[0] configure
⋮	
0x26 00 00 a0	User project area GPIO mprj_io[37] configure

bit 0 = management control enable (0 = user control, 1 = management control) (default 1)
bit 1 = output disable (0 = output enabled, 1 = output disabled) (default 1)
bit 2 = hold override value (value = value during hold mode) (default 0)
bit 3 = input disable (0 = input enabled, 1 = input disabled) (default 0)
bit 4 = IB mode select (0 = , 1 =)
bit 5 = analog bus enable (0 = disabled, 1 = enabled)
bit 6 = analog bus select (0 = , 1 =)
bit 7 = analog bus polarity (0 = , 1 =)
bit 8 = slow slew (0 = fast slew, 1 = slow slew) (default 0)
bit 9 = input voltage trip point select (0 = , 1 =)
bits 10–12 = digital mode (see below) (default 001)

Digital mode bits bit 12 11 10	Digital mode description
0 0 0	
0 0 1	
0 1 0	
0 1 1	
1 0 0	
1 0 1	
1 1 0	
1 1 1	

Address (bytes)	Function
0x26 00 00 a4	User project area GPIO power[0] configure
0x26 00 01 b4	User project area GPIO power[3] configure

These are currently undefined/unused.

Address (bytes)	Function
0x2d 00 00 00	QSPI controller config

bit 31 MEMIO enable (reset = 1) 0 = bit-bang mode
bit 22 DDR enable
bit 21 QSPI enable
bit 20 CRM enable
bits 19-16 Read latency cycles
bits 11-8 I/O output enable bits (bit bang mode)
bit 5 Chip select line (bit bang mode)
bit 4 Serial clock line (bit bang mode)
bits 3-0 Data bits (bit bang mode)

Note: These cannot be used due to the limited number of data pins.

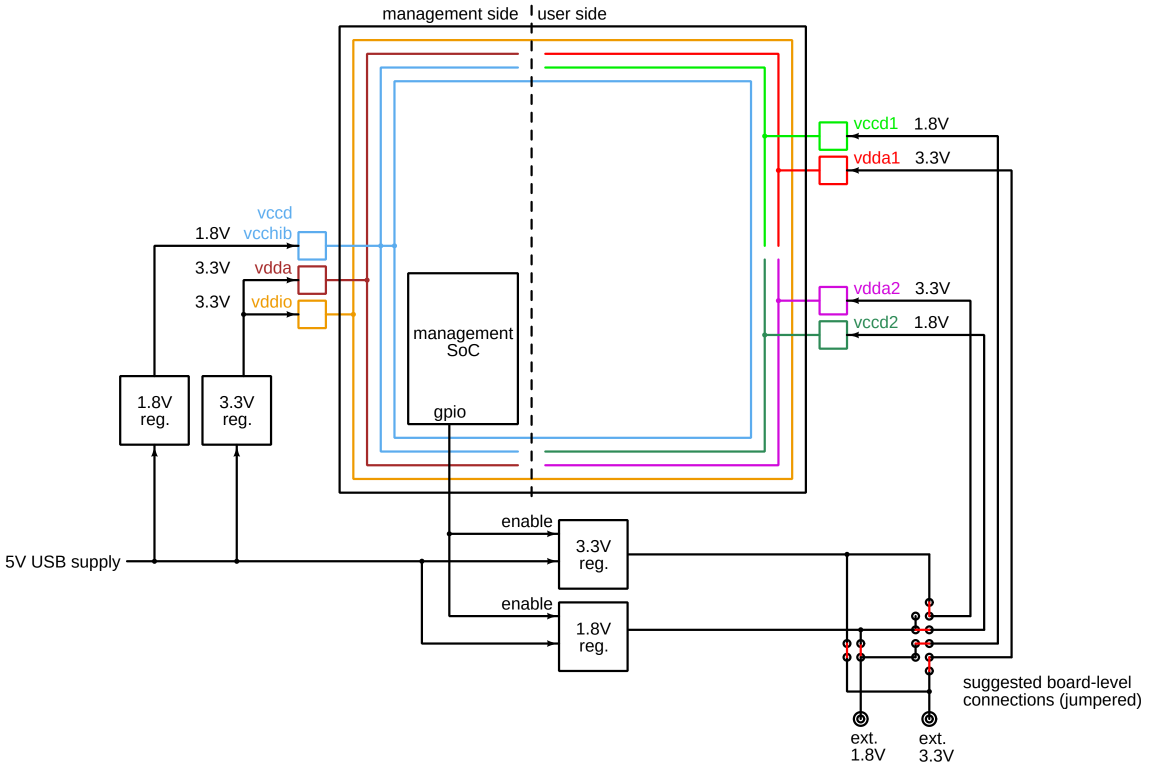
Memory Mapped I/O summary by address *(continued)*

Address (bytes)	Function	
0x2f 00 00 00	PLL clock output destination (low bit) 0 = none 1 = GPIO	The PLL clock (crystal oscillator clock multiplied up by PLL) can be viewed on the GPIO pin. The GPIO pin cannot be used as general-purpose I/O when selected for PLL clock output. It is unlikely that a full-speed (100MHz) clock will be able to toggle the GPIO at full swing, but is detectable.
0x2f 00 00 04	Trap output destination (low bit) 0 = none 1 = GPIO	The CPU fault state (trap) can be viewed at the GPIO pin as a way to monitor the CPU trap state externally.
0x2f 00 00 08	IRQ 7 input source (low bit) 0 = none 1 = GPIO	The GPIO input can be used as an IRQ event source and passed to the CPU through IRQ channel 7. When used as an IRQ source, the GPIO pin must be configured as an input.
0x30 00 00 00	User area base	A user project may define additional wishbone slave modules starting at this address.
0x80 00 00 00	QSPI controller	
0x90 00 00 00	Storage area SRAM	
0xa0 00 00 00	Any slave 1	
0xb0 00 00 00	Any slave 2	

Supplementary material (to be incorporated into the documentation text):

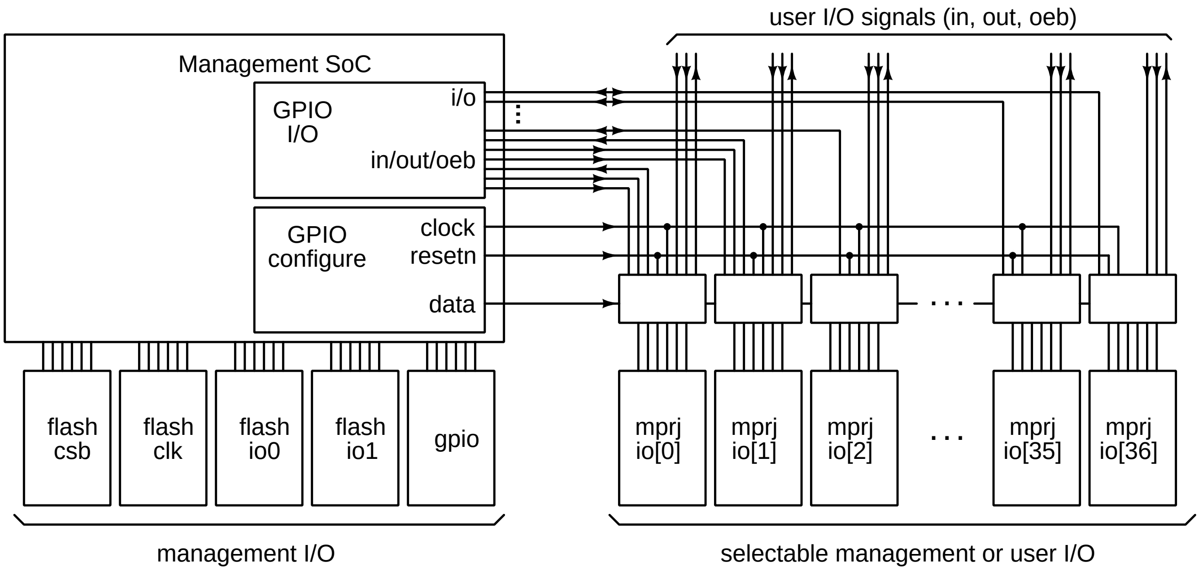
“Caravel” harness chip

power domain splits



“Caravel” harness chip

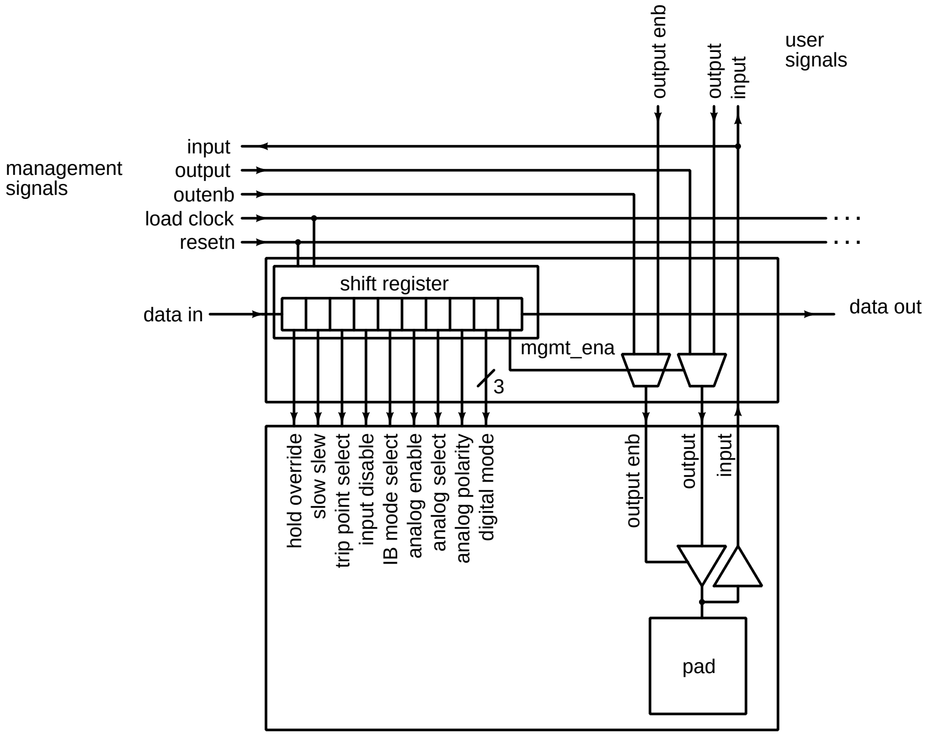
GPIO pads



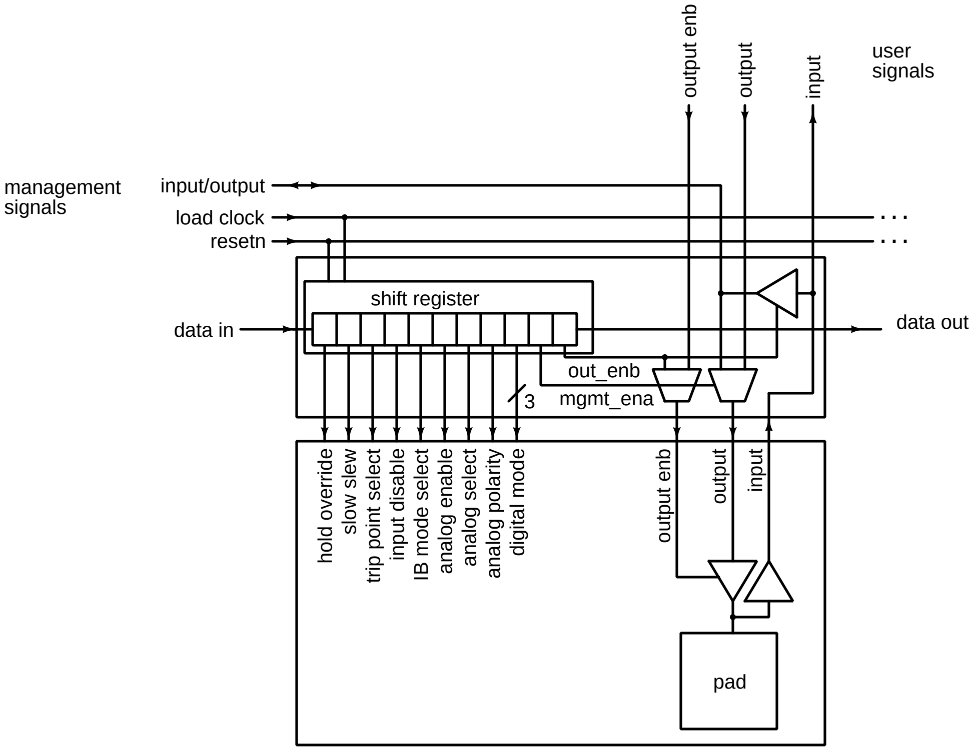
Supplementary material (to be incorporated into the documentation text):

Single GPIO pad structure

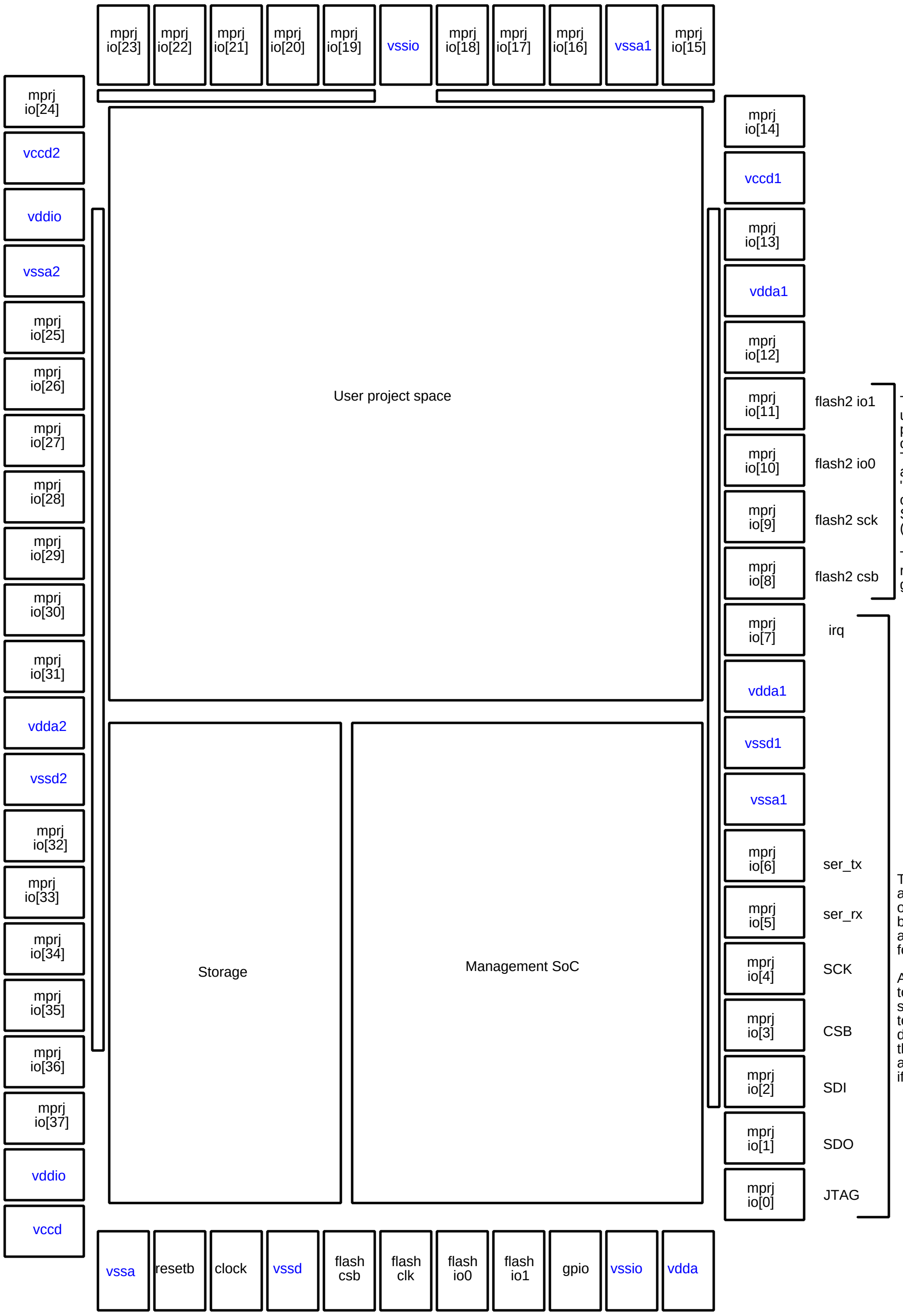
Used for pad 0 (JTAG) and pad 1 (SDO)



Single GPIO pad structure
Used for all pads except 0 and 1



Supplementary material (to be incorporated into the documentation text):



These pins can be used for a user project that has its own flash memory. They can be accessed with the "pass-thru" mode of the housekeeping SPI for flash (re)programming.

The user may repurpose these for general-purpose I/O.

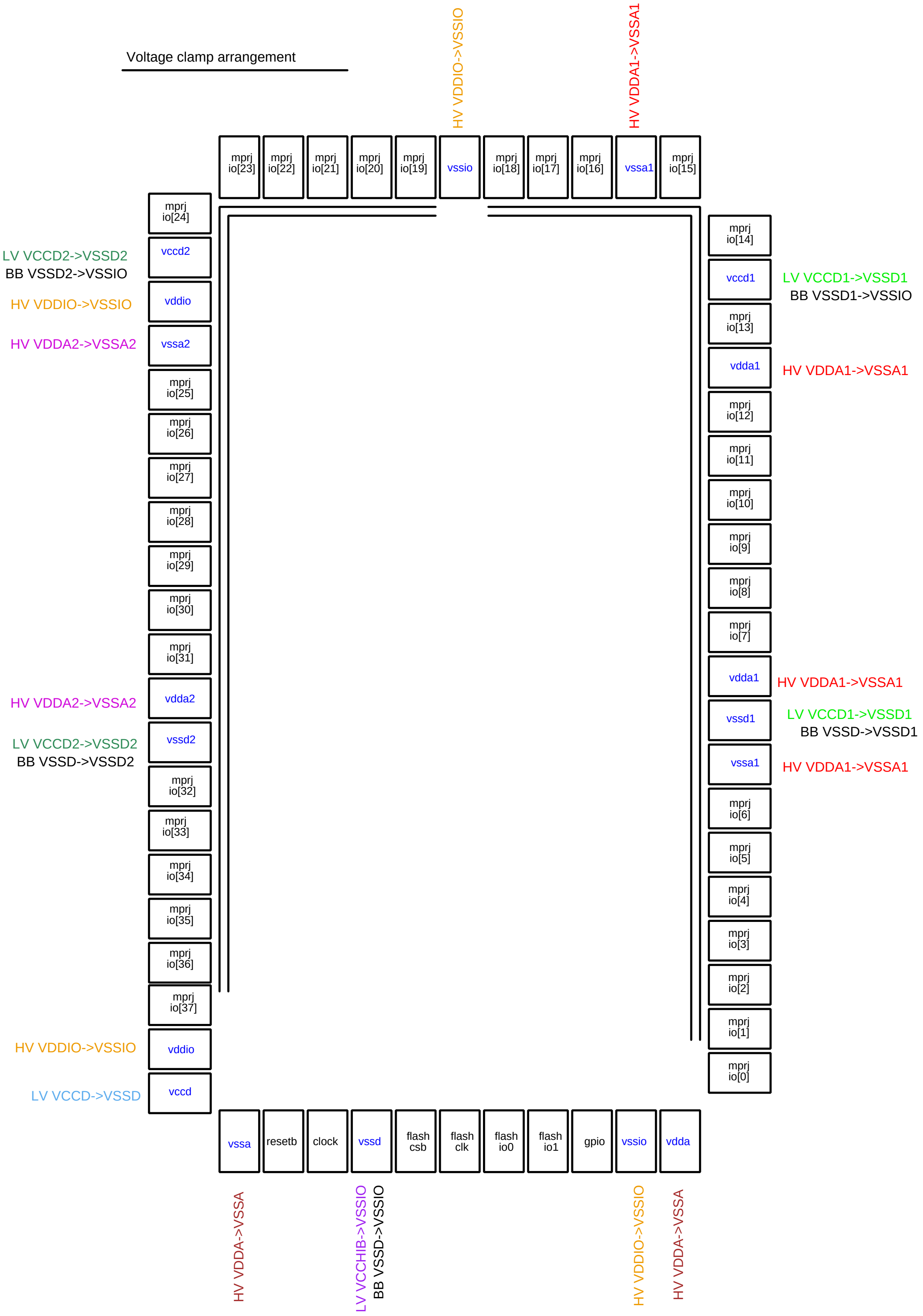
These pins have a dedicated function on startup, but can be programmed to any use by the user for the user project.

All connections are to the FTDI chip and should be jumpered to allow them to be disconnected from the FTDI and available to the user if needed.

3.2 x 5.3 mm, 62 pins (20, 20, 11, and 11)

Supplementary material (to be incorporated into the documentation text):

Voltage clamp arrangement



Clamps needed

HV VDDIO * 4
HV VDDA * 2
HV VDDA1 * 2
HV VDDA2 * 2

10 pads w/HV

LV VCCD * 2
LV VCCD1 * 2
LV VCCD2 * 2
LV VCCHIB * 2

BB VSSD, VSSD1
BB VSSD, VSSD2

BB VSSD1, VSSIO
BB VSSD2, VSSIO

4 pads w/LV

Redundant power pins:

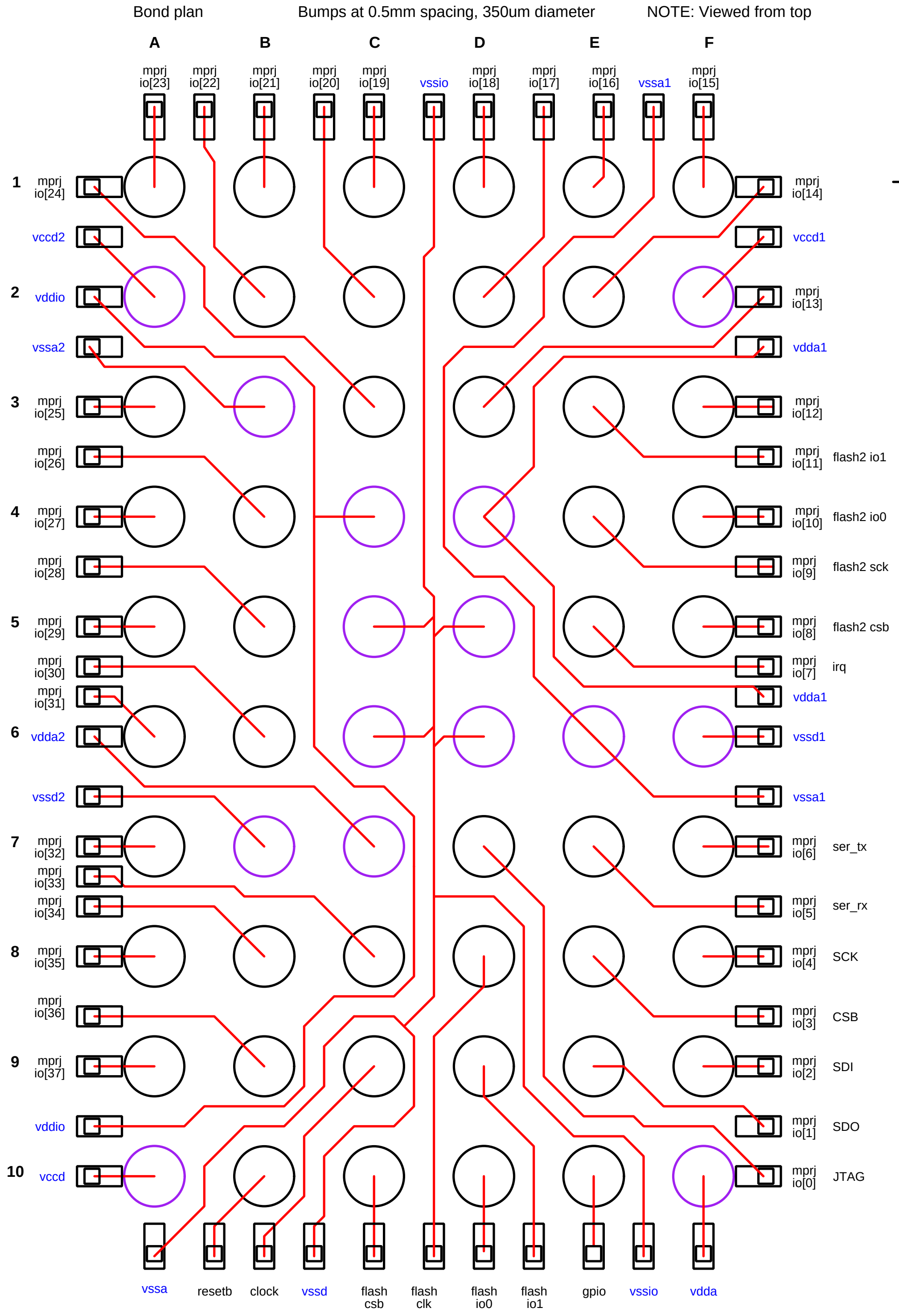
VDDIO x2
VSSIO x2

(VDDIO domain powers all output drivers)

VDDA1 x2
VSSA1 x2

(High current user applications can use this domain)

Supplementary material (to be incorporated into the documentation text):



45 signal pins

14 unique power pins

4 redundant power pins

---

59 pins

60 pad positions (6 x 10 array)

Unique power domains:

vddio	vssio
vdda	vssa
vccd	vssd
vdda1	vssa1
vccd1	vssd1
vdda2	vssa2
vccd2	vssd2

Combine these into one net and route to center bumps

Total: 56 pins

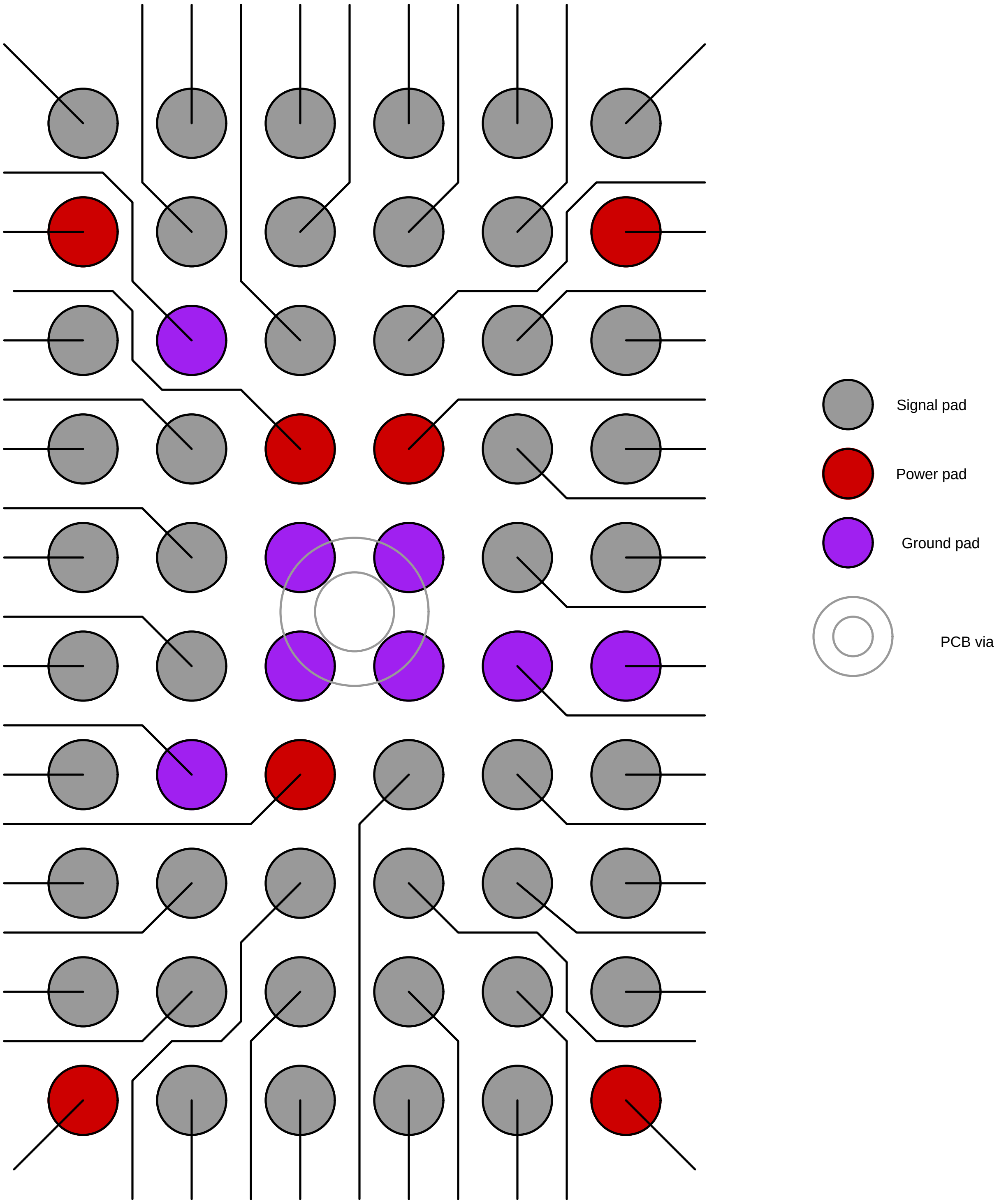
Pinout

A1	mprj_io[23]	D1	mprj_io[18]
A2	vccd2	D2	mprj_io[17]
A3	mprj_io[25]	D3	mprj_io[13]
A4	mprj_io[27]	D4	vdda1
A5	mprj_io[29]	D5	vssio/vssa/vssd
A6	mprj_io[31]	D6	vssio/vssa/vssd
A7	mprj_io[32]	D7	mprj_io[0]/JTAG
A8	mprj_io[35]	D8	flash clk
A9	mprj_io[37]	D9	flash io1
A10	vccd	D10	flash io0
B1	mprj_io[21]	E1	mprj_io[16]
B2	mprj_io[22]	E2	mprj_io[14]
B3	vssa2	E3	mprj_io[11]/flash2 io1
B4	mprj_io[26]	E4	mprj_io[9]/flash2 sck
B5	mprj_io[28]	E5	mprj_io[7]/irq
B6	mprj_io[30]	E6	vssa1
B7	vssd2	E7	mprj_io[5]/ser_rx
B8	mprj_io[34]	E8	mprj_io[3]/CSB
B9	mprj_io[36]	E9	mprj_io[1]/SDO
B10	resetb	E10	gpio
C1	mprj_io[19]	F1	mprj_io[15]
C2	mprj_io[20]	F2	vccd1
C3	mprj_io[24]	F3	mprj_io[12]
C4	vddio	F4	mprj_io[10]/flash2 io0
C5	vssio/vssa/vssd	F5	mprj_io[8]/flash2 csb
C6	vssio/vssa/vssd	F6	vssd1
C7	vdda2	F7	mprj_io[6]/ser_tx
C8	mprj_io[33]	F8	mprj_io[4]/SCK
C9	clock	F9	mprj_io[2]/SDI
C10	flash csb	F10	vdda

Supplementary material (to be incorporated into the documentation text):

PCB example route pattern

Via in center connects center ground pads



	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	°C
V_{OH}	0.8 · VDDIO			V
V_{OL}			0.4	V
Management area power		TBD		mW
Storage area power		TBD		mW

Errata

Known errors in the efabless Caravel harness version 1:

There are no known errors in Caravel version 1 at this time.

Documentation errata:

There are no known errors in the Caravel documentation at this time.