# Table of contents

1 Introduction ......................................................... 3  
  1.1 Pinout Diagrams ................................................. 3  
  1.2 Detailed overview ............................................... 6  
  1.3 Board components location  
     1.3.1 Front components location ................................ 6  
     1.3.2 Back components location  
  2 Quick Start ........................................................ 9  
  2.1 What’s included in the box? ....................................... 9  
  2.2 Unboxing ............................................................ 9  
  2.3 Tethering to PC .................................................... 9  
  2.4 Flashing eMMC  
     2.4.1 Flash the latest image on eMMC  
  2.5 Access UART debug console ...................................... 9  
  2.6 Demos and Tutorials ............................................... 11  
3 Design & specifications ............................................. 13  
  3.1 Block diagram ..................................................... 13  
  3.2 System on Chip (SoC) .............................................. 14  
  3.3 Power management ................................................ 14  
  3.4 General Connectivity and Expansion  
     3.4.1 USB-C port ..................................................... 14  
     3.4.2 P8 & P9 cape header pins .................................... 14  
     3.4.3 ADC ............................................................... 14  
  3.5 Buttons and LEDs  
     3.5.1 User LEDs and Power LED ..................................... 14  
     3.5.2 User and reset button ....................................... 14  
  3.6 Connectivity ......................................................... 14  
     3.6.1 Ethernet ......................................................... 14  
  3.7 Memory, Media and Data storage  
     3.7.1 DDR memory .................................................... 17  
     3.7.2 eMMC .............................................................. 17  
     3.7.3 microSD .......................................................... 17  
     3.7.4 EEPROM .......................................................... 17  
     3.7.5 SPI flash .......................................................... 17  
  3.8 Multimedia I/O ...................................................... 17  
     3.8.1 CSI ................................................................. 17  
  3.9 Debug  
     3.9.1 UART debug port ............................................. 17  
     3.9.2 JTAG debug port .............................................. 17  
  3.10 Mechanical Specifications ...................................... 21  
4 Expansion .............................................................. 29  
  4.1 Cape Headers ....................................................... 29  
    4.1.1 Connector P8 ................................................... 29  
    4.1.2 Connector P9 ................................................... 33  
5 Demos .................................................................. 37
5.1 Upgrade BeagleV-Fire Gateware ............................................. 37
  5.1.1 Required Equipment ..................................................... 37
  5.1.2 Connect to BeagleV-Fire Linux Command Line Interface ............... 37
  5.1.3 Gateware Upgrade Linux Commands ..................................... 38
5.2 Flashing gateware and Linux image ........................................ 39
  5.2.1 Programming & Debug tools installation ................................ 39
  5.2.2 Flashing gateware image ................................................ 40
  5.2.3 Flashing eMMC ............................................................ 42
5.3 Microchip FPGA Tools Installation Guide .................................... 44
  5.3.1 Install Libero 2022.3 ...................................................... 44
  5.3.2 Install SoftConsole 2022.2 ................................................ 46
  5.3.3 Install the Libero licensing daemon ...................................... 47
  5.3.4 Request a Libero Silver license .......................................... 47
  5.3.5 Execute tool setup script ................................................ 47
5.4 Gateware Design Introduction ................................................. 48
  5.4.1 Gateware Architecture ..................................................... 49
5.5 How to retrieve BeagleV-Fire’s gateware version ............................. 50
  5.5.1 Device Tree ................................................................. 50
  5.5.2 Bootloader messages ....................................................... 51
5.6 Gateware Full Build Flow ...................................................... 51
  5.6.1 Introduction ................................................................. 51
  5.6.2 Programming artifacts ..................................................... 52
  5.6.3 Programming BeagleV-Fire with new gateware ........................... 53
5.7 Gateware TCL Scripts Structure ............................................... 53
  5.7.1 Gateware Project ........................................................... 53
  5.7.2 Gateware Components ..................................................... 54
  5.7.3 Gateware Build Options .................................................. 54
  5.7.4 Gateware Component Directories ........................................ 55
  5.7.5 Opening the gateware as a libero project ................................ 55
5.8 Customize BeagleV-Fire Cape Gateware Using Verilog ....................... 56
  5.8.1 Fork BeagleV-Fire Gateware Repository ................................ 57
  5.8.2 Create A Custom Gateware Build Option ................................ 57
  5.8.3 Rename A Copy Of The Cape Gateware Verilog Template ................ 58
  5.8.4 Customize The Cape’s Verilog Source Code ................................ 58
  5.8.5 Commit And Push Changes To Your Forked Repository .................. 61
  5.8.6 Retrieve The Forked Repositories Artifacts .............................. 62
  5.8.7 Program BeagleV-Fire With Your Custom Bitstream ...................... 62
5.9 Exploring Gateware Design with Libero ...................................... 63
  5.9.1 Prerequisites ............................................................... 63
  5.9.2 Cloning and Building the Gateware ...................................... 63
  5.9.3 Exploring The Design ...................................................... 64
  5.9.4 Adding Custom HDL ....................................................... 64
  5.9.5 Exporting The Design ..................................................... 66
6 Support ................................................................................. 73
  6.1 Production board boot media ................................................. 73
  6.2 Certifications and export control ............................................ 73
    6.2.1 Export designations ....................................................... 73
    6.2.2 Size and weight ........................................................... 73
  6.3 Additional documentation .................................................... 73
    6.3.1 Hardware docs ............................................................. 73
    6.3.2 Software docs .............................................................. 73
    6.3.3 Support forum .............................................................. 74
    6.3.4 Pictures .................................................................. 74
  6.4 Change History ............................................................... 74
    6.4.1 Board Changes ............................................................. 74
BeagleV®-Fire is a revolutionary SBC powered by the Microchip’s PolarFire® MPFS025T RISC-V System on Chip (SoC) with FPGA fabric. BeagleV®-Fire opens up new horizons for developers, tinkerers, and the open-source community to explore the vast potential of RISC-V architecture and FPGA technology. It has the same P8 & P9 cape header pins as BeagleBone Black allowing you to stack your favorite BeagleBone cape on top to expand it’s capability. Built around the powerful and energy-efficient RISC-V instruction set architecture (ISA) along with its versatile FPGA fabric, BeagleV®-Fire SBC offers unparalleled opportunities for developers, hobbyists, and researchers to explore and experiment with RISC-V technology.
Chapter 1

Introduction

BeagleV®-Fire is a revolutionary SBC powered by the Microchip’s PolarFire® MPFS025T System on Chip (SoC) with 4x RV64GC Application cores, 1x RV64IMAC monitor/boot core, and FPGA fabric. BeagleV®-Fire opens up new horizons for developers, tinkerers, and the open-source community to explore the vast potential of RISC-V architecture and FPGA technology. It has the same P8 & P9 cape header pins as BeagleBone Black allowing you to stack your favourite BeagleBone cape on top to expand it’s capability. Built around the powerful and energy-efficient RISC-V instruction set architecture (ISA) along with its versatile FPGA fabric, BeagleV®-Fire SBC offers unparalleled opportunities for developers, hobbyists, and researchers to explore and experiment with RISC-V technology.

1.1 Pinout Diagrams

Choose the cape header to see respective pinout diagram.
P8 cape header
P9 cape header
Fig. 1.1: BeagleV-Fire P8 cape header pinout
Fig. 1.2: BeagleV-Fire P9 cape header pinout
1.2 Detailed overview

Table 1.1: BeagleV-Fire features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>MPPS025T-FCVG484E</td>
</tr>
<tr>
<td>Memory</td>
<td>2GB (1Gb x 16)- 1866MHz 3733Mbps, LPDDR4</td>
</tr>
<tr>
<td>Storage</td>
<td>Kingston 16GB eMMC</td>
</tr>
<tr>
<td>Wireless</td>
<td>1x M.2 Key E, support 2.4GHz/5GHz WiFi module</td>
</tr>
<tr>
<td>Ethernet</td>
<td>• PHY: Realtek RTL8211F-VD-CG Gigabit Ethernet phy</td>
</tr>
<tr>
<td></td>
<td>• Connector: integrated magnetics RJ-45</td>
</tr>
<tr>
<td>USB C</td>
<td>• Connectivity: Flash/programming support</td>
</tr>
<tr>
<td></td>
<td>• Power: Input: 5V @ 3A</td>
</tr>
<tr>
<td>Other connectors</td>
<td>• 1x SYZYGY High speed connector</td>
</tr>
<tr>
<td></td>
<td>• microSD card slot</td>
</tr>
<tr>
<td></td>
<td>• CSI connector compatible with BeagleBone AI-64, BeagleV-Ahead, Raspberry Pi Zero / CM4 (22-pin)</td>
</tr>
</tbody>
</table>

1.3 Board components location

This section describes the key components on the board, their location and function.

1.3.1 Front components location

![BeagleV-Fire board front components location](image)

Fig. 1.3: BeagleV-Fire board front components location
Table 1.2: BeagleV-Fire board front components location

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power LED</td>
<td>Power (Board ON) indicator</td>
</tr>
<tr>
<td>JTAG (MPFS025T)</td>
<td>MPFS025T SoC JTAG debug port</td>
</tr>
<tr>
<td>RTL8211F</td>
<td>Gigabit IEEE 802.11 Ethernet PHY</td>
</tr>
<tr>
<td>P8 &amp; P9 cape header</td>
<td>Expansion headers for BeagleBone capes.</td>
</tr>
<tr>
<td>2GB RAM</td>
<td>2GB (1Gb x 16): 1866MHz 3773Mbps, LPDDR4</td>
</tr>
<tr>
<td>16GB eMMC</td>
<td>Kingston 16GB eMMC Flash storage</td>
</tr>
<tr>
<td>CSI</td>
<td>22pin MIPI Camera connectors</td>
</tr>
<tr>
<td>M.2 Key E</td>
<td>PCIe M.2 Key E connector</td>
</tr>
<tr>
<td>UART debug header</td>
<td>6 pin UART debug header</td>
</tr>
<tr>
<td>Reset button</td>
<td>Press to reset BeagleV-Fire board (MPFS025T SoC)</td>
</tr>
<tr>
<td>User button</td>
<td>User defined (custom) action button</td>
</tr>
<tr>
<td>User LEDs</td>
<td>12x user programmable LEDs to show various board status during boot.</td>
</tr>
<tr>
<td>GigaBit Ethernet</td>
<td>1Gb/s Wired internet connectivity</td>
</tr>
<tr>
<td>Barrel jack</td>
<td>Power input</td>
</tr>
<tr>
<td>USB C</td>
<td>Power, connectivity, and board flashing.</td>
</tr>
</tbody>
</table>

1.3.2 Back components location

![BeagleV-Fire board back components location](image)

Fig. 1.4: BeagleV-Fire board back components location

Table 1.3: BeagleV-Fire board back components location

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>microSD</td>
<td>microSD card slot</td>
</tr>
<tr>
<td>SYZYGY</td>
<td>SYZYGY High speed connector</td>
</tr>
</tbody>
</table>
Chapter 2

Quick Start

2.1 What’s included in the box?

When you purchase a brand new BeagleV-Fire, in the box you’ll get:

1. BeagleV-Fire board
2. Quick-start card

Tip: For board files, 3D model, and more, you can checkout BeagleV-Fire repository on OpenBeagle.

2.2 Unboxing

2.3 Tethering to PC

To connect BeagleV-Fire board to PC via USB Type C receptacle you need a USB type C cable. Connection guide for the same is shown below:

Tip: To get a USB type C cable you can checkout links below:
1. USB C cable 0.3m (mouser)
2. USB C cable 1.83m (digkey)

2.4 Flashing eMMC

2.4.1 Flash the latest image on eMMC

2.5 Access UART debug console

Note: Some tested devices that are working good includes:
1. Adafruit CP2102N Friend - USB to Serial Converter
2. Raspberry Pi Debug Probe Kit for Pico and RP2040
Fig. 2.1: https://youtu.be/5clyv1R-1mc

Fig. 2.2: BeagleV-Fire tethered connection
To access a BeagleV-Fire serial debug console you can connect a USB to UART to your board as shown below:

![BeagleV-Fire UART debug port connection](image)

Fig. 2.3: BeagleV-Fire UART debug port connection

To see the board boot log and access your BeagleV-Fire’s console you can use application like \texttt{tio} to access the console. If you are using Linux your USB to UART converter may appear as \texttt{/dev/ttyUSB}. It will be different for Mac and Windows operating systems. To find serial port for your system you can checkout this guide.

```
[lorforlinux@fedora ~] $ tio /dev/ttyUSB0
	tio v2.5
Press ctrl-t q to quit
Connected
```

### 2.6 Demos and Tutorials

- \textit{How to retrieve BeagleV-Fire’s gateware version}
- \textit{Upgrade BeagleV-Fire Gateware}
- \textit{Flashing gateware and Linux image}
- \textit{Gateware Design Introduction}
- \textit{Microchip FPGA Tools Installation Guide}
Chapter 3

Design & specifications

If you want to know how BeagleV-Fire board is designed and what are its high-level specifications then this chapter is for you. We are going to discuss each hardware design element in detail and provide high-level device specifications in a short and crisp form as well.

Tip: For hardware design files and schematic diagram you can checkout BeagleV-Fire GitLab repository: https://git.beagleboard.org/beaglev-fire/beaglev-fire

3.1 Block diagram

Fig. 3.1: System block diagram
Fig. 3.2: Power tree diagram

3.2 System on Chip (SoC)

3.3 Power management

3.4 General Connectivity and Expansion

3.4.1 USB-C port

3.4.2 P8 & P9 cape header pins

3.4.3 ADC

3.5 Buttons and LEDs

3.5.1 User LEDs and Power LED

3.5.2 User and reset button

3.6 Connectivity

3.6.1 Ethernet
3.6. Connectivity

Fig. 3.3: I2C tree diagram

Fig. 3.4: SoC bank0
Chapter 3. Design & specifications
3.7 Memory, Media and Data storage

3.7.1 DDR memory

3.7.2 eMMC

3.7.3 microSD

3.7.4 EEPROM

3.7.5 SPI flash

3.8 Multimedia I/O

3.8.1 CSI

3.9 Debug

3.9.1 UART debug port

3.9.2 JTAG debug port
Fig. 3.9: SoC power

Fig. 3.10: DC 5V input
Fig. 3.11: Ideal diode

Fig. 3.12: VCC 1V0

Fig. 3.13: VCC 1V1

3.9. Debug
Fig. 3.14: VCC 1V8

Fig. 3.15: VCC 2V5

Fig. 3.16: VCC 3V3
3.10 Mechanical Specifications

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>86.38 * 54.61 * 18.8 mm</td>
</tr>
<tr>
<td>Max height</td>
<td>18.8 mm</td>
</tr>
<tr>
<td>PCB Size</td>
<td>86.38 * 54.6 mm</td>
</tr>
<tr>
<td>PCB Layers</td>
<td>12 Layers</td>
</tr>
<tr>
<td>PCB Thickness</td>
<td>1.6 mm</td>
</tr>
<tr>
<td>RoHS compliant</td>
<td>Yes</td>
</tr>
<tr>
<td>Gross Weight</td>
<td>106 g</td>
</tr>
<tr>
<td>Net weight</td>
<td>45.8 g</td>
</tr>
</tbody>
</table>
Fig. 3.19: USB C

Fig. 3.20: P8 cape header
3.10. Mechanical Specifications
Fig. 3.24: ADC LDO power supply

Fig. 3.25: User LEDs and power LED

Fig. 3.26: User button
3.10. Mechanical Specifications
Fig. 3.29: LPDDR memory

Fig. 3.30: EMMC flash storage
3.10. Mechanical Specifications
Fig. 3.34: CSI

Fig. 3.35: UART debug header

Fig. 3.36: JTAG debug header
Chapter 4

Expansion

Work in progress

4.1 Cape Headers

The expansion interface on the board is comprised of two headers P8 (46 pin) & P9 (46 pin). All signals on the expansion headers are 3.3V unless otherwise indicated.

**Note:** Do not connect 5V logic level signals to these pins or the board will be damaged.

**Note:** DO NOT APPLY VOLTAGE TO ANY I/O PIN WHEN POWER IS NOT SUPPLIED TO THE BOARD. IT WILL DAMAGE THE PROCESSOR AND VOID THE WARRANTY.

**NO PINS ARE TO BE DRIVEN UNTIL AFTER THE SYS_RESET LINE GOES HIGH.**

4.1.1 Connector P8

The following tables show the pinout of the P8 expansion header. The gateware image is responsible for setting the function of each pin. Refer to the processor documentation for more information on these pins and detailed descriptions of all of the pins listed. In some cases there may not be enough signals to complete a group of signals that may be required to implement a total interface.

The column heading is the pin number on the expansion header.

The **Name** row is the pin name on the processor.

The **BALL** row is the pin number on the processor.

The rows below **BALL** are the gateware setting for each pin.

**NOTES:**

**DO NOT APPLY VOLTAGE TO ANY I/O PIN WHEN POWER IS NOT SUPPLIED TO THE BOARD. IT WILL DAMAGE THE PROCESSOR AND VOID THE WARRANTY.**

**NO PINS ARE TO BE DRIVEN UNTIL AFTER THE SYS_RESET LINE GOES HIGH.**
### P8.01-P8.02

<table>
<thead>
<tr>
<th>Pin</th>
<th>P8.01</th>
<th>P8.02</th>
</tr>
</thead>
<tbody>
<tr>
<td>GND</td>
<td>GND</td>
<td></td>
</tr>
</tbody>
</table>

### P8.03-P8.05

<table>
<thead>
<tr>
<th>Pin</th>
<th>P8.03</th>
<th>P8.04</th>
<th>P8.05</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>HSIO6NB0</td>
<td>HSIO6PB0/CCC_NE_CLKIN_N_11</td>
<td>HSIO12NB0</td>
</tr>
<tr>
<td>BALL</td>
<td>V22</td>
<td>V22</td>
<td>V19</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>MSS GPIO_2[0]</td>
<td>MSS GPIO_2[1]</td>
<td>MSS GPIO_2[2]</td>
</tr>
<tr>
<td>GPIOS</td>
<td>User LED 0</td>
<td>User LED 1</td>
<td>User LED 2</td>
</tr>
<tr>
<td>ROBOTICS</td>
<td>MSS GPIO_2[0]</td>
<td>MSS GPIO_2[1]</td>
<td>MSS GPIO_2[2]</td>
</tr>
</tbody>
</table>

### P8.06-P8.09

<table>
<thead>
<tr>
<th>Pin</th>
<th>P8.06</th>
<th>P8.07</th>
<th>P8.08</th>
<th>P8.09</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>HSIO12PB0/CLKIN_N_9/CCC_NE_CLKIN_N_11</td>
<td>HSIO30NB0</td>
<td>HSIO30PB0/CLKIN_N_3/CCC_NW_CLKIN_N_9</td>
<td>HSIO8NB0</td>
</tr>
<tr>
<td>BALL</td>
<td>V20</td>
<td>V15</td>
<td>V14</td>
<td>V21</td>
</tr>
<tr>
<td>GPIOS</td>
<td>User LED 3</td>
<td>User LED 4</td>
<td>User LED 5</td>
<td>User LED 6</td>
</tr>
</tbody>
</table>

### P8.10-P8.13

<table>
<thead>
<tr>
<th>Pin</th>
<th>P8.10</th>
<th>P8.11</th>
<th>P8.12</th>
<th>P8.13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>HSIO8PB0/CCC_NE_CLKIN_N_10/CCC_NE_PLL0_OU</td>
<td>HSIO7NB0</td>
<td>HSIO7PB0/CCC_NE_PLL0_OU</td>
<td>GPIO47PB1</td>
</tr>
<tr>
<td>BALL</td>
<td>W21</td>
<td>Y21</td>
<td>Y20</td>
<td>B10</td>
</tr>
<tr>
<td>GPIOS</td>
<td>User LED 7</td>
<td>User LED 8</td>
<td>User LED 9</td>
<td>PWM_2:1</td>
</tr>
<tr>
<td>ROBOTICS</td>
<td>User LED 7</td>
<td>User LED 8</td>
<td>User LED 9</td>
<td>User LED 10</td>
</tr>
</tbody>
</table>

30 Chapter 4. Expansion
## P8.14-P8.16

<table>
<thead>
<tr>
<th>Pin</th>
<th>P8.14</th>
<th>P8.15</th>
<th>P8.16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>GPIO47NB1</td>
<td>HSIO34NB0</td>
<td>HSIO34PB0/CCC_NW_CLKIN_N_1</td>
</tr>
<tr>
<td>BALL</td>
<td>B9</td>
<td>T12</td>
<td>U12</td>
</tr>
</tbody>
</table>

## P8.17-P8.19

<table>
<thead>
<tr>
<th>Pin</th>
<th>P8.17</th>
<th>P8.18</th>
<th>P8.19</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>HSIO29PB0</td>
<td>HSIO15PB0/DQS/CCC_PLL1_OUT0</td>
<td>HSIO19NB0</td>
</tr>
<tr>
<td>BALL</td>
<td>W13</td>
<td>T16</td>
<td>W18</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>MSS GPIO_2[14]</td>
<td>MSS GPIO_2[15]</td>
<td>core_pwm[0] @ 0x41500000</td>
</tr>
<tr>
<td>GPIO</td>
<td>GPIO</td>
<td>GPIO</td>
<td>PWM_2:0</td>
</tr>
<tr>
<td>ROBOTICS</td>
<td>MSS GPIO_2[14]</td>
<td>MSS GPIO_2[15]</td>
<td>core_pwm[0] @ 0x41500000</td>
</tr>
</tbody>
</table>

## P8.20-P8.22

<table>
<thead>
<tr>
<th>Pin</th>
<th>P8.20</th>
<th>P8.21</th>
<th>P8.22</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>HSIO15NB0/DQS</td>
<td>HSIO9PB0/DQS/CCC_PLL0_OUT0</td>
<td>HSIO9NB0/DQS</td>
</tr>
<tr>
<td>BALL</td>
<td>R16</td>
<td>AA21</td>
<td>AA22</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>MSS GPIO_2[17]</td>
<td>MSS GPIO_2[18]</td>
<td>MSS GPIO_2[19]</td>
</tr>
<tr>
<td>GPIO</td>
<td>GPIO</td>
<td>GPIO</td>
<td>GPIO</td>
</tr>
</tbody>
</table>

## P8.23-P8.26

<table>
<thead>
<tr>
<th>Pin</th>
<th>P8.23</th>
<th>P8.24</th>
<th>P8.25</th>
<th>P8.26</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>HSIO18PB0/CLKIN_N_7</td>
<td>HSIO18NB0</td>
<td>HSIO16PB0</td>
<td>GPIO49NB1</td>
</tr>
<tr>
<td>BALL</td>
<td>A818</td>
<td>AA18</td>
<td>V17</td>
<td>A12</td>
</tr>
<tr>
<td>GPIO</td>
<td>GPIO</td>
<td>GPIO</td>
<td>GPIO</td>
<td>GPIO</td>
</tr>
</tbody>
</table>

## 4.1. Cape Headers
### P8.27-P8.29

<table>
<thead>
<tr>
<th>Pin</th>
<th>P8.27</th>
<th>P8.28</th>
<th>P8.29</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>GPIO49PB1/CLKIN_5_5</td>
<td>GPIO51NB1</td>
<td>GPIO51PB1/CLKIN_5_6</td>
</tr>
<tr>
<td>BALL</td>
<td>A13</td>
<td>B14</td>
<td>B13</td>
</tr>
<tr>
<td>GPIOS</td>
<td>GPIO</td>
<td>GPIO</td>
<td>GPIO</td>
</tr>
</tbody>
</table>

### P8.30-P8.32

<table>
<thead>
<tr>
<th>Pin</th>
<th>P8.30</th>
<th>P8.31</th>
<th>P8.32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>GPIO50NB1/DQS</td>
<td>GPIO50PB1/DQS</td>
<td>GPIO53NB1</td>
</tr>
<tr>
<td>BALL</td>
<td>D14</td>
<td>D13</td>
<td>B15</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>MSS_GPIO_2[27]</td>
<td>core_gpio[0] @ 0x41100000</td>
<td>core_gpio[1] @ 0x41100000</td>
</tr>
<tr>
<td>GPIOS</td>
<td>GPIO</td>
<td>GPIO</td>
<td>GPIO</td>
</tr>
<tr>
<td>ROBOTICS</td>
<td>MSS_GPIO_2[27]</td>
<td>core_gpio[0] @ 0x41100000</td>
<td>core_gpio[1] @ 0x41100000</td>
</tr>
</tbody>
</table>

### P8.33-P8.35

<table>
<thead>
<tr>
<th>Pin</th>
<th>P8.33</th>
<th>P8.34</th>
<th>P8.35</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>GPIO53PB1/CLKIN_5_7</td>
<td>GPIO52NB1/LPRB_B</td>
<td>GPIO52PB1/LPRB_A</td>
</tr>
<tr>
<td>BALL</td>
<td>A15</td>
<td>C15</td>
<td>C14</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>core_gpio[2] @ 0x41100000</td>
<td>core_gpio[3] @ 0x41100000</td>
<td>core_gpio[4] @ 0x41100000</td>
</tr>
<tr>
<td>GPIOS</td>
<td>GPIO</td>
<td>GPIO</td>
<td>GPIO</td>
</tr>
<tr>
<td>ROBOTICS</td>
<td>core_gpio[2] @ 0x41100000</td>
<td>core_gpio[3] @ 0x41100000</td>
<td>core_gpio[4] @ 0x41100000</td>
</tr>
</tbody>
</table>

### P8.36-P8.38

<table>
<thead>
<tr>
<th>Pin</th>
<th>P8.36</th>
<th>P8.37</th>
<th>P8.38</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>GPIO37NB1</td>
<td>GPIO37PB1/CCC_SW_CLKIN_5_1</td>
<td>GPIO3NB1</td>
</tr>
<tr>
<td>BALL</td>
<td>B4</td>
<td>C4</td>
<td>C17</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>core_gpio[5] @ 0x41100000</td>
<td>core_gpio[6] @ 0x41100000</td>
<td>core_gpio[7] @ 0x41100000</td>
</tr>
<tr>
<td>GPIOS</td>
<td>GPIO</td>
<td>GPIO</td>
<td>GPIO</td>
</tr>
<tr>
<td>ROBOTICS</td>
<td>core_gpio[5] @ 0x41100000</td>
<td>core_gpio[6] @ 0x41100000</td>
<td>core_gpio[7] @ 0x41100000</td>
</tr>
</tbody>
</table>

---

32 Chapter 4. Expansion
### 4.1.2 Connector P9

The following tables show the pinout of the **P9** expansion header. The gateware image is responsible for setting the function of each pin. Refer to the processor documentation for more information on these pins and detailed descriptions of all of the pins listed. In some cases there may not be enough signals to complete a group of signals that may be required to implement a total interface.

The column heading is the pin number on the expansion header.

The **Name** row is the pin name on the processor.

The **BALL** row is the pin number on the processor.

The rows below **BALL** are the gateware setting for each pin.

**NOTES:**

**DO NOT APPLY VOLTAGE TO ANY I/O PIN WHEN POWER IS NOT SUPPLIED TO THE BOARD. IT WILL DAMAGE THE PROCESSOR AND VOID THE WARRANTY.**

**NO PINS ARE TO BE DRIVEN UNTIL AFTER THE SYS_RESET LINE GOES HIGH.**

#### P8.39-P8.41

<table>
<thead>
<tr>
<th>Pin</th>
<th>P8.39</th>
<th>P8.40</th>
<th>P8.41</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>GPIO3PB1/CCC_SE_CLKIN_S_10/CCC_SE_PLL1_OUT0</td>
<td>GPIO5NB1</td>
<td>GPIO5PB1/CCC_SE_CLKIN_S_11</td>
</tr>
<tr>
<td>BALL</td>
<td>B17</td>
<td>B18</td>
<td>A18</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>core_gpio[8] @ 0x41100000</td>
<td>core_gpio[9] @ 0x41100000</td>
<td>core_gpio[10] @ 0x41100000</td>
</tr>
<tr>
<td>GPIO</td>
<td>GPIO</td>
<td>GPIO</td>
<td>GPIO</td>
</tr>
<tr>
<td>GPIONAME</td>
<td>core_gpio[8] @ 0x41100000</td>
<td>core_gpio[9] @ 0x41100000</td>
<td>core_gpio[10] @ 0x41100000</td>
</tr>
<tr>
<td>ROBOTICS</td>
<td>GPIO</td>
<td>_GPIO</td>
<td>GPIO</td>
</tr>
</tbody>
</table>

#### P8.42-P8.44

<table>
<thead>
<tr>
<th>Pin</th>
<th>P8.42</th>
<th>P8.43</th>
<th>P8.44</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>GPIO36NB1</td>
<td>GPIO36PB1/CCC_SW_CLKIN_S_0</td>
<td>GPIO42NB1</td>
</tr>
<tr>
<td>BALL</td>
<td>D6</td>
<td>D7</td>
<td>D8</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>core_gpio[11] @ 0x41100000</td>
<td>core_gpio[12] @ 0x41100000</td>
<td>core_gpio[13] @ 0x41100000</td>
</tr>
<tr>
<td>GPIO</td>
<td>GPIO</td>
<td>GPIO</td>
<td>GPIO</td>
</tr>
<tr>
<td>GPIONAME</td>
<td>core_gpio[11] @ 0x41100000</td>
<td>core_gpio[12] @ 0x41100000</td>
<td>core_gpio[13] @ 0x41100000</td>
</tr>
<tr>
<td>ROBOTICS</td>
<td>GPIO</td>
<td>_GPIO</td>
<td>GPIO</td>
</tr>
</tbody>
</table>

#### P8.45-P8.46

<table>
<thead>
<tr>
<th>Pin</th>
<th>P8.45</th>
<th>P8.46</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>GPIO42PB1</td>
<td>GPIO4PB1/CCC_SE_PLL1_OUT1</td>
</tr>
<tr>
<td>BALL</td>
<td>D9</td>
<td>D18</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>core_gpio[14] @ 0x41100000</td>
<td>core_gpio[15] @ 0x41100000</td>
</tr>
<tr>
<td>GPIO</td>
<td>GPIO</td>
<td>GPIO</td>
</tr>
<tr>
<td>GPIONAME</td>
<td>core_gpio[14] @ 0x41100000</td>
<td>core_gpio[15] @ 0x41100000</td>
</tr>
<tr>
<td>ROBOTICS</td>
<td>GPIO</td>
<td>_GPIO</td>
</tr>
</tbody>
</table>
### Chapter 4. Expansion

#### P9.01-P9.05

<table>
<thead>
<tr>
<th>Pin</th>
<th>P9.01</th>
<th>P9.02</th>
<th>P9.03</th>
<th>P9.04</th>
<th>P9.05</th>
</tr>
</thead>
<tbody>
<tr>
<td>GND</td>
<td>GND</td>
<td>VCC_3V3</td>
<td>VCC_3V3</td>
<td>VDD_5V</td>
<td></td>
</tr>
</tbody>
</table>

#### P9.06-P9.10

<table>
<thead>
<tr>
<th>Pin</th>
<th>P9.06</th>
<th>P9.07</th>
<th>P9.08</th>
<th>P9.10</th>
</tr>
</thead>
<tbody>
<tr>
<td>VDD_5V</td>
<td>SYS_5V</td>
<td>SYS_5V</td>
<td>SYS_RSTN</td>
<td></td>
</tr>
</tbody>
</table>

#### P9.11-P9.13

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>GPIO38NB1/DQS</td>
<td>GPIO38PB1/DQS/CCC_SW_PLL1_OUT0</td>
<td>GPIO2NB1/DQS</td>
</tr>
<tr>
<td>BALL</td>
<td>B5</td>
<td>C5</td>
<td>D19</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>MMUART4</td>
<td>core_gpio[1] @ 0x41200000</td>
<td>MMUART4</td>
</tr>
<tr>
<td>GPIO</td>
<td>UART4 RX</td>
<td>core_gpio[1] @ 0x41200000</td>
<td>UART4 RX</td>
</tr>
<tr>
<td>ROBOTICS</td>
<td>core_gpio[0] @ 0x41200000</td>
<td>core_gpio[1] @ 0x41200000</td>
<td>core_gpio[2] @ 0x41200000</td>
</tr>
</tbody>
</table>

#### P9.14-P9.16

<table>
<thead>
<tr>
<th>Pin</th>
<th>P9.14</th>
<th>P9.15</th>
<th>P9.16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>GPIO39PB1/CLKIN_5_2/CCC_SW_CLKIN_5_2/CCC_SW_PLL1_O</td>
<td>GPIO40NB1</td>
<td>GPIO40PB1/CCC_SW_PLL1_OUT1</td>
</tr>
<tr>
<td>BALL</td>
<td>C6</td>
<td>A5</td>
<td>A6</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>core_pwm[0] @ 0x41400000</td>
<td>core_gpio[4]</td>
<td>core_pwm[1] @ 0x41400000</td>
</tr>
<tr>
<td>GPIO</td>
<td>core_gpio[3] @ 0x41200000</td>
<td>core_gpio[4]</td>
<td>core_pwm[1] @ 0x41200000</td>
</tr>
<tr>
<td>ROBOTICS</td>
<td>core_pwm[0] @ 0x41200000</td>
<td>core_gpio[1]</td>
<td>core_pwm[1] @ 0x41200000</td>
</tr>
<tr>
<td>PWM_1:0</td>
<td>@</td>
<td>@</td>
<td></td>
</tr>
</tbody>
</table>

#### P9.17-P9.19

<table>
<thead>
<tr>
<th>Pin</th>
<th>P9.17</th>
<th>P9.18</th>
<th>P9.19</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>GPIO44NB1/DQS</td>
<td>GPIO44PB1/DQS/CCC_SW_PLL0_OUT0</td>
<td>GPIO45PB1/CCC_SW_PLL0_OUT0</td>
</tr>
<tr>
<td>BALL</td>
<td>C9</td>
<td>C10</td>
<td>A10</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>–</td>
<td>–</td>
<td>MSS I2C0</td>
</tr>
<tr>
<td>GPIO</td>
<td>core_gpio[6] @ 0x41200000</td>
<td>core_gpio[7] @ 0x41200000</td>
<td>I2C0 SCL</td>
</tr>
<tr>
<td>ROBOTICS</td>
<td>–</td>
<td>–</td>
<td>MSS I2C0</td>
</tr>
<tr>
<td>PWM_1:0</td>
<td>–</td>
<td>–</td>
<td>I2C0 SCL</td>
</tr>
</tbody>
</table>
### P9.20-P9.22

<table>
<thead>
<tr>
<th>Pin</th>
<th>P9.20</th>
<th>P9.21</th>
<th>P9.22</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>GPIO45NB1</td>
<td>GPIO43NB1</td>
<td>GPIO43PB1</td>
</tr>
<tr>
<td>BALL</td>
<td>A11</td>
<td>B8</td>
<td>A8</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>MSS I2C0</td>
<td>I2C0 SDA</td>
<td>I2C0 SDA</td>
</tr>
<tr>
<td>GPIOS</td>
<td>core_gpio[8] @ 0x41200000</td>
<td>core_gpio[8] @ 0x41200000</td>
<td>core_gpio[8] @ 0x41200000</td>
</tr>
<tr>
<td>ROBOTICS</td>
<td>MSS I2C0</td>
<td>I2C0 SDA</td>
<td>I2C0 SDA</td>
</tr>
</tbody>
</table>

### P9.23-P9.25

<table>
<thead>
<tr>
<th>Pin</th>
<th>P9.23</th>
<th>P9.24</th>
<th>P9.25</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>GPIO48NB1</td>
<td>GPIO48PB1/CLKIN_S_4</td>
<td>GPIO41NB1</td>
</tr>
<tr>
<td>BALL</td>
<td>C12</td>
<td>B12</td>
<td>B7</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>core_gpio[10] @ 0x41200000</td>
<td>core_gpio[12] @ 0x41200000</td>
<td>core_gpio[12] @ 0x41200000</td>
</tr>
<tr>
<td>GPIOS</td>
<td>core_gpio[10] @ 0x41200000</td>
<td>core_gpio[11] @ 0x41200000</td>
<td>core_gpio[12] @ 0x41200000</td>
</tr>
<tr>
<td>ROBOTICS</td>
<td>core_gpio[2] @ 0x41200000</td>
<td>core_gpio[3] @ 0x41200000</td>
<td>core_gpio[3] @ 0x41200000</td>
</tr>
</tbody>
</table>

### P9.26-P9.28

<table>
<thead>
<tr>
<th>Pin</th>
<th>P9.26</th>
<th>P9.27</th>
<th>P9.28</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>GPIO41PB1/CLKIN_S_3/CCC_SW_CLKIN_S_3</td>
<td>GPIO46NB1</td>
<td>GPIO46PB1/CCC_SW_PLL0_OUT1</td>
</tr>
<tr>
<td>BALL</td>
<td>A7</td>
<td>D11</td>
<td>C11</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>core_gpio[13] @ 0x41200000</td>
<td>core_gpio[14] @ 0x41200000</td>
<td>core_gpio[15] @ 0x41200000</td>
</tr>
<tr>
<td>GPIOS</td>
<td>core_gpio[13] @ 0x41200000</td>
<td>core_gpio[14] @ 0x41200000</td>
<td>core_gpio[15] @ 0x41200000</td>
</tr>
<tr>
<td>ROBOTICS</td>
<td>core_gpio[2] @ 0x41200000</td>
<td>core_gpio[3] @ 0x41200000</td>
<td>core_gpio[3] @ 0x41200000</td>
</tr>
</tbody>
</table>

### P9.29-P9.31

<table>
<thead>
<tr>
<th>Pin</th>
<th>P9.29</th>
<th>P9.30</th>
<th>P9.31</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>GPIO1PB1/CLKIN_S_9/CCC_SE_CLKIN_S_9</td>
<td>GPIO1NB1</td>
<td>GPIO4NB1</td>
</tr>
<tr>
<td>BALL</td>
<td>F17</td>
<td>F16</td>
<td>E18</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>core_gpio[16] @ 0x41200000</td>
<td>core_gpio[17] @ 0x41200000</td>
<td>core_gpio[18] @ 0x41200000</td>
</tr>
<tr>
<td>GPIOS</td>
<td>core_gpio[16] @ 0x41200000</td>
<td>core_gpio[17] @ 0x41200000</td>
<td>core_gpio[18] @ 0x41200000</td>
</tr>
<tr>
<td>ROBOTICS</td>
<td>core_gpio[5] @ 0x41200000</td>
<td>core_gpio[5] @ 0x41200000</td>
<td>core_gpio[5] @ 0x41200000</td>
</tr>
</tbody>
</table>

### P9.32-P9.40

<table>
<thead>
<tr>
<th>Pin</th>
<th>P9.32</th>
<th>P9.34</th>
</tr>
</thead>
<tbody>
<tr>
<td>VDD_ADC</td>
<td>GND</td>
<td></td>
</tr>
</tbody>
</table>
### P9.41-P9.42

<table>
<thead>
<tr>
<th>Pin</th>
<th>P9.41</th>
<th>P9.42</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>GPIO0PB1/CLKIN_S_8/CCC_SE_CLKIN_S_8/CCC_SE_PLL0_OUT0</td>
<td>GPIO0NB1</td>
</tr>
<tr>
<td>BALL</td>
<td>E15</td>
<td>E14</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>core_gpio[19] @ 0x41200000</td>
<td>core_pwm[0] @ 0x41000000</td>
</tr>
<tr>
<td>GPIO</td>
<td>core_gpio[19] @ 0x41200000</td>
<td>PWM_0.0</td>
</tr>
<tr>
<td>GPIO</td>
<td>core_gpio[19] @ 0x41200000</td>
<td>core_gpio[20] @ 0x41200000</td>
</tr>
<tr>
<td>ROBOTICS</td>
<td>core_gpio[19] @ 0x41200000</td>
<td>–</td>
</tr>
<tr>
<td>GPIO</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

### P9.43-P9.46

<table>
<thead>
<tr>
<th>P9.43</th>
<th>P9.44</th>
<th>P9.45</th>
<th>P9.46</th>
</tr>
</thead>
<tbody>
<tr>
<td>GND</td>
<td>GND</td>
<td>GND</td>
<td>GND</td>
</tr>
</tbody>
</table>
Chapter 5

Demos

5.1 Upgrade BeagleV-Fire Gateware

This document describes how to upgrade your BeagleV-Fire's gateware. This approach can be used out of the box using Linux commands executed on BeagleV-Fire.

5.1.1 Required Equipment

- BeagleV-Fire board
- USB-C cable
- Ethernet cable

The USB-C cable provides power, a serial interface to BeagleV-Fire and allows connecting to BeagleV-Fire through a browser using IP address 192.168.7.2.

The Ethernet cable connected to your local network (LAN) allows connecting to BeagleV-Fire using the SSH protocol. It also allows BeagleV-Fire to retrieve updated packages through your local network's Internet connection.

5.1.2 Connect to BeagleV-Fire Linux Command Line Interface

BeagleV-Fire boots Linux out of the box. Like all Beagleboard boards there are several methods to get BeagleV-Fire's Linux command prompt.

- Cockpit
- SSH
- Serial port

Cockpit

Enter the following URL in your web browser: https://beaglev.localdomain:9090/

On first use, click through the security warning. Login using beagle/temppwd as user/password. Click on Terminal in the left pane. You now have a Linux command prompt running on your BeagleV-Fire. Next step: enter the commands described in the Gateware Upgrade Linux Commands section of this document.

Note: You can connect to the Cockpit using the IP address dynamically assigned to your BeagleV-Fire in your local Ethernet network. One method of finding the value of that dynamically assigned IP address is to open...
a serial terminal though the USB port and use the `ip address` Linux command. Please refer to the USB Serial Port section.

**SSH**

Like all Beagleboard boards, you can SSH to the board through the USB interface by using IP address 192.168.7.2.

**Note:** On Windows, this approach may require some drivers to be updated or installed. Use one of the other approaches if you are not immediately successful with this one. You can circle back later to adjust your Windows installation if required.

**Serial Port**

A serial port is available through the USB-C port. This serial port becomes available once Linux has booted on BeagleV-Fire. Please wait a couple of minutes after powering up the board before looking for additional serial ports reported by your host computer’s operating system. You can then use your favorite serial port terminal tool such as Putty or Screen to access the BeagleV-Fire Linux command prompt.

For example on your Linux host computer:

```
screen /dev/ttyACM0 115200
```

Where `/dev/ttyACM0` is an additional serial port that appeared after BeagleV-Fire was connected to your Linux host computer. This serial port can be identified using the `dmesg | grep tty` Linux command which will show the most recent serial port added to the host computer.

On Windows, BeagleV-Fire’s serial port number will show in the Windows Device Manager. Use that serial port number in Putty with a speed 115200 baud, no flow-control.

### 5.1.3 Gateware Upgrade Linux Commands

**Note:** BeagleV-Fire needs to be connected to the internet through your local network for the commands in this section to work. The connection can be through the Ethernet port or the Wi-Fi module.

**Install bbb.io-gateware**

You need to install the `bbb.io-gateware` package. This will allow retrieving the most up-to-date gateware.

```
sudo apt install bbb.io-gateware
```

**Retrieve Available Updated Linux packages List**

The list will include the latest BeagleV-Fire gateware packages.

```
sudo apt update
```
Upgrade Linux Packages

This will upgrade the BeagleV-Fire gateware Linux programming files located under /usr/share/beagleboard/gateware. Several directories are found in that location, each containing programming files for one individual gateware configuration.

```
sudo apt upgrade
```

Launch Reprogramming of BeagleV-Fire's FPGA

Change directory to /usr/share/beagleboard/gateware. This directory contains a script performing the gateware's reprogramming. It also contains one directory for each of the possible gateware configuration that can be programmed into your BeagleV-Fire. The name of one of these directories is passed as argument to the script to specify which gateware configuration you wish to program your BeagleV-Fire with.

```
cd /usr/share/beagleboard/gateware
./change-gateware.sh default
```

**Important:** Do not power-off BeagleV-Fire until it has rebooted by itself. The gateware reprogramming may take a couple of minutes.

The change-gateware script programs the selected gateware and its associated device tree overlays into the PolarFire SoC System Controllers SPI flash and triggers a software reboot. During the reboot, the Hart Software Services (HSS) will request the PolarFire SoC System Controller to reprogram the FPGA and eNVM. The PolarFire SoC System Controller will reprogram the FPGA if it finds it contains a different design version than the one in the SPI Flash. The board reboots on completion of the FPGA reprogramming.

5.2 Flashing gateware and Linux image

In this tutorial we are going to learn to flash the gateware image to FPGA and sdcard.image to eMMC storage.

**Important:** Additional hardware required:

1. FlashPro5/6 programmer
2. Tag connect TC2050-IDC-NL 10-Pin No-Legs Cable with Ribbon connector
3. TC2050-CLIP-3PACK Retaining CLIP board for TC2050-NL cables

5.2.1 Programming & Debug tools installation

To flash a gateware image to your BeagleV-Fire board you will require a FlashPro5/6 and FlashPro Express (FPExpress) tool which comes pre-installed as part of Libero SoC Design Suite. A standalone FlashPro Express tool is also available with MicroChip's Programming and Debug Tools package, which we are going to use for this tutorial. Below are the steps to install the software:

1. Download the zip for your operating system from Programming and Debug Tools page.
2. Unzip the file and in the unzipped folder you will find launch_installer.sh and Program_Debug_v2023.1.bin.
3. Execute the launch_installer.sh script to start the installation procedure.
[lorforlinux@fedora Program_Debug_v2023.1_lin] $ ./launch_installer.sh

No additional packages to install for installer usage

Requirement search complete.

See /tmp/check_req_installer608695.log for information.

Launch of installer
Preparing to install
Extracting the JRE from the installer archive...
Unpacking the JRE...

**Note:** It’s recommended to install under `home/user/microchip` for Linux users.

---

### Enabling non-root user to access FlashPro

1. Download 60-openocd.rules
2. Copy udev rule `sudo cp 60-openocd.rules /etc/udev/rules.d`
3. Trigger udevadm using `sudo udevadm trigger` or reboot the PC for the changes to take effect

---

### 5.2.2 Flashing gateware image

**Note:** Content below is valid for beta testers only.

---

### Launch FPExpress

1. Download FlashProExpress.zip file and unzip, it contains the *.job file which we need to create a new project in FPExpress.
2. Open up a terminal and go to `/home/user/microchip/Program_Debug_v202X.Y/Program_Debug_Tool/bin` which includes FPExpress tool.
3. Execute `. /FPExpress` in terminal to start FlashPro Express software.

---

### Create new project

**Important:** Make sure you have your FlashPro5/6 connected before you create a new project.

Press CTRL+N to create a file and you will see a pop-up window like shown below,
Follow the steps below as annotated in the image above:

1. Click on browse (1) button to select the job file.
2. Click on browse (2) button to select the project location.
3. Click ok button to finish.

If your FlashPro5/6 is connected properly you’ll see the window shown below:

Following the annotation in the image above:

1. From drop-down select Program action
2. Click on RUN button

5.2. Flashing gateware and Linux image
3. Shows the progress

If you see a lot of green color and the progress bar says **PASSED** then well done you have successfully flashed the gateware image on your BeagleV-Fire board.

### 5.2.3 Flashing eMMC

Connect to BeagleV-Fire UART debug port using a 3.3v USB to UART bridge.

![USB to UART debug port](image)

Now you can run `tio <port>` in a terminal window to access the UART debug port connection. Once you are connected properly you can press the Reset button which will show you a progress bar like in the image below:

```
HSS: decompressing from eNVM to L2 Scratch ... Passed
DDR training ...
  60% [--------------------]
```

Once you see `>` you can execute the commands below:

1. `>> mmc`
2. `>> usbdmsc`

After executing the commands above your BeagleV-Fire’s eMMC will be exposed as a mass storage device like shown in the image below:

Once your board is exposed as a mass storage device you can use Balena Etcher to flash the `sdcard.img` on your BeagleV-Fire’s eMMC.

Select image

1. Select the `sdcard.img` file from your local drive storage.
5.2. Flashing gateware and Linux image
2. Click on select target.

Select Target

1. Select MCC PolarFireSoC_msd as target.
2. Click Select (1) to proceed.

Flash image

1. Click on Flash! to flash the sdcard.img on BeagleV-Fire eMMC storage.

Congratulations! with that done you have fully updated BeagleV-Fire board with up to date gateware image on it’s PolarFire SoC’s FPGA Fabric and linux image on it’s eMMC storage.

5.3 Microchip FPGA Tools Installation Guide

Instructions for installing the Microchip FPGA tools on a Ubuntu 20.04 desktop.

Important: We will be providing instances of Libero that you can run from git.beagleboard.org’s gitlab-runners such that you do not need to install the tools on your local machine.

5.3.1 Install Libero 2022.3

- Download installer from the Microchip’s fpga and soc design tools section.
- Install Libero

```
unzip Libero_SoC_v2022.3_lin.zip

cd Libero_SoC_v2022.3_lin/
```

(continues on next page)
5.3. Microchip FPGA Tools Installation Guide
./launch_installer.sh

**Important:** Do not use the default location suggested by the Libero installer. Instead of /usr/local/Microchip/Libero_SoC_v2022.3 install into ~/Microchip/Libero_SoC_v2022.3

Run the post installation script which will install missing packages:

```bash
sudo /home/<USER-NAME>/Microchip/Libero_SoC_v2022.3/Logs/req_to_install.sh
```

No need to run the FlashPro hardware installation scripts. This will be taken care of as part of the SoftConsole installation.

### 5.3.2 Install SoftConsole 2022.2

- Download installer from Microchip website.

```bash
sudo chmod +x Microchip-SoftConsole-v2022.2-RISC-V-747-linux-x64-installer.run
./Microchip-SoftConsole-v2022.2-RISC-V-747-linux-x64-installer.run
```

Accept the license, Click Forward, Finish.

Perform the post installation steps as described in the html file opened when you click Finish.

**Important:** Please pay special attention to the “Enabling non-root user to access FlashPro” section of the post-installation instructions. This will actually allow you to program the board using Libero.
5.3.3 Install the Libero licensing daemon

Download the 64 bit Licensing Daemons from the Microchip’s daemons section

- Linux_Licensing_Daemon_11.16.1_64-bit.tar.gz
- Windows_Licensing_Daemon_11.16.1_64-bit.zip

Copy the downloaded file to the Microchip directory within your home directory and untar it.

```bash
cd ~/Microchip

tar -xvf Linux_Licensing_Daemon_11.16.1_64-bit.tar.gz
```

Install the Linux Standard Base:

```bash
sudo apt-get update
sudo apt-get -y install lsb
```

5.3.4 Request a Libero Silver license

- Visit microchip’s fpga and soc design tool licensing page
- Click on Register a free license button and Register or login.
- Click “Request Free License” and choose “Libero Silver 1Yr Floating License for Windows/Linux Server” from the list.
- Enter your MAC address and click register.

**Note:** A MAC address looks something like 12:34:56::78:ab:cd when you use the “ip address” command to find out its value on your Linux machine. However, you need to enter it as 123456abcd in this dialog box.

You will get an email with a license.dat file. Copy it into the ~/Microchip/license directory. Edit the License.dat file to replace the <put.hostname.here> string with... localhost.

5.3.5 Execute tool setup script

Download the script:

```bash
#!/bin/bash

# Edit the following section with the location where the following tools are installed:
# - SoftConsole (SC_INSTALL_DIR)
# - Libero (LIBERO_INSTALL_DIR)
# - Licensing daemon for Libero (LICENSE_DAEMON_DIR)

export SC_INSTALL_DIR=/home/$USER/Microchip/SoftConsole-v2022.2-RISC-V-747
export LIBERO_INSTALL_DIR=/home/$USER/Microchip/Libero_SoC_v2023.2
export LICENSE_DAEMON_DIR=/home/$USER/Microchip/Linux_Licensing_Daemon
export LICENSE_FILE_DIR=/home/$USER/Microchip/license
```

(continues on next page)
# The following was tested on Ubuntu 20.04 with:
# - Libero 2023.2
# - SoftConsole 2022.2

# SoftConsole
export PATH=$PATH:$SC_INSTALL_DIR/ri5cy-unknown-elf-gcc/bin
export FPGENPROG=$LIBERO_INSTALL_DIR/Libero/bin64/fpgenprog

# Libero
export PATH=$PATH:$LIBERO_INSTALL_DIR/Libero/bin:$LIBERO_INSTALL_DIR/Libero/bin64
export PATH=$PATH:$LIBERO_INSTALL_DIR/Synplify/bin
export PATH=$PATH:$LIBERO_INSTALL_DIR/Model/modeltech/linuxacoem
export LOCALE=C
export LD_LIBRARY_PATH=/usr/lib/i386-linux-gnu:$LD_LIBRARY_PATH

# Libero License daemon
export LM_LICENSE_FILE=1702@localhost
export SNPSLMD_LICENSE_FILE=1702@localhost
$LICENSE_DAEMON_DIR/lmgrd -c $LICENSE_FILE_DIR-License.dat -l $LICENSE_FILE_DIR/.../license.log

setup-microchip-tools.sh

Source the script:

```
./setup-microchip-tools.sh
```

### Important:
Do not forget the leading dot. It matters. You will need to run this every time you restart your machine.

You can then start Libero to open an existing Libero project.

```
libero
```

However you will more than likely want to use Libero to run a TCL script that will build a design for you.

```
libero SCRIPT:BUILD_A_DESIGN.tcl
```

## 5.4 Gateware Design Introduction

The PolarFire SoC device used on BeagleV-Fire is an SoC FPGA which includes a RISC-V processors subsystem and a PolarFire FPGA on the same die. The gateware configures the Microprocessor subsystem’s hardware and programs the FPGA with digital logic allowing customization of the use of BeagleV-Fire connectors.
5.4.1 Gateware Architecture

The diagram below is a simplified overview of the gateware's structure.

The overall gateware is made-up of several blocks, some of them interchangeable. These blocks are all clocked and reset by another “Clock and Resets” block not showed in the diagram for clarity. A 125MHz, and a 160MHz clock are provided for use by the gateware blocks.

Each gateware block is associated with one of BeagleV-Fire’s connectors.

All gateware blocks have an AMBA APB target interface for software to access control and status registers defined by the gateware to operate digital logic defined by the gateware block. This is the software's control path into the gateware block.

Some gateware blocks also have an AMBA AXI target and/or source interfaces. The AXI interfaces are typically used to move high volume of data at high throughput in and out of DDR memory. For example, the M.2 gateware uses these interfaces to transfer data in and out of its PCIe root port.

Cape Gateware

The cape gateware handles the P8 and P9 connectors signals. This is where support for specific capes is implemented.

This is a very good place to start learning about FPGA and how to customize gateware.
The SYZYGY gateware handles the high-speed connector signals. This connector includes:

- up to three transceivers capable of 12.7Gbps communications
- One SGMII interface
- 10 high-speed I/Os
- Clock inputs

There is a lot of fun that can be had with this interface given its high-speed capabilities.

Please note that only two transceivers can be used when the M.2 interface is enabled.

The MIPI gateware handles the signals coming from the camera interface.

Gateware for the MIPI-CSI interface is Work-In-Progress.

The M.2 gateware implements the PCIe interface used for Wi-Fi modules. It connects the processor subsystem to the PCIe controller associated with the tranceivers bank.

There is limited fun to have here. You either include this block or not in your bitstream.

The M.2 gateware uses one of the four available 12.7 Gbps transceivers. Only two out of the three SYZYGY tranceivers can be used when the M.2 is included in the bitstream. This gateware needs to omited from the bitstream if you want to use all three 12.7Gbps transceivers on the SYZYGY high-speed connector.

The RISC-V Processors Subsystem also includes some gateware mostly dealing with exposing AMBA bus interfaces for the other gateware blocks to attach to. It also handles immutable aspects of the gateware related to how some PolarFire-SoC signals are used to connect BeagleV-Fire peripherals such as the ADC and EEPROM. As such the RISC-V Processors Subsystem gateware is not intended to be customized.

### 5.5 How to retrieve BeagleV-Fire’s gateware version

There are two methods to find out what gateware is programmed on a board.

#### 5.5.1 Device Tree

The device tree overlays contains the list of gateware blocks included in the overall gateware design. You can retrieve that information using the following command:

```
tree /proc/device-tree/chosen/overlays/
```

This should give an output similar to the one below.

The gateware version can be retrieve by reading one of the overlay files. For example, the command:

```
cat /proc/device-tree/chosen/overlays/ROBOTICS-CAPE-GATEWARE
```
5.5.2 Bootloader messages

The Hart Software Services display the gateware design name and design version retrieve from the FPGA at system start-up.

```
beagle@BeagleV:$ tree /proc/device-tree/chosen/overlays/
    /proc/device-tree/chosen/overlays/
        name
        ROBOTICS-CAPE-GATEWARE

beagle@BeagleV:$ cat /proc/device-tree/chosen/overlays/ROBOTICS-CAPE-GATEWARE
BVF-8.3.0-5-g3e0d338
```

should result in:

where the result of a “git describe” command on the gateware repository is displayed. This provides the most recent tag on the gateware repository followed by information about additional commits if some exist. In the example above, the gateware was created from a gateware repository hash 3e0d338 which is 5 commits more recent than tag BVF-0.3.0.

5.6 Gateware Full Build Flow

5.6.1 Introduction

BeagleV-Fire gateware is made up of several components:

- Digital design for the FPGA fabric.
- Microprocessor Subsystem (MSS) configuration containing MSS configuration register values.
- A zero stage bootloader (HSS).
- A set of device tree overlays describing the content of the FPGA fabric.

The FPGA’s digital design is a combination of:

- HDL/Verilog source code
- TCL scripts configuring IP blocks
- TCL scripts stitching IP blocks together
- Microprocessor Subsystem (MSS) configuration describing the MSS port list
- Pin, placement and timing constraints

The Hart Software Service (HSS) zero stage bootloader
- Configures the PolarFire SoC chip.
- Retrieves the next boot stage from eMMC and hand-over to the next boot stage (e.g. u-boot)
- Makes the board appear as a USB mass-storage for populating the eMMC with secondary boot-loader and operating system image.

The chip configuration applied by the HSS includes the configuration of:
- Clocks
- Memory controllers
- IOs
- Transceivers

Of course all these components need to be in sync with each other for the system to work properly. This is the reason for using a gateware build system rather than building and tracking each component individually.

### 5.6.2 Programming artifacts

The gateware builder for BeagleV-Fire produces two programming artifacts:
- A bitstream containing the FPGA fabric and eNVM programming
- A device tree overlay describing the FPGA content.
These two artifacts are packaged differently depending on the programming method. They are merged into a single programming file for DirectC (.dat) and FlashPro Express (.job). They are kept separate for Linux programming (mpfs_bitstream.spi and mpfs_dtbo.spi).

5.6.3 Programming BeagleV-Fire with new gateware

There are several methods possible for programming the BeagleV-Fire with new gateware:

- Linux script executed on the BeagleV-Fire board.
- Running DirectC on another single board computer
- Using Microchip’s FlashPro Express

Linux script

This is the recommended approach. It does not require any additional hardware. Simply run the script located in /usr/share/beagleboard/gateware. You should use this method unless you have soft-bricked your BeagleV-Fire.

DirectC

This approach uses a single board computer (SBC) connected to the BeagleV-Fire JTAG port. The SBC bit-bangs the FPGA programming protocol over GPIOs. This approach is only required for recovering a soft-bricked BeagleV-Fire.

FlashPro Express

This approach uses Microchip’s FlashPro Express programming software and a FlashPro6 JTAG programmer. I would recommend using the Linux script even if you are an existing Microchip FPGA user with all the tools. This approach makes most sense when doing bare metal software development and already have a FlashPro programmer and don’t care about device tree overlays.

5.7 Gateware TCL Scripts Structure

This document describes the structure of the gateware TCL scripts. It is of interest to understand how to extend or customize the gateware.

The Libero SoC TCL Command Reference Guide describes the TCL command used in the gateware scripts.

5.7.1 Gateware Project

The gateware project is made up of:

- TCL scripts
- HDL/Verilog source code
- IO pin constraints
- Placement constraints
- Device tree overlays

All these files are found in the FPGA-design directory.
5.7.2 Gateware Components

The gateware is organized into 6 components:

- Clocks and reset control
- A base RISC-V microprocessor subsystem
- Cape interface
- M.2 interface
- MIPI camera interface
- SYZYGY high speed interface

5.7.3 Gateware Build Options

Each interface component may have a number of build options. For example, which cape will be supported by the generated gateware.
The name of the directories within the component’s directory are the option names passed to the top Libero BUILD_BVF_GATEWATE.tcl script. These directory names are the option name specified in the bitstream builder’s build option YAML files.

The gateware is extended or customized by creating additional directories within the component directory of interest. For example, add a MY_CUSTOM_CAPE directory under the CAPE directory to add a gateware build option to support a custom cape.

### 5.7.4 Gateware Component Directories

The component directory contains subdirectories for:

- Constraint files
- Device tree overlay
- Optional HDL/Verilog source code

### Gateware TCL Scripts

The component directory contains the TCL scripts executed by Libero to generate the gateware. The TCL script framework executes a hand-crafted ADD_<COMPONENT_NAME>.tcl script which instantiates the component and stiches it to the base RISC-V subsystem and top level IOs. The other TCL scripts are typically IP configuration scripts and SmartDesign stiching scripts.

### 5.7.5 Opening the gateware as a libero project

It can be slightly difficult to explore the gateware design through the TCL files. To inspect the gateware design in detail easily, you can open the gateware as a Libero project. This is done by running the following command in the gateware directory:

```
python build-bitstream.py ./build-options/default.yaml # build option...
```

You will need to have all microchip tools installed and the environment variables set up correctly. Refer to the `microchip tools installation guide<mchp-fpga-tools-installation-guide>` for information on how to install these tools.
5.8 Customize BeagleV-Fire Cape Gateware Using Verilog

This document describes how to customize gateware attached to BeagleV-Fire’s cape interface using Verilog as primary language. The methodology described can also be applied when using other HDL languages.

It will describe:

- How to generate programming bitstreams without requiring the installation of the Libero FPGA toolchain on your development machine.
- How to use the cape Verilog template
- How to use the git.beagleboard.org CI infrastructure to generate programming bitstreams for your custom gateware

Steps:

1. Fork BeagleV-Fire gateware repository on git.beagleboard.org
2. Create a custom gateware build option
3. Rename a copy of the cape gateware Verilog template
4. Customize the cape’s Verilog source code
5. Commit and push changes to your forked repository
6. Retrieve the forked repositories artifacts
7. Program BeagleV-Fire with your custom bitstream
5.8.1 Fork BeagleV-Fire Gateware Repository

**Important:** All new users need to be manually approved to protect from BOT spam. You will not be able to fork the Gateware Repository until you have been approved. A request to the forum may expedite the process.

Navigate to BeagleV-Fire’s gateware source code repository. Click on the Forks button on the top-right corner.

On the Fork Project page, select your namespace and adjust the project name to help you manage multiple custom gateware (e.g. my-lovely-gateware). Click the Fork project button.

Clone the forked repository

```bash
git clone git@git.beagleboard.org:<MY-NAMESPACE>/my-lovely-gateware.git
```

Where `<MY-NAMESPACE>` is your Gitlab username or namespace.

5.8.2 Create A Custom Gateware Build Option

BeagleV-Fire’s gateware build system uses “build configuration” YAML files to describe the combination of gateware components options that will be used to build the gateware programming bitstream. You need to create one such file to describe to the gateware build system that you want your own custom gateware to be built. You need to have one such file describing your gateware in directory `custom-fpga-design`.

Let’s modify the `.custom-fpga-design/my_custom_fpga_design.yaml` build configuration file to specify that your custom cape gateware should be included in the gateware bitstream. In this instance will call our custom cape gateware `MY_LOVELY_CAPE`.
HSS:

type: git
link: https://git.beagleboard.org/beaglev-fire/hart-software-services.git
branch: develop-beaglev-fire
board: bvf
gateware:

type: sources
build-args: "M2_OPTION:NONE CAPE_OPTION:MY_LOVELY_CAPE" #
unique-design-version: 9.0.2

① On the gateware build-args line, replace VERILOG_TUTORIAL with MY_LOVELY_CAPE.

Note: The custom-fpga-design directory has a special meaning for the Beagleboard Gitlab CI system. Any build configuration found in this directory will be built by the CI system. This allows generating FPGA programming bitstreams without the requirement for having the Microchip FPGA toolchain installed on your computer.

5.8.3 Rename A Copy Of The Cape Gateware Verilog Template

Move to the cape gateware source code

cd my-lovely-gateware/sources/FPGA-design/script_support/components/CAPE

Create a directory that will contain your custom cape gateware source code

mkdir MY_LOVELY_CAPE

Copy the cape Verilog template

cp -r VERILOG_TEMPLATE/* ./MY_LOVELY_CAPE/

5.8.4 Customize The Cape’s Verilog Source Code

Move to your custom gateware source directory

cd MY_LOVELY_CAPE

You will need to first edit the ADD_CAPE.tcl TCL script to use your source code within your custom gateware directory and not the Verilog template source code. In this example this means using source code within the MY_LOVELY_CAPE directory rather the VERILOG_TEMPLATE directory.

Edit ADD_CAPE.tcl

Replace VERILOG_TEMPLATE with MY_LOVELY_CAPE in file ADD_CAPE.tcl.

# Import HDL source files
#----------------------------------------------
# (continues on next page)
Add the path to your additional Verilog source code files.

```vhd
① In our case we will be adding a new Verilog source file called blinky.v.
You will only need to revisit the content of ADD_CAPE.tcl if you want to add more Verilog source files or want to modify how the cape interfaces with the rest of the gateware (RISC-V processor subsystem, clock and reset blocks).

**Customize The Cape’s Verilog source code**

We will add a simple Verilog source file, `blinky.v`, in the `MY_LOVELY_CAPE` directory. Code below:

```vhd
`timescale 1ns/100ps

module blinky(
  input  clk,
  input  resetn,
  output blink
);

reg [22:0] counter;

assign blink = counter[22];

always @(posedge clk or negedge resetn)
begin
  if(~resetn)
```
begin
  counter <= 23'h0;
end
else
  begin
    counter <= counter + 23'h1;
  end
endmodule

Let's connect the blinky Verilog module within the cape by editing the CAPE.v file.

Add the instantiation of the blinky module:

```verilog
P9_41_42_IOPADS P9_41_42_IOPADS_0(
  .GPIO_OE ( GPIO_OE_const_net_3 ),
  .GPIO_OUT ( GPIO_OUT_const_net_3 ),
  .GPIO_IN  ( ),
  .P9_41   ( P9_41 ),
  .P9_42   ( P9_42 )
);

blinky blinky_0( //
  .clk   ( PCLK ),  //
  .resetn ( PRESETN ), //
  .blink ( BLINK )   //
);
```

1. Create a blinky module instance called blinky_0.
2. Connect the clock using the existing PCLK wire.
3. Connect the reset using the existing PRESETN wire.
4. Connect the blinky's blink output using the BLINK wire. This BLINK wire needs to be declared.

Add the BLINK wire:

```verilog
wire PCLK;
wire PRESETN;
wire BLINK; //
wire [31:0] APB_SLAVE_PRDATA_net_0;
wire [27:0] GPIO_IN_net_1;
```

5. Create a wire called BLINK.

The BLINK wire will be used to connect the blinky module's output to a top level output connected to an LED. Do you see where this is going?

Now for the complicated part. We are going to change the wiring of the bi-directional buffers controlling the cape I/Os including the user LEDs.

The original code populates two 43 bits wide wires for controlling the output-enable and output values of the P8 cape connector I/Os. The bottom 28 bits being controlled by the microprocessor subsystem’s GPIO block.

(continues on next page)
We are going to hijack the 6th I/O with our blinky’s output:

```verilog
// Concatenation assignments
assign GPIO_OE_net_0 = { 16'h0000, GPIO_OE[27:6], 1'b1, GPIO_OE[4:0] }; // ①
assign GPIO_OUT_net_0 = { 16'h0000, GPIO_OE[27:6], BLINK, GPIO_OE[4:0] }; // ②
```

① Tie high the output-enable of the 6th bit to constantly enable that output.
② Control the 6th output from the blink module through the WIRE wire.

**Edit The Cape’s Device Tree Overlay**

You should always have a device tree overlay associated with your gateware even if there is limited control from Linux. The device tree overlay is very useful to identify which gateware is currently programmed on your BeagleV-Fire.

```dts
/dts-v1/
/plugin/

{/chosen} { overlays {
  MY-LOVELY-CAPE-GATEWARE = "GATEWARE_GIT_VERSION"; // ①
}
};
```

① Replace VERILOG-CAPE-GATEWARE with MY-LOVELY-CAPE-GATEWARE.

This change will result in MY-LOVELY-CAPE-GATEWARE being visible in `/proc/device-tree/chosen/overlays` at run-time, allowing to check that my lovely gateware is successfully programmed on BeagleV-Fire.

**5.8.5 Commit And Push Changes To Your Forked Repository**

Move back up to the root directory of your gateware project. This is the my-lovely-gateware directory in our current example.

Add the `my-lovely-gateware/sources/FPGA-design/script_support/components/CAPE/MY_LOVELY_CAPE` directory content to your git repository.

```
    git add sources/FPGA-design/script_support/components/CAPE/MY_LOVELY_CAPE/
    git commit -m "Add my lovely gateware."
    git push
```

Push changes to your beagleboard Gitlab repository:
5.8.6 Retrieve The Forked Repositories Artifacts

Navigate to your forked repository. Click Pipelines in the left pane then the Download Artifacts button on the right handside. Select build-job:archive. This will result in an artifacts.zip file being downloaded.

![Fig. 5.3: gateware pipeline](image)

5.8.7 Program BeagleV-Fire With Your Custom Bitstream

Unzip the downloaded artifacts.zip file. Go to the gateware-builds-tester/artifacts/bitstreams directory:

```
cd gateware-builds-tester/artifacts/bitstreams
```

On your Linux host development computer, use the scp command to copy the bitstream to BeagleV-Fire home directory, replacing `<IP_ADDRESS>` with the IP address of your BeagleV-Fire.

```
scp -r .:/my_custom_fpga_design beagle@<IP_ADDRESS>:/home/beagle/
```

On BeagleV-Fire, initiate the reprogramming of the FPGA with your gateware bitstream:

```
sudo /usr/share/beagleboard/gateware/change-gateware.sh ./my_custom_fpga_design
```

Wait for a couple of minutes for the BeagleV-Fire to reprogram itself.

You will see the 6th user LED flash once the board is reprogrammed. That’s the Verilog you added blinking the LED.

On BeagleV-Fire, You can check that your gateware was loaded using the following command to see the device tree overlays:

```
tree /proc/device-tree/chosen/overlays/
```

![Fig. 5.4: gateware lovely overlay](image)
5.9 Exploring Gateware Design with Libero

In this demonstration, we’ll be exploring the BeagleV-Fire gateware in the Libero Design Suite, making changes to the default gateware. This demo will serve as an introduction to the design tool, an alternative method for developing gateware.

5.9.1 Prerequisites

The prerequisites required for creating the libero project locally are:

1. Microchip design tools: Refer to the document here for installation instructions of microchip FPGA tools.

2. Python requirements for gateware build scripts:

   `pip3 install gitpython
   pip3 install pyyaml`

3. Environment variables: The following environment variables are required for compilation

   - SC_INSTALL_DIR
   - FPGENPROG
   - LIBERO_INSTALL_DIR
   - LM_LICENSE_FILE

   A script is provided for setting up these variables in the fpga tools installation section. An example script for setting up the environment is available here.

4. It is highly recommended to go through the Customize BeagleV-Fire Cape Gateware Using Verilog tutorial to understand the basics of the gateware structure.

5.9.2 Cloning and Building the Gateware

First, we must source the environment to include the microchip tools.

```
source /path/to/microchip/fpga/tools/setup-microchip-tools.sh
```

Next, we’ll clone the gateware repository to get a local copy of the project.

```
git clone https://openbeagle.org/beaglev-fire/gateware.git
cd gateware
```

We can use the `build-gateware` script to generate a libero project for us, where we can start making our changes.

```
python build-gateware.py ./build-options/default.yaml # exploring the...
```

This should start a big log stating the compilation of the project. First, the device tree overlays are compiled, which contain information for linux about the gateware being compiled.

Next, the Hart Software Services (HSS) is compiled. This acts as a zero-stage bootloader, configuring the Polarfire SoC and allowing services like loading the next stage bootloader and flashing the eMMC of the board.

Then the libero project generating is started. Here, TCL scripts inside the sources directory are executed, starting with `BUILD_BVF_GATEWARE.tcl` script. This stitches each HDL module, IP, hardware configuration in the gateware.
Once bitstream generation is completed, the Libero project is ready to be opened. Start Libero on the same terminal in linux, or from the start menu in Windows, and open the project file by pressing CTRL+O and selecting the generated project as gateware/work/libero/BVF_GATEWARE_025T.prjx.

![Libero project location](image)

**5.9.3 Exploring The Design**

Let the IDE load everything, and then you’re all set to browse around! You can go to the Design Hierarchy view to look at all Smart Design components. Here, all the gateware components are listed in block-like views. Select the DEFAULT_****** option in the hierarchy to have a look at the whole gateware. You should also be able to see the cape, M.2 interface and the RISC-V subsystem modules. These modules are explained in Gateware Introduction.

**5.9.4 Adding Custom HDL**

Once you’re done exploring, we can start by adding our first HDL to the design. Create a new HDL file through the menu bar, and name it blinky. Once created, you can find the HDL file under the User HDL Source Files heading in the Design Hierarchy.

Next, add this code to the file

```verilog
`timescale 1ns/100ps
module blinky(
    input clk,
    input resetn,
    input [27:0] gpio_out,
)` (continues on next page)
Fig. 5.6: Libero gateware overview

Fig. 5.7: Adding new HDL

5.9. Exploring Gateware Design with Libero
input [27:0] gpio_enable,
output [27:0] modified_gpio,
output [27:0] modified_gpio_enable
);

reg [22:0] counter;
assign modified_gpio = {gpio_out[27:6], counter[22], gpio_out[4:0]};
assign modified_gpio_enable = {gpio_enable[27:6], 1'b1, gpio_enable[4:0]};

always@(posedge clk or negedge resetn)
begin
  if (~resetn)
    counter <= 23'h0;
  else
    counter <= counter + 23'h1;
end
endmodule

After adding it, press the Build Hierarchy button in the Design Hierarchy window to refresh it and bring the added HDL to the work directory. Right click on it to select the “Create Core from HDL....” option. Press No on the dialog that follows since we’ve described the ports perfectly in our HDL.

Now, open the CAPE design under the DEFAULT_**** smart design, to have a look at what’s in the cape. Drag and drop the blinky file appearing in the work section into the cape design. You will have successfully instantiated the new verilog file into the cape smart design.

Making The Connections

You should see the blinky module within the CAPE design, and it should be fairly obvious where we’re going to be connecting the module if you’ve gone through the previous demo. First, delete the wires connecting the GPIO_OUT and GPIO_OE to the CAPE_DEFAULT_GPIOS module. Then, simply connect the GPIO_OUT and the GPIO_OE terminals of the cape to the gpio_out and the gpio_enable pins respectively. Similarly connect the outputs of the blinky module to the CAPE_DEFAULT_GPIOS module.

Finally, connect the CLK and the RESET pins to the PCLK and the PRESETN pins below in the cape. You can use the compress layout button in the toolbar to make the design neat once you’re done connecting the wires.

Go ahead and save the CAPE file. You can also verify the design by pressing the checkmark icon in the editor toolbar. Now, it’s time to export our design back to the gateware repository.

5.9.5 Exporting The Design

Exporting the Cape

The SmartDesigns you have changed should show an “i” icon in front of them indicating that they need to be regenerated. First, regenerate the designs by right clicking on them and selecting “Generate Component”. Rebuild the Hierarchy too as we’ve done before.

Next, right-click on the cape and select “Export Component Description (TCL)” to export it as a script which can be used in the gateware repository. I suggest creating an export directory where you can temporarily store the exported gateware files before getting them into the repository.

Now, simply copy it into the gateware at the following path.
Fig. 5.8: Create core from HDL
Fig. 5.9: Add blinky to cape

Fig. 5.10: Connect blinky to cape
Fig. 5.11: Regenerate designs
Exporting The HDL

To add new HDL to the gateware repository, first we need to copy it to the HDL directory at `gateware/sources/FPGA-design/script_support/HDL`. You can do that by just creating a folder named `blinky` inside and copying the HDL to it.

```bash
cp ~/export/gateware/blinky.v ~/gateware/sources/FPGA-design/script_support/HDL/BLINKY/
```

Now, to add the TCL script to import this design for the CAPE scripts, we can export the script by right-clicking on the HDL file in the Design Hierarchy and select `Export Component Description`.

Now, copy the contents of this exported file to our gateware’s HDL sourcing script at `gateware/sources/FPGA-design/script_support/hdl_source.tcl`.

First, copy the contents of the exported TCL file to the bottom of the file. Replace the `-file` argument in the line with `-file $project_dir/hdl/blinky.v`. Finally, source the file by add a line below line no. 11 as:

```tcl
-hdl_source {script_support/HDL/AXI4_address_shim/AXI4_address_shim.v}
-hdl_source {script_support/HDL/BLINKY/blinky.v}
```

Verify your script as above, save it and now you’re good to compile your project!

Go ahead and run the python script to build the gateware and verify your changes to the gateware.

```bash
gatewaredi python build-bitstream.py ./build-options/default.yaml # run this in the gateware di
```

If at any point the compilation fails, you can debug the script at the mentioned line. If it compiles successfully, it will mention by saying:

```
The Execute Script command succeeded.
The BVF_GATEWARE_025T project was closed.
```

Now, you can commit the changes to your gateware repository fork, download the artifacts after compilation, and program the gateware using the `change_gateware.sh` script. Have fun!
Fig. 5.12: Export HDL
Chapter 6

Support

All support for BeagleV Fire design is through BeagleBoard.org community at BeagleBoard.org forum.

6.1 Production board boot media

6.2 Certifications and export control

6.2.1 Export designations

- HS: 8471504090
- US HS: 8543708800
- EU HS: 8471707000

6.2.2 Size and weight

- Bare board dimensions: 86.38*54.61*18.8mm
- Bare board weight: 45.8g
- Full package dimensions: 140 x 100 x 40 mm
- Full package weight: 106g

6.3 Additional documentation

6.3.1 Hardware docs

For any hardware document like schematic diagram PDF, EDA files, issue tracker, and more you can checkout the BeagleV-Fire design repository.

6.3.2 Software docs

For BeagleV-Fire specific software projects you can checkout all the BeagleV-Fire project repositories group.
6.3.3 Support forum

For any additional support you can submit your queries on our forum, https://forum.beagleboard.org/tags/c/beaglev/15/fire

6.3.4 Pictures

6.4 Change History

**Note:** This section describes the change history of this document and board. Document changes are not always a result of a board change. A board change will always result in a document change.

6.4.1 Board Changes

For all changes, see https://git.beagleboard.org/beaglev-fire/beaglev-fire/. Versions released into production are noted below.

<table>
<thead>
<tr>
<th>Rev</th>
<th>Changes</th>
<th>Date</th>
<th>By</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Initial production version</td>
<td>2023-11-02</td>
<td>JK</td>
</tr>
</tbody>
</table>