Emp-Fwk Guide

Release FW 0.8.2 / SW 0.8.2

EMP Collaboration

Rev2.0: May 14th, 2019

TABLE OF CONTENTS

Scope of this Document

The EMP framework provides common infrastructural firmware - as well as corresponding control and monitoring software - to support arbitrary user-created payload firmware, e.g. the reconstruction algorithm in a level-1 track finder or level-1 trigger board.

The scope of this infrastructural firmware and software includes:

- a control bus master;
- TTC generation and reception;
- the transmission and reception of data over high-speed electrical links and optical links;
- the use of the datapath buffers to load user-specific data patterns over the control bus, play this input data into the payload, capture payload's output data, and read that over the control bus

Table of contents

CHAPTER

ONE

THE BASIC IDEA

The `emp-fwk` repo contains top-level designs for various different FPGAs and boards. Each of these designs instantiates an entity - named `emp_payload` - and connects its input/output ports to the clocking infrastructure, control bus, and input/output buffers. Parameters that might have to be changed between different algorithms or different FPGAs are specified as constants in a VHDL package `emp_project_decl`.

EMP stands for Extensible, Modular (data) Processor.

CHAPTER

TWO

WALKTHROUGHS FOR FIRST USERS

The following are a set of walkthroughs to demonstrate the functionality and usability of the EMP framework. To begin with (*Prerequisites*) the required software is downloaded and installed. The walkthroughs demonstrate the following functionality:

1. walkthrough1 - Create a repository and a *null algorithm* (copy input buffers to output) in the users's own development area. Key points:
    - create repo
    - synthesis
    - build firmware (bitfile)
2. walkthrough2 - Extend the implementation to a simple custom algorithm payload to add 5 to the input buffers. Demonstrating how to:
    - implement VHDL algo
    - simulate
    - build and upload firmware
    - interrogate buffers (both simulated and in actual hardware)
3. walkthrough3: Create a modular payload entity which demonstrates how to combine distributed components within `emp_payload.vhd` and to instantiate them whithin the dependency files.
    - This walkthrough will appear in the future.
4. walkthrough4: Create an algorithm written in HLS (high level synthesis) and implement it within the firmware - key points:
    - Introduce C++ code to your algorithm
    - Create an IP-core for instantiating within EMP
    - Add the HLS algo to EMP and implement on an FPGA
5. walkthrough5: Simulate the firmware that you created in walkthrough4. ModelSim can be used to simulate the performance of an FPGA without having to carry out full synthesis, implementation and deployment. This walkthrough will demonstrate how to:
    - Create from EMP the vivado project for simulation purposes
    - Include the algorithm and carry out test buffer injection and observe the result (output).

CHAPTER

THREE

EMP FIRMWARE

The pages listed below explain how to build EMP-based firmware design, and specify the interfaces between the key EMP firmware components and the payload.

3.1 Firmware: Build and simulation instructions

This page outlines the commands required in order to build or simulate your payload firmware within the EMP framework.

Note: This page has been written for people who already have payload firmware that is integrated into the EMP framework. If you are a new user building your payload firmware within the EMP framework for the first time, then you should follow the *walkthroughs* instead. If your project is set up to use the MP7 framework, and you want to migrate it to the EMP framework, you might want to go to the payload_firmware_mp7_migration page instead.

The prerequisites and steps 1 to 2 below are the same for building a bitfile and for simulating your payload; the commands in those two workflows are only different for step 3 (creating the project area, and running Vivado/ModelSim).

3.1.1 Prerequisites

You should create a working directory on a Linux machine which has *Vivado* installed on it, as well as *ModelSim* for simulations. The machine should also have Python and pip installed on it. In addition, in order for the TCDS2 functionality to be correctly implemented and for the command `ipbb ipbus gendecoders` to run, **uHAL** needs to be installed.

Build tool

The easiest way to build the framework firmware is to use IPBB; in case you haven't used it before, the IPBB primer page gives a short introduction to IPBB. In order to run IPBB the package `python3-virtualenv` (or `python3-venv`) needs to be installed on your system.

IPBB can be downloaded as follows:

```
curl -L https://github.com/ipbus/ipbb/archive/dev/2023a.tar.gz | tar xvz
```

... and then you just need to source the `env.sh` script in order to add the `ipbb` command to your `$PATH`:

```
source ipbb-dev-2023a/env.sh
# To run the ``ipbus gendecoders`` command later on
export PATH=/opt/cactus/bin/uhal/tools:$PATH LD_LIBRARY_PATH=/opt/cactus/lib:$LD_LIBRARY_
↪PATH
```

Vivado

The framework firmware has been developed and tested with Vivado versions from 2021.2 to 2022.2

3.1.2 Step 1: Setup your IPBB area

You should begin by creating the IPBB work area itself, then download the framework firmware and its dependencies. You need to be a member of the e-group `emp-fwk-users` to download the EMP framework git repo and the e-group `cms-tcds2-users` to download the TCDS2 firmware. You will need to have a valid CERN *Kerberos ticket* on the machine on which you are running IPBB; if you don't then an error will appear during the `ipbb add git` commands. You can change the URL prefix `https://:@gitlab.cern.ch:8443` to `https://gitlab.cern.ch` in order to use your CERN username and password directly instead of Kerberos tokens.

The commands to clone the repositories using Kerberos authentication are:

```
ipbb init algo-work
cd algo-work
ipbb add git https://:@gitlab.cern.ch:8443/p2-xware/firmware/emp-fwk.git -b v0.8.2
ipbb add git https://gitlab.cern.ch/ttc/legacy_ttc.git -b v2.1
ipbb add git https://:@gitlab.cern.ch:8443/cms-tcds/cms-tcds2-firmware.git -b v0_1_1
ipbb add git https://gitlab.cern.ch/HPTD/tclink.git -r fda0bcf
ipbb add git https://gitlab.cern.ch/dth_p1-v2/slinkrocket_ips.git -b v03.12
ipbb add git https://:@gitlab.cern.ch:8443/dth_p1-v2/slinkrocket.git -b v03.12
ipbb add git https://github.com/ipbus/ipbus-firmware -b v1.9

# If using GBT/lpGBT cores, also run:
ipbb add git https://gitlab.cern.ch/gbt-fpga/gbt-fpga.git -b gbt_fpga_6_1_0
ipbb add git https://gitlab.cern.ch/gbt-fpga/lpgbt-fpga.git -b v.2.1
ipbb add git https://:@gitlab.cern.ch:8443/gbtsc-fpga-support/gbt-sc.git -b gbt_sc_4_3
```

Alternative git commands with manual CERN username/password authentication:

```
ipbb init algo-work
cd algo-work
ipbb add git https://gitlab.cern.ch/p2-xware/firmware/emp-fwk.git -b v0.8.2
ipbb add git https://gitlab.cern.ch/ttc/legacy_ttc.git -b v2.1
ipbb add git https://gitlab.cern.ch/cms-tcds/cms-tcds2-firmware.git -b v0_1_1
ipbb add git https://gitlab.cern.ch/HPTD/tclink.git -r fda0bcf
ipbb add git https://gitlab.cern.ch/dth_p1-v2/slinkrocket_ips.git -b v03.12
ipbb add git https://gitlab.cern.ch/dth_p1-v2/slinkrocket.git -b v03.12
ipbb add git https://github.com/ipbus/ipbus-firmware -b v1.9

# If using GBT/lpGBT cores, also run:
ipbb add git https://gitlab.cern.ch/gbt-fpga/gbt-fpga.git -b gbt_fpga_6_1_0
ipbb add git https://gitlab.cern.ch/gbt-fpga/lpgbt-fpga.git -b v.2.1
ipbb add git https://gitlab.cern.ch/gbtsc-fpga-support/gbt-sc.git -b gbt_sc_4_3
```

(A different name than `algo-work` can be used for working area as desired.)

```
(ipbb) [ahoward@greg-special algo-work]$ ipbb add git https://:@gitlab.cern.ch:8443/p2-xware/firmware/emp-fwk.git -b v0.
2.3
Adding git repository https://:@gitlab.cern.ch:8443/p2-xware/firmware/emp-fwk.git
ERROR ('ErrorReturnCode_128' exception): '

  RAN: /usr/bin/git ls-remote -h -t https://:@gitlab.cern.ch:8443/p2-xware/firmware/emp-fwk.git v0.2.3

  STDOUT:


  STDERR:
fatal: Authentication failed for 'https://:@gitlab.cern.ch:8443/p2-xware/firmware/emp-fwk.git/'
'

File "/home/ahoward/IPPB/ipbb-0.4.2/external/ipbb/lib/python2.7/site-packages/sh.py", line 815, in handle_command_exit_c
ode
    raise exc
Exception in user code:
------------------------------------------------------------
Traceback (most recent call last):
  File "/home/ahoward/IPPB/ipbb-0.4.2/src/ipbb/scripts/builder.py", line 195, in main
    climain()
  File "/home/ahoward/IPPB/ipbb-0.4.2/external/ipbb/lib/python2.7/site-packages/click/core.py", line 764, in __call__
    return self.main(*args, **kwargs)
  File "/home/ahoward/IPPB/ipbb-0.4.2/external/ipbb/lib/python2.7/site-packages/click/core.py", line 717, in main
    rv = self.invoke(ctx)
  File "/home/ahoward/IPPB/ipbb-0.4.2/external/ipbb/lib/python2.7/site-packages/click/core.py", line 1137, in invoke
    return _process_result(sub_ctx.command.invoke(sub_ctx))
  File "/home/ahoward/IPPB/ipbb-0.4.2/external/ipbb/lib/python2.7/site-packages/click/core.py", line 1137, in invoke
    return _process_result(sub_ctx.command.invoke(sub_ctx))
  File "/home/ahoward/IPPB/ipbb-0.4.2/external/ipbb/lib/python2.7/site-packages/click/core.py", line 956, in invoke
    return ctx.invoke(self.callback, **ctx.params)
  File "/home/ahoward/IPPB/ipbb-0.4.2/external/ipbb/lib/python2.7/site-packages/click/core.py", line 555, in invoke
    return callback(*args, **kwargs)
  File "/home/ahoward/IPPB/ipbb-0.4.2/external/ipbb/lib/python2.7/site-packages/click/decorators.py", line 27, in new_fu
nc
    return f(get_current_context().obj, *args, **kwargs)
  File "/home/ahoward/IPPB/ipbb-0.4.2/src/ipbb/cli/repo.py", line 37, in git
    git(env, repo, branch, dest)
  File "/home/ahoward/IPPB/ipbb-0.4.2/src/ipbb/cli/impl/repo.py", line 72, in git
    llsRemote = sh.git('ls-remote', '-h', '-t', repo, branch)
  File "/home/ahoward/IPPB/ipbb-0.4.2/external/ipbb/lib/python2.7/site-packages/sh.py", line 1427, in __call__
    return RunningCommand(cmd, call_args, stdin, stdout, stderr)
  File "/home/ahoward/IPPB/ipbb-0.4.2/external/ipbb/lib/python2.7/site-packages/sh.py", line 774, in __init__
    self.wait()
  File "/home/ahoward/IPPB/ipbb-0.4.2/external/ipbb/lib/python2.7/site-packages/sh.py", line 792, in wait
    self.handle_command_exit_code(exit_code)
  File "/home/ahoward/IPPB/ipbb-0.4.2/external/ipbb/lib/python2.7/site-packages/sh.py", line 815, in handle_command_exit
_code
    raise exc
ErrorReturnCode_128:

  RAN: /usr/bin/git ls-remote -h -t https://:@gitlab.cern.ch:8443/p2-xware/firmware/emp-fwk.git v0.2.3

  STDOUT:


  STDERR:
fatal: Authentication failed for 'https://:@gitlab.cern.ch:8443/p2-xware/firmware/emp-fwk.git/'


------------------------------------------------------------
(ipbb) [ahoward@greg-special algo-work]$
```

Fig. 3.1: Git authentication error with incorrectly set-up Kerberos tokens.

3.1.3 Step 2: Add the payload repository

Add the repository containing your payload firmware to your IPBB work area by running the `ipbb add` command again.

If you don't already have a git repo and project, then you can create you own algorithm repository as described in walkthrough1.

3.1.4 Step 3: Build firmware or run simulation

The commands required for either building a bitfile or simulating your payload firmware are outlined in the sections below - you don't need to follow the commands from both sections. Make sure that you have a valid Vivado licence (and ModelSim licence for the simulation workflow) in order for the commands to execute correctly.

Note: The top-level `.dep` file that you use in the following sections must reference your files that implement:

1. The `emp_payload` entity
2. The `emp_project_decl` package

If simulating your payload firmware, your top-level `.dep` file must also reference a file that implements the `tb_decl` package. The expected `emp_payload` port map and lists of required constants for the packages are specified in the *Payload firmware: Interface* page.

Building the firmware

Assuming that your top-level `.dep` file is located in source area `my-algo-repo`, at path `an-algo/firmware/cfg/top.dep`, you can create a project area (under directory `proj/my_algo`) by running:

```
ipbb proj create vivado my_algo my-algo-repo:an-algo top.dep
cd proj/my_algo
```

Only three more commands are required to setup the project, then build and package the bitfile (make sure *Vivado* is correctly set in your `$PATH`):

```
ipbb vivado generate-project
ipbb vivado synth -j4 impl -j4
ipbb vivado package
```

During synthesis and implementation a number of checks are carried out, therefore you should make sure that the command is executed successfully before loading the bitfile on your FGPA. Also note that the synthesis stage may take 30-60 minutes to complete depending upon your machine.

Simulating the payload firmware

Assuming that your top-level `.dep` file is located in source area `my-algo-repo`, at path `an-algo/firmware/cfg/top_sim.dep`, you can create a project area (under directory `proj/my_algo`) by running:

```
ipbb proj create sim my_algo my-algo-repo:an-algo top_sim.dep
cd proj/my_algo
```

You then need to create the simulation libraries as follows - by default, the simulation libraries will be created under `${HOME}/.xilinx_sim_lib`, but they can be saved under another path by setting the `IPBB_SIMLIB_BASE` environment variable, or by using the `-x` flag (e.g. `-x /data/my-simlib-dir`):

```
ipbb sim setup-simlib
ipbb sim ipcores
```

The ModelSim project should then be created:

```
ipbb sim generate-project
```

You can then run the simulation, in which data from input file <`input.txt`> is fed into the algorithm, and the data captured from the output channels is written to <`out.txt`>:

```
./run_sim -c work.top -Gsourcefile=<input.txt> -Gsinkfile=<out.txt> -do 'run 5us' -do
↪'quit'
```

The format for the input and output files here is the same as that used by the EMP software when injecting data into, and capturing data from, the input/output channels of EMP-based firmware running in an FPGA.

Note: The duration used in the last command (5 microseconds in the example given above) should be at least as long as the amount of time that you want the algorithm to be simulated (i.e. payload clock period multiplied by the larger of `(WAIT_CYCLES_AT_START + PLAYBACK_LENGTH)` and `(WAIT_CYCLES_AT_START + CAPTURE_OFFSET + CAPTURE_LENGTH)`)

3.2 Payload firmware: Interface

The EMP framework top-level designs instantiate an entity - named `emp_payload` - and connects its input/output ports to the clocking infrastructure, control bus, and input/output buffers, as described in the introduction page. Various configurable aspects of the infrastructural firmware (such as the I/O clock frequency and the depths of the I/O latency buffers) are controlled by the values of constants defined in the `emp_payload_decl` package. This page outlines requirements on the port map of the `emp_payload` entity, and lists the constants that users must define in the `emp_project_decl` and `tb_decl` packages.

3.2.1 `emp_payload` entity

If you are developing a custom "physics algorithm" payload to integrate into the EMP framework designs, this payload should be implemented in an entity named `emp_payload`, with the following input and output ports:

	Name	Type	Description
in	`clk`	`std_logic`	IPbus clock
in	`rst`	`std_logic`	IPbus reset
in	`ipb_in`	`ipbus.ipb_wbus`	IPbus fabric (in)
out	`ipb_out`	`ipbus.ipb_rbus`	IPbus fabric (out)
in	`clk_payload`	`std_logic_vector(2 downto 0)`	Auxiliary clocks for algorithms
in	`rst_payload`	`std_logic_vector(2 downto 0)`	Resets aligned to auxiliary clocks
in	`clk_p`	`std_logic`	Clock for main I/O data
in	`rst_loc`	`std_logic_vector(N_REGION - 1 downto 0)`	Region-local reset (clock domain: `clk_p`)
in	`clken_loc`	`std_logic_vector(N_REGION - 1 downto 0)`	
in	`ctrs`	`emp_ttc_decl.ttc_stuff_array`	TTC counters
out	`bc0`	`std_logic`	
in	`d`	`ldata(4 * N_REGION - 1 downto 0)`	Input data channels
out	`q`	`ldata(4 * N_REGION - 1 downto 0)`	Output data channels
out	`gpio`	`std_logic_vector(29 downto 0)`	
out	`gpio_en`	`std_logic_vector(29 downto 0)`	

3.2.2 `emp_project_decl` package

The `emp_project_decl` package must define the following constants:

Name	Type	Description
LHC_BUNCH_COUNT	integer	Number of LHC bunches (should be 3564)
LB_ADDR_WIDTH	integer	Address width for I/O channel latency buffers (i.e. if 10, they can store 2^10 = 1024 words). *Only required for builds, not for simulation.*
CLOCK_COMMON_RATIO	integer	Defines MMCM VCO frequency (multiple of 40MHz) in TTC block Must be multiple of CLOCK_RATIO and all elements of CLOCK_AUX_RATIO
CLOCK_RATIO	integer	Determines clock frequency for I/O channels (i.e. clk_p port) Frequency = CLOCK_RATIO * 40MHz
CLOCK_AUX_RATIO	clock_ratio_array_t [From emp_framework_decl]	Determines frequency of 'auxilary' clocks (i.e. clk_payload port) Frequency of clk_payload(i) = CLOCK_AUX_RATIO(i) * 40MHz E.g. With constant CLOCK_AUX_RATIO : clock_ratio_array_t := (2, 4, 6);, the frequencies of clk_payload(0), (1) and (2) are 240, 160 and 80MHz respectively
REGION_CONF	region_conf_array_t [From emp_device_types]	Defines whether MGT logic and buffers are instantiated in each I/O region

Examples of the emp_project_decl package for various boards can be found in the emp-fwk repository, under the projects/example directory.

3.2.3 `tb_decl` package

The `tb_decl` package is only required when simulating the payload firmware; it must define the following constants:

Name	Type	Description
`SOURCE_FILE`	`string`	Default path to the source file (Used if `sourcefile` generic not specified at runtime)
`SINK_FILE`	`string`	Default path to the sink file (Used if `sinkfile` generic not specified at runtime)
`PLAYBACK_LENGTH`	`natural`	Number of frames to be played from source file
`CAPTURE_LENGTH`	`natural`	Number of frames to be captured in sink file
`WAIT_CYCLES_AT_START`	`natural`	"Quiet" time at the beginning of the simulation before data from source is injected
`PLAYBACK_OFFSET`	`natural`	Index of the first frame from the source file that will be injected into the payload
`CAPTURE_OFFSET`	`natural`	Number of clock cycles before the output data is captured
`PLAYBACK_LOOP`	`boolean`	Toggles playback of data in a continuous loop (rather than only playing the data once)
`STRIP_HEADER`	`boolean`	Toggle whether the header frame - i.e. the first valid frame of the packet - is removed before injecting data into the payload
`INSERT_HEADER`	`boolean`	Toggle whether a header frame is inserted before the first valid frame

An example of the `tb_decl` package can be found in the `emp-fwk` repository, in file `projects/examples/testbench/firmware/hdl/tb_decl.vhd`.

3.3 Backend links

Since version 0.7.0, the EMP firmware framework implements the CMS standard trigger link protocol (CSP) for links that transmit data between backend boards with low, fixed latency. This page describes the meaning of the fields in the payload's `d` and `q` ports for I/O channels connected to the backend link engine, how they should be used, and gives some related examples.

3.3.1 CSP protocol in a nutshell

The CSP protocol is used to transmit LHC-synchronous data over asynchronous links between backend boards. The throughput of the asynchronous links we use is slightly larger than that required for the LHC-syncronous data transmission, and so the spare bandwidth is used to ensure robustness and to carry link metadata. For 25G links, an EMP payload transmits/receives nine 64-bit words per LHC clock cycle (i.e. payload frequency is 360.xxMHz); for 16G links, an EMP payload transmits/receives six 64-bit words per LHC clock cycloe (i.e. payload frequency is 240.xxMHz).

There are two types of words on the link itself (i.e. after CSP encoding):

- Data words, carrying the LHC-sychronous data from/to payload firmware; and
- Filler words, added to reach the line rates

From the perspective of the payload firmware data is organised in packets, which are contiguous groups of data words. A CRC is calculated for each packet by the CSP transmitter firmware and sent in a filler word following the packet. The CRC is also calculated by the CSP transmitter firmware, and the number of mismatches - indicating bitflips in data words - are stored in one of the status registers. For links with time multiplexing period greater than 1, all the data contained in each packet should be associated with the same collision. Due to the small filler-to-data word ratio on a fully-utilised link, for a time multiplexing period of 1, data from two or three collisions must be transmitted in a single packet. If there are gaps of multiple clock cycles between subsequent packets, then additional filler words referred to as *idle words* are transmitted on the link inbetween the packets, in place of data words.

3.3.2 Payload port specification

For backend link channels, the meaning of the `ldata` fields in the payload's `d` and `q` ports are as follows:

`valid`
: Distinguishes data within packets vs the idle words sent between packets. Asserted on the first clock cycle in each packet until the last clock cycle in the packet. Remains high constantly when back-to-back packets are transmitted

`start`
: Asserted on first clock cycle in each packet.

`last`
: Asserted on last clock cycle in each packet.

`start_of_orbit`
: Asserted on the first clock cycle of the first packet of the orbit – i.e. the packet that contains data from BXn < M, where M is the TMUX period (e.g. the packet containing BX0 data on links in timeslace 0, the packet containing BX1 data on links in timeslice 1).

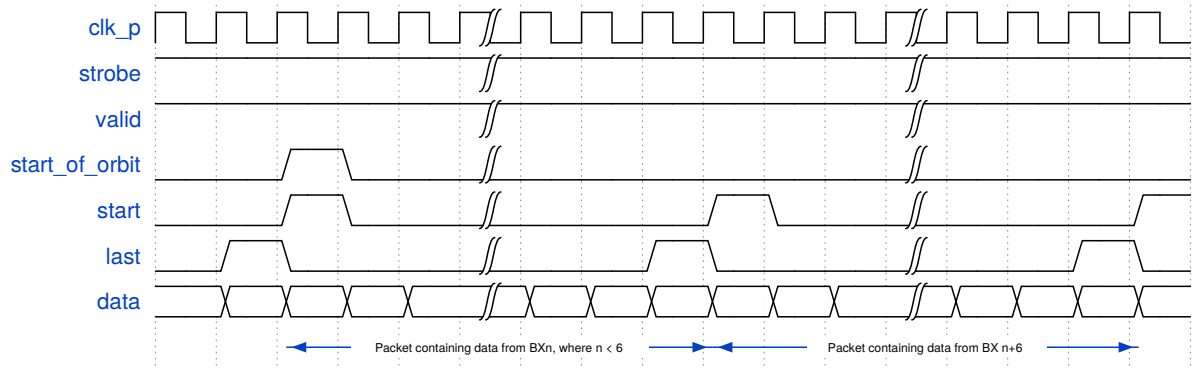
`data`
: 64-bit word of payload data received/transmitted inside packets. Undefined for clock cycles inbetween packets (when `valid` is low).

The `strobe` signal should currently always be asserted for backend links.

3.3.3 Example use cases

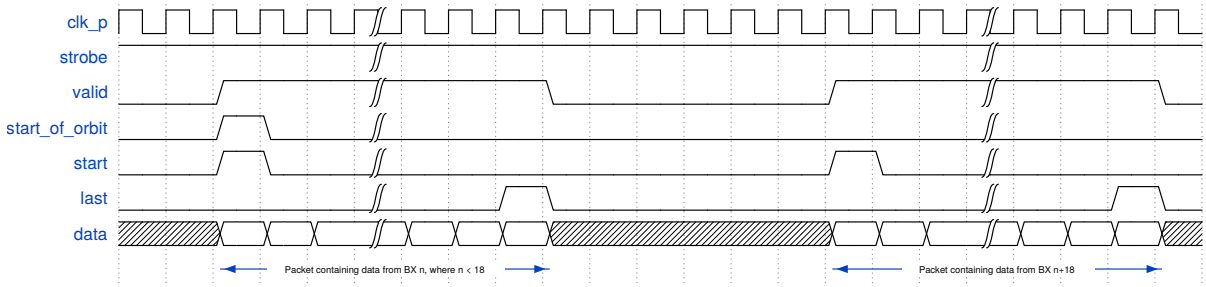
TM6 link with back-to-back packets

The diagram below shows the port values for a TM6 25G link carrying back-to-back packets (hence packets are 6 x 9 = 54 clock cycles in length):



TM18 link with gap between packets

The diagram below shows the port values for a TM18 25G link with a gap of 6 clock cycles between packets (hence packets are 18 x 9 - 6 = 156 clock cycles in length):



The following example payload entity implements this packet structure - with the data word set to a counter pattern composed from the channel index, packet index, and word index:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

use work.ipbus.all;
use work.emp_data_types.all;
use work.emp_project_decl.all;

use work.emp_device_decl.all;
use work.emp_ttc_decl.all;

use work.emp_slink_types.all;

entity emp_payload is
  port(
    clk         : in  std_logic;        -- ipbus signals
```

(continues on next page)

(continued from previous page)

```
    rst         : in  std_logic;
    ipb_in      : in  ipb_wbus;
    ipb_out     : out ipb_rbus;
    clk40       : in  std_logic;
    clk_payload : in  std_logic_vector(2 downto 0);
    rst_payload : in  std_logic_vector(2 downto 0);
    clk_p       : in  std_logic;        -- data clock
    rst_loc     : in  std_logic_vector(N_REGION - 1 downto 0);
    clken_loc   : in  std_logic_vector(N_REGION - 1 downto 0);
    ctrs        : in  ttc_stuff_array;
    bc0         : out std_logic;
    d           : in  ldata(4 * N_REGION - 1 downto 0);  -- data in
    q           : out ldata(4 * N_REGION - 1 downto 0);  -- data out
    gpio        : out std_logic_vector(29 downto 0);  -- IO to mezzanine connector
    gpio_en     : out std_logic_vector(29 downto 0);  -- IO to mezzanine connector␣
↪(three-state enables)
    slink_q : out slink_input_data_quad_array(SLINK_MAX_QUADS-1 downto 0);
    backpressure : in std_logic_vector(SLINK_MAX_QUADS-1 downto 0)
    );

end emp_payload;

architecture rtl of emp_payload is

begin

  -- This example code sends 156-word-long TMUX18 packets (i.e. same packet length as␣
↪track finder output)
  -- with channel index, packet index, and word index embedded in the data word
  gen : for i in N_REGION * 4 - 1 downto 0 generate

    -- Index of word within a packet
    signal word_index : std_logic_vector(7 downto 0) := x"00";
    -- Index of packet within an orbit
    signal packet_index : std_logic_vector(8 downto 0) := "000000000";

  begin

    process (clk_p)
    begin
      if rising_edge(clk_p) then
        -- Reset counters on receiving BC0 from TCDS
        if (ctrs(i/4).bctr = x"000") and (ctrs(i/4).pctr = "0000") then
          word_index <= x"00";
          packet_index <= "000000000";
        -- Reset word index and increment packet index every 162 clock cycles (TMUX18:␣
↪18BX * 9 clocks/BX)
        elsif word_index = x"A1" then
          word_index <= x"00";
          packet_index <= std_logic_vector(unsigned(packet_index) + 1);
        else
          word_index <= std_logic_vector(unsigned(word_index) + 1);
```

(continues on next page)

(continued from previous page)

```
        end if;
      end if;
    end process;

    -- Set valid high for full duration of packet
    q(i).valid <= '1' when word_index <= x"9B" else '0';
    -- Start & last are only high for first & last clock cycle of packet
    q(i).start <= '1' when word_index = x"00" else '0';
    q(i).last <= '1' when word_index = x"9B" else '0';

    -- Start of orbit is high in the first clock cycle of the first packet in orbit -␣
↪though in final system this should instead
    -- be high in the first clock cycle of the packet containing the data from BX0 (or␣
↪BXn in time slice n of a TMUX system)
    q(i).start_of_orbit <= '1' when ((word_index = x"00") and (packet_index = "000000000
↪")) else '0';

    -- Data word: Bits 63 to 32 = channel index; bits 31 to 16 = packet index; bits 15␣
↪to 0 = word index.
    q(i).data(63 downto 32) <= std_logic_vector(to_unsigned(i, 32));
    q(i).data(31 downto 16) <= "0000000" & packet_index;
    q(i).data(15 downto 0) <= x"00" & word_index;

  end generate gen;

end rtl;
```

CHAPTER

FOUR

EMP SOFTWARE

The EMP software is a set of libraries and applications that allow users to control and monitor the EMP firmware once it has been loaded onto an FPGA.

The pages listed below explain how to install/compile the software, how to use `empbutler` to control and monitor the EMP firmware, and how to test the firmware and software.

4.1 Installation

The EMP software is packaged as RPMs, stored in YUM repositories, for CentOS 7 and 8 (64-bit, x86 architecture). If you are using one of these platforms, you should follow the instructions in the *YUM repositories* section below, to install the RPMs for the latest tagged release. If you need to use the EMP software on a different OS or achitecture then instead follow the instructions in the *Compiling from source* section.

4.1.1 YUM repositories

The RPMs from our YUM repositories can be installed as follows:

1. If you already have a previous version of the EMP software installed on your computer, first uninstall this old version:

```
sudo yum groupremove emp
sudo yum remove "cactus*emp*"
```

2. Download the YUM repo file, and copy to `/etc/yum.repos.d/`, renaming to `emp.repo`:

```
sudo curl https://serenity.web.cern.ch/serenity/emp-fwk/_downloads/emp.repo -o /etc/
↪yum.repos.d/emp.repo
```

3. Install the software:

```
sudo yum clean all
sudo yum install "cactusboards-emp-*"
```

4. Install Python dependencies

```
sudo python3.6 -m pip install --upgrade click==8.0.4 click_didyoumean==0.3.0␣
↪pytest==7.0.1 python-dateutil==2.8.2 pyyaml==5.1.2 colorama==0.4.5
```

Note: The RPMs install the EMP software under `/opt/cactus` by default. So before using the EMP software, you need to update your environment variables accordingly:

```
export PATH=/opt/cactus/bin/emp:$PATH
export LD_LIBRARY_PATH=/opt/cactus/lib:$LD_LIBRARY_PATH
export PYTEST_ADDOPTS="--rootdir=."
```

The latter variable is only needed if you're running the *test suite*, to set the location for `.pytest_cache`.

4.1.2 Compiling from source

The EMP software is implemented in C++, and built on top of the uHAL library, which in turn uses the BOOST and pugiXML libraries.

Part 1: Prerequisites

The prerequisites can be installed as follows:

1. **BOOST** and **pugiXML**

   ```
   sudo yum -y install boost-devel pugixml-devel python-devel
   ```

2. **uHAL**: See instructions (install the RPMs built with GCC4).

3. **CACTUS build utilities**

   ```
   sudo curl http://serenity.web.cern.ch/serenity/emp-fwk/_downloads/cactus-build-
   ↪utils.repo -o /etc/yum.repos.d/cactus-build-utils.repo
   sudo yum clean all
   sudo yum install cactuscore-build-utils-0.2.9
   ```

4. Python dependencies:

   ```
   sudo python3.6 -m pip install --upgrade click==8.0.4 click_didyoumean==0.3.0␣
   ↪pytest==7.0.1 python-dateutil==2.8.2 pyyaml==5.1.2 colorama==0.4.5
   ```

Part 2: EMP toolbox

After installing the prerequisites, the latest tag can be downloaded and built as follows:

```
git clone --recurse-submodules https://:@gitlab.cern.ch:8443/p2-xware/software/emp-
↪toolbox.git -b v0.8.2
cd emp-toolbox
make
```

Then, in order to set up environment variables (`PATH`, `LD_LIBRARY_PATH` etc) to use the software, run:

```
source env.sh
```

4.2 Standard workflows

The pages below briefly explain how to carry out various common procedures - such as payload firmware tests and link tests - using the EMP software.

4.2.1 Testing payload firmware in a single FPGA

You can "play" user-defined data through your payload, and capture the output, by following the steps listed below.

Note: In these commands, you will need to replace the following arguments with appropriate values:

- `CONNECTIONS_FILE.xml` : Relative/absolute path to your connections file (that specifies the communication protocol and address table files that will be used for each board/FPGA).
- `BOARD_ID` : ID string of your board/FPGA in that connections file
- `RX_CHANNEL_LIST`: Comma-separated list of input channels (e.g. `4-7,12` for links 4, 5, 6, 7 and 12)
- `TX_CHANNEL_LIST`: Comma-separated list of output channels.
  - The list of TX channels does not have to be the same as the list of RX channels - the exact set of channels used depends on the nature of the payload firmware and the overall purpose of the test (e.g. what parts of the algorithm firmware you want to test). However, the payload firmware should set the `strobe` signal high on all output channels that you will try to capture data from.
- `INPUT_DATA.txt` : File containing data that will be loaded into device's input buffers.

---

1. Add the URI and address table for your device to a uHAL connections file (if not done already). For example, for the X1 daughter card on the Serenity, using address table at `path/to/addrtab/top_emp.xml`, the connections file would be:

```
<connections>
  <connection id="x1" uri="ipbuspcie-2.0:///dev/serenity_pcie/x1/h2c,/dev/serenity_
↪pcie/x1/c2h" address_table="file://path/to/addrtab/top_emp.xml" />
</connections>
```

2. Setup your environment

```
export PATH=/opt/cactus/bin/emp:$PATH
export LD_LIBRARY_PATH=/opt/cactus/lib:$LD_LIBRARY_PATH
```

3. Reset the TTC block

```
empbutler -c CONNECTIONS_FILE.xml do BOARD_ID reset internal
```

4. Load data into the input channels' buffers, and configure those buffers to inject that data into the payload

```
empbutler -c CONNECTIONS_FILE.xml do BOARD_ID buffers rx PlayOnce -c RX_CHANNEL_
↪LIST --inject file://path/to/input_data.txt
```

5. Configure buffers for the specified output channels to capture data from the payload, then download the data from the input and output buffers.

```
empbutler -c CONNECTIONS_FILE.xml do BOARD_ID buffers tx Capture -c TX_CHANNEL_LIST
empbutler -c CONNECTIONS_FILE.xml do BOARD_ID capture --rx RX_CHANNEL_LIST --tx TX_
↪CHANNEL_LIST
```

By default, the captured rx/tx data is written to `data/rx_summary.txt` and `data/tx_summary.txt`, but that directory can be changed using the `-o` argument.

4.2.2 Configuring and monitoring link firmware

The sections below outline the `empbutler` commands for manually configuring and testing the link firmware, for both loopback and board-to-board tests. As part of these tests, the datapath firmware is configured to "play" pattern data out of the TX (transmit) MGTs into the RX (receive) MGTs, and then capture the results.

Note: In these commands, you will need to replace the following arguments with appropriate values:

- `CONNECTIONS_FILE.xml` : Relative/absolute path to your connections file
- `DEVICE_ID` : ID string for your FPGA in that connections file
- `RX_CHANNEL_LIST` / `TX_CHANNEL_LIST` / `CHANNEL_LIST`: Comma-separated list of input/output channels (e.g. `4-7,12` for links 4, 5, 6, 7 and 12)

Loopback

1. Add the URI and address table for your device to a uHAL connections file (if not done already)
2. Setup your environment

```
export PATH=/opt/cactus/bin/emp:$PATH
export LD_LIBRARY_PATH=/opt/cactus/lib:$LD_LIBRARY_PATH
```

3. Reset the TTC block

```
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID reset internal
```

4. Load pattern into the output channels' buffers, and configure those buffers to send that data into the TX MGTs at the start of each LHC orbit

```
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID buffers tx PlayOnce -c CHANNEL_LIST -
↪-inject generate:counter-with-index
```

Note: The `PlayOnce` argument here specifies that the data from the input file should only be played into the TX MGTs at the start of each LHC orbit (i.e. once per 3564 cycles of the LHC clock). If this argument is changed to `PlayLoop`, then the data from the input file will be played continually in a loop (i.e. the data played into the TX MGTs on the Nth clock cycle will be `N % 1024`).

5. Reset and configure the link firmware

```
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID mgts configure tx -c CHANNEL_LIST --
↪loopback=nearPMA
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID mgts configure rx -c CHANNEL_LIST
```

By default, these commands will automatically check the status of the link firmware. The status check can be skipped by adding the `--no-check` flag to the end of each `configure` command, or run separately using the `mgts status` command:

```
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID mgts status -c CHANNEL_LIST
```

6. Align data received on each channel, so that the data valid signal rises from 0 to 1 on the same clock edge for all channels.

```
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID mgts align -c CHANNEL_LIST
```

7. Configure input channels' buffers to capture data received from RX MGTs, then download the data from the input and output buffers.

```
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID buffers rx Capture -c CHANNEL_LIST
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID capture --rx CHANNEL_LIST --tx␣
↪CHANNEL_LIST
```

By default, the captured rx/tx data is written to `data/rx_summary.txt` and `data/tx_summary.txt`, but that directory can be changed using the `-o` argument. If you open up the RX capture file - e.g. `less -S data/rx_summary.txt` (the `-S` flag prevents line wrapping) - then you should see the input data after 20 to 30 clock cycles, with the first valid word for each channel occurring on the same clock cycle.

Board-to-board tests

On each board:

1. Add the URI and address table for your device to a uHAL connections file (if not done already)
2. Setup your environment

```
export PATH=/opt/cactus/bin/emp:$PATH
export LD_LIBRARY_PATH=/opt/cactus/lib:$LD_LIBRARY_PATH
```

3. Reset the TTC block, configuring it to use an external source (if using an FC7 to distribute legacy-style TTC signals, change `tcds2` to `legacy` here)

```
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID reset tcds2
```

Then, on the board that will be transmitting data (or on both boards if you want bidirectional data transfer):

1. Load a pattern into the output channels' buffers, and configure those buffers to send that data into the TX MGTs at the start of each LHC orbit

```
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID buffers tx PlayOnce -c TX_CHANNEL_
↪LIST --inject generate:counter-with-index
```

Note: The `PlayOnce` argument here specifies that the data from the input file should only be played into the TX MGTs at the start of each LHC orbit (i.e. once per 3564 cycles of the LHC clock). If this argument is changed to `PlayLoop`, then the data from the input file will be played continually in a loop (i.e. the data played into the TX MGTs on the Nth clock cycle will be `N % 1024`).

2. Reset and configure the TX link firmware

```
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID mgts configure tx -c TX_CHANNEL_LIST
```

Add the `--invert` flag to the end of the command if the TX polarity needs to be inverted; the command can be run multiple times for different sets of channels, if inversion is only required for some of them.

Finally, on the board that will be receiving the data (or on both boards if you want bidirectional data transfer):

1. Reset and configure the RX link firmware

```
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID mgts configure rx -c RX_CHANNEL_LIST
```

   Add the `--invert` flag to the end of the command if the RX polarity needs to be inverted, or the `--enable-dfe` flag if you want to enable DFE mode. The command can be run multiple times for different sets of channels if inversion/DFE is only required for some of them.

   By default, this command will automatically check the status of the link firmware. The status check can be skipped by adding the `--no-check` flag to the end of each `configure` command, or run separately using the `mgts status` command:

```
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID mgts status rx -c CHANNEL_LIST
```

2. Align data received on each channel, so that the data valid signal rises from 0 to 1 on the same clock edge for all channels.

```
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID mgts align -c RX_CHANNEL_LIST
```

3. Configure input channels' buffers to capture data received from RX MGTs, then download the data from the input buffers.

```
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID buffers rx Capture -c RX_CHANNEL_LIST
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID capture --rx RX_CHANNEL_LIST
```

   By default, the captured rx/tx data is written to `data/rx_summary.txt` and `data/tx_summary.txt`, but that directory can be changed using the `-o` argument. If you open up the RX capture file - e.g. `less -S data/rx_summary.txt` (the `-S` flag prevents line wrapping) - then you should see the input data after 20 to 30 clock cycles, with the first valid word for each channel occurring on the same clock cycle.

4.2.3 Bathtub and eye scans

The sections below outline the `empbutler` commands for running and plotting bathtub and eye scans. The first steps are in common with the *Configuring the link firmware* section.

Note: In these commands, you will need to replace the following arguments with appropriate values:

- `CONNECTIONS_FILE.xml` : Relative/absolute path to your connections file
- `LINKS_YAML.yaml` : Relative/absolute path to your links yaml file (described in detail below)
- `DEVICE_ID` : ID string for your FPGA in that connections file
- `CHANNEL_LIST`: Comma-separated list of input/output channels (e.g. `4-7,12` for links 4, 5, 6, 7 and 12)

Python dependencies

Although not necessary to run the eye scans, the optional plotting part has the following additional dependencies:

```
sudo pip install --upgrade pip
sudo pip install matplotlib scipy "iminuit<2.0.0" probfit
```

The latter three packages are required only for the bathtub fitting. For python 2, "iminuit<1.4" is instead required.

Links yaml file

In order to correctly identify the physical links with the EMP channels, the connections and their properties must be specified in a links yaml file.

There are examples for possible single-board and inter-board connections at /opt/cactus/etc/emp/links/singleBoardConnections.yml and /opt/cactus/etc/emp/links/interBoardConnections.yml, but typically users will need to provide their own links yaml file LINKS_YAML.yaml to match their setup.

The links yaml files have two top-level keys, and we'll look at /opt/cactus/etc/emp/links/singleBoardConnections.yml as an example. The first top-level key is fpgas, which holds details of the connected FPGAs:

```
fpgas:
  x0:
    uid: FT-9999A-999
    board:
      - serenity_dc_ku15p_so1_v1
      - serenity_v1.1

  x1:
    uid: FT-8888A-888
    board:
      - serenity_dc_ku15p_so1_v1
      - serenity_v1.1
```

In fpgas there is a key for each connected site, named by the DEVICE_ID (in this case x0 and x1). Each holds the following information:

- uid: serial number of the FPGA
- board: list of two strings identifying the daughter card type (in this case serenity_dc_ku15p_so1_v1) and carrier card type (in this case serenity_v1.1). These strings must **exactly** match the name of a corresponding yaml file in /opt/cactus/etc/emp/links/ (with the .yaml part omitted). These additional yaml files provide the hardware-specific mapping between the EMP channels and the carrier card sockets, and should not need to be edited by the user.

The second top-level key is connections, which contains a list of dictionaries describing the Tx -> Rx connections. Each dictionary in connections represents a group of similar connections and has the following keys:

- from: the DEVICE_ID of the FPGA the connections are from
- to: the DEVICE_ID of the FPGA the connections are to
- type: the type of the connections (can be direct or reverse). This determines the mapping between the 12 channels **within** each tx and rx socket. For direct, rx channel i maps to tx channel i. For reverse, rx socket channel i maps to tx channel (11-i).
- links: map specifying the carrier card sockets the connections are from:to.

The example from `/opt/cactus/etc/emp/links/singleBoardConnections.yml` is as follows:

```
connections:
  - from: x0
    to: x1
    type: reverse
    links:
      Tx0: Rx0
      Tx1: Rx1
      Tx2: Rx2
      Tx3: Rx3
      Tx4: Rx4
      Tx5: Rx5

  - from: x0
    to: x1
    type: direct
    links:
      intraTx: intraRx
```

In this example, there are two dictionaries in the connections list. The first dictionary in the list indicates that there are cables connecting the tx carrier card sockets Tx0-5 at site x0 to the rx carrier card sockets Rx0-5 at site x1. These connections are all of `type: reverse` (in this case, Cu12 cable with X polarity).

The second dictionary in the list describes the direct inter-interposer link from intraTx (site x0) to intraRx (site x1).

Note: When running in internal loopback mode (e.g. `mgts configure tx --loopback nearPMA`), it is known that the tx is connected internally to the rx for each EMP channel, so only the `fpgas` section is required in `LINKS_YAML.yaml`.

Scanning the links

1. Follow the instructions to *configure the link firmware*, up to the `mgts align` command.
2. Scan the links:

```
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID mgts scan -c CHANNEL_LIST --ber=1e-8␣
↪-l LINKS_YAML.yaml --scantype bathtub
```

   - The links yaml file (described in the section above) is specified by `-l` or `--linkmap LINKS_YAML.yaml`.
   - The scan type (bathtub or eye) is specified by `-s` or `--scantype [bathtub|eye]`.
   - The target BER sensitivity of the scan is set using `-b` or `--ber FLOAT`. This specifies the smallest BER that the scan will be able to detect, at a given confidence level (CL). The chosen target BER will be rounded down to the nearest discrete step (corresponding to an integer prescale value).
   - The target BER CL is 90% by default, which means there's a 90% chance an error will be seen if the true BER equals the target BER. A custom CL can be set in % via e.g. `--bercl 99` (this is **not recommended** for standard workflows).
   - Use option `-p, --plot` to plot the scan results. The plots can also be produced by the *Standalone plotting* script, with options for customisation.
   - The default range and granularity of the time (x) and voltage (y) offsets can be overridden using the `--xmax`, `--xstep`, `--ymax`, and `--ystep` options. If these are set, it is not necessary to specify the `--scantype`.

- To scan channels with good Rx status only, use option `-g, --goodrxonly` (this is **not recommended** for standard workflows).

Further details of the options are described in the help: `... mgts scan -h`.

The scan command will create a folder for the results of the scan, named `scans_YYYYMMDD_HHMMSS` using the current date and time. The contents are as follows:

- `data/`: a folder containing the scan data and metadata, saved in one text file per EMP channel. The scan data (measured sample and error counts, i.e. the denominator and numerator for the BER) is stored in csv format. The sample count is stored as *log2(sample_count + 1)*, because this is always an integer power of 2 in the interesting part of the scan (where the error count is not saturated). The metadata is stored in yaml format at the top of the file, but commented out (each line preceded by a #) so it can be easily ignored when reading the csv part.
- If running with option `-l, --linkmap` (recommended):
  - `links.yml`, a copy of the links yaml file
  - `processed_data/`: an updated copy of `data/`, with the links metadata added and the text file names modified to include the link numbers.
- If running with option `-p, --plot`, the plots will be saved here in pdf format.

Note: When running sequences of commands like this it can be useful to use the *Butler scripting interface*, e.g.:

```
empbutler -c CONNECTIONS_FILE.xml script bathtubScanLoopbackTest.script
```

The above example script can be found at `/opt/cactus/etc/emp/examples/bathtubScanLoopbackTest.script`

Standalone plotting

The plotting part can be (re)run independently of empbutler using the following script (which is included in the RPM):

```
plot_emp_scans plot -i indir -o outdir
```

Here, indir is the output folder from the `mgts scan` command (`scans_YYYYMMDD_HHMMSS`), and outdir is the desired location for the output plots (it defaults to indir if unspecified).

The resulting plots are identical to those automatically produced when running `mgts scan` with option `--plot` as described above.

The behaviour of the fit can be customised as follows:

- The upper threshold for the BER data to be included in bathtub fit is set using `-t, --threshold FLOAT`
- The rho density parameter used in the fit is set using `-r, --rho FLOAT`
- The minimum required opening fraction for a bathtub to be considered 'good' is set using `-a, --acceptance FLOAT`

If desired, the links metadata can be (re)processed by specifying option `-m, --processmetadata` or, to set or override the saved links yaml file, `-l` or `--linkmap LINKS_YAML.yaml`.

Comparison plots for two `scans_YYYYMMDD_HHMMSS` directories can be generated as follows:

```
plot_emp_scans compare -i indir -c compdir -o outdir
```

Further options are described in the help: `plot_emp_scans plot -h` and `plot_emp_scans compare -h`.

4.3 EMP butler: Reference guide

`empbutler` is the main command-line tool for controlling and monitoring the EMP framework's TTC, buffer and link firmware. It has a hierarchy of subcommands, each of which report information or carry out actions related to a different aspect of the firmware's functionality. These subcommands are described in the rest of this page, and in the pages therein. These pages are a comprehensive reference guide for people who already have experience in using `empbutler`, so if this is your first time using it, or you just want to know what sequence of commands should be run in a specific scenario, please read the *Standard workflows* page first.

Note: At every level of the `empbutler` subcommand hierarchy, the available subcommands, arguments and options can be listed by adding `--help` - e.g. for the top-level options and subcommands, by running `empbutler --help`

4.3.1 Reading build metadata

```
empbutler -c CONNECTIONS.xml do DEVICE_ID info
```

The `info` subcommand reads registers containing the framework's static build metadata and prints out a summary. This includes version registers, git repository information, and the build-time configuration (i.e. several constants from the `emp_payload_decl` package).

For example, for a Serenity KU115 build with buffer and MGT logic instantiated only in regions 0 to 11 and 14 to 16, the output from this command is:

```
FPGA: KU115   (DNA: 40020001 00f19221 2d008185)

TTC masters available: Internal, Legacy
Payload clock: 240.0MHz

Datapath:
    Region    MGT in     rx buffer   tx buffer    MGT out
    ========================================================
    0-11     16G (GTH)   present     present     16G (GTH)
    12-13    no MGT      none        none        no MGT
    14-16    16G (GTH)   present     present     16G (GTH)
    17       no MGT      none        none        no MGT

Build type: Automated (GitLab CI)
    Project:     32301
    Pipeline:  2335672
    Job:      12258103

Source areas:
                         Branch/Tag     SHA     Uncommitted changes?
    ================================================================
    legacy_ttc           v2.1         7300e64            No
    cms-tcds2-firmware   v0_1_1       650ae73            No
    emp-fwk              v0.5.0       94faeb0            No
    tclink                            fda0bcf            No
    ipbus-firmware       v1.8         c008d1c            No

Version registers:
```

(continues on next page)

(continued from previous page)

```
    Framework: 0.5.0  (design 0x42)
    Payload:   0x12345678
```

4.3.2 Resets and TTC configuration

```
empbutler -c CONNECTIONS.xml do DEVICE_ID reset (internal|external) [--no-check]
```

The `reset` command resets all of the framework's major components, configures the TTC block, and checks its status.

Options & arguments:

- The first argument (which is mandatory) selects the clock source used by the TCC block as the LHC clock:
  - `internal`: Pseudo-LHC clock, generated from the PCIe / AXI chip-to-chip clock.
  - `external`: External clock, connected to the `osc_clk_p/_n` pins.
- The TTC status checks can be skipped by adding the `--no-check` flag.

For example, to use the clock from the `osc_clk_p/_n` pins:

```
empbutler -c CONNECTIONS.xml do DEVICE_ID reset external
```

4.3.3 Buffer configuration

```
empbutler -c CONNECTIONS.xml do DEVICE_ID buffers (rx|tx) MODE [-c CHANNEL_LIST] [--
↪inject SOURCE] [--bx-range BX_RANGE | --frame-range BX_RANGE]
```

The `buffers` command configures the 'datapath' logic, which connects the I/O buffers and pattern generator logic to the link firmware and payload. It has two required arguments:

1. `DIRECTION`: Specifies which set of buffers are being configured - `rx` or `tx`
2. `MODE`: Specifies the mode of operation for the buffers. Valid values:

- `Pattern` - Use data from internal pattern generator logic (an incrementing counter), rather the buffers.
- `PlayOnce` - Play data from buffers at start of each LHC orbit (i.e. when BX counter resets to 0) for length of buffer, typically 1024 words; after `BUFFER_LENGTH` cycles of payload clock, the 'valid' signal will be low until start of next orbit.
- `PlayLoop` - Similar to `PlayOnce`, but `BUFFER_LENGTH` cycles after the start of the orbit, rather than setting the valid signal to 0, the datapath logic will start playing data from start of buffer again, and keep looping round the buffer until the end of the orbit.
- `Capture` - Capture received data; i.e. data from the payload for the TX buffers, or data from the RX MGTs for the RX buffers.

Additional options:

- To configure only a subset of channels, add the `-c` option. By default the buffers for all channels are configured.
- In `PlayOnce` and `PlayLoop` modes, you can specify what data will be loaded into the buffers using the `--inject` option. Valid values are:
  - `file:///path/to/buffers.txt` - load contents of user-created EMP buffer `.txt` file (at `/path/to/buffers.txt`)

  - `generate:counter` - load incrementing counter pattern (i.e. value N for N'th frame), with valid bit set to '1' for the full length of the buffer.
  - `generate:counter-with-index` - load incrementing counter pattern (i.e. value N for N'th frame), with channel index encoded in bits 16 to 19, and valid bit set to '1' for the full length of the buffer.
  - `generate:random` - load random data in 128-frame packets, with an 128-cycle gap between packets.
  - `generate:random:N` - load random data in `N`-frame packets, with a `N`-cycle gap between packets.
  - `generate:random:N:M` - load random data in `N`-frame packets, with a `M`-cycle gap between packets.
- The BX and clock cycle at which playback or capture starts and stops can be specified using either the `--bx-range` or `--frame-range` options. By default capture/playback starts on the first clock cycle of an orbit, and continues for the full length of the I/O buffers.

4.3.4 Data capture

For all I/O buffers that have been configured in capture mode, the `capture` subcommand triggers those buffers to capture the incoming data, and then reads the data and saves it to files.

Options & arguments:

- By default the captures are written to the `data` directory (in the current working directory). This can be changed using the `-o` / `--outputpath` option.
- By default full length of the buffers is read. However, users can read only a fraction of the data by using the `--caplen` (e.g. `--caplen 256` to only capture the first 256 words).
- By default all RX and TX channels will be read. However this can be restricted to a subset of channels using the `--rx` and `--tx` options. You can also avoid capturing on one of these sets of channels via the special `none` value - i.e. `--rx none` or `--tx none`.

For example, to capture data on on all channels:

```
empbutler -c CONNECTIONS.xml do DEVICE_ID capture
```

Or, to only read out the data from RX channels 0 to 39, and TX channels 0 to 15:

```
empbutler -c CONNECTIONS.xml do DEVICE_ID capture --rx 0-39 --tx 0-15
```

Or, to only read out data from RX channels 0 to 39, and not capture on any TX channels:

```
empbutler -c CONNECTIONS.xml do DEVICE_ID capture --rx 0-39 --tx none
```

4.3.5 MGT/link configuration and monitoring

The `mgts` subcommands are listed below. In each case, the list of channels is specified using the `-c` or `--channels` option, i.e:

```
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID mgts SUBCOMMAND [-c CHANNEL_LIST] [OTHER␣
↪OPTIONS]
```

configure tx

Resets and configures TX MGTs plus associated logic, then reads status registers and reports any erroneous values (unless the status check has been disabled)

- The TX polarity can be inverted by adding the `--invert` flag. **Note:** This should only be done if the board traces or cables connect the positive TX pin of each channel to the negative RX pin, and vice versa.
- The MGT can be configured to internally loop the transmitted data to the RX logic using the `--loopback` flag. Valid values are: `nearPCS`, `nearPMA`, `farPCS`, `farPMA`
- The PRBS (pseudorandom binary sequence) pattern generator can be enabled using the `--prbs` flag. Valid values are: `PRBS7`, `PRBS9`, `PRBS15`, `PRBS23`, `PRBS31`
- The status register check can be skipped by adding the `--no-check` flag (though skipping status checks is **not recommended** for standard workflows)

For example, to configure TX channels 0 to 3 and 8 to 11 inclusive (standard polarity, internal loopback disabled):

```
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID mgts configure tx -c 0-3,8-11
```

Or, to configure the same channels with inverted polarity, and skip the status checks:

```
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID mgts configure tx -c 0-3,8-11 --invert --
↪no-check
```

configure rx

Resets and configures RX MGTs plus associated logic, then reads status registers and reports any erroneous values (unless the status check has been disabled)

- The RX polarity can be inverted by adding the `--invert` flag. **Note:** This should only be done if the board traces or cables connect the positive TX pin of each channel to the negative RX pin, and vice versa.
- By default, the MGTs will be configured in LPM mode. DFE mode can be used by adding the `--enable-dfe` flag.
- The PRBS (pseudorandom binary sequence) pattern checker can be enabled using the `--prbs` flag. Valid values are: `PRBS7`, `PRBS9`, `PRBS15`, `PRBS23`, `PRBS31`
- The eyescan circuitry can be disabled by adding the `--disable-eyescan` flag (this is **not recommended** for standard workflows)
- The status register check can be skipped by adding the `--no-check` flag (though skipping status checks is **not recommended** for standard workflows)

For example, to configure TX channels 0 to 3 and 8 to 11 inclusive (standard polarity, LPM):

```
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID mgts configure tx -c 0-3,8-11
```

Or, to configure the same channels in DFE mode:

```
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID mgts configure tx -c 0-3,8-11 --enable-dfe
```

status

Reads the MGT status registers, and reports whether there are any erroneous values (printing the values in that case)

- By default, both RX and TX status registers are checked. They can be restricted to one direction by adding `rx` or `tx`.

For example, to check the RX and TX status registers on channels 0 to 3 and 8 to 11 inclusive:

```
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID mgts status -c 0-3,8-11
```

Or, to check only the status of the RX MGTs:

```
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID mgts status rx -c 0-3,8-11
```

off

Powers off the MGTs and QPLLs

- Although the command is specified at the `--channels` level (like all `mgts` subcommands), the power is controlled at the quad level. Powering off one channel will power off all channels in the same quad.
- By default, both RX and TX MGTs and the QPLL for each quad are powered off.
- If restricted to one direction by adding `rx` or `tx`, the QPLL will only be powered off if both RX and TX MGTs are off.

For example, to power off the RX and TX MGTs and the QPLLs for channels 0 to 3 and 8 to 11 inclusive:

```
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID mgts off -c 0-3,8-11
```

Or, to power off only the RX MGTs (and the QPLLs if the TX MGTs are already off):

```
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID mgts off rx -c 0-3,8-11
```

scan

Run bathtub or eye scans.

See the *Bathtub and eye scans* page for details of the `scan` subcommand.

measure-clocks

Measures frequencies of MGT reference clocks, as well as the RX and TX MGT data clocks

```
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID mgts measure-clocks -c CHANNEL_LIST
```

No additional options/flags, beyond the list of channels.

Note: If any given LHC-sync/-async ref clock that's been declared to the framework isn't used in a design (e.g. the LHC-sync ref clock for a set of regions where only BE MGT link firmware is instantiated), then the framework will not actually connect those pins to anything at all, and the frequency reported by the `measure-clocks` command will be 0.

4.3.6 Top-level interface

`empbutler` has three top-level subcommands:

`list`
: Lists all devices specified by the uHAL connections file.

`do`
: Configures or monitors framework firmware components, via the following subcommands (each covered in a dedicated page, linked from this table):

Subcommand	Scope
info	Reading build metadata
reset	Resets and TTC configuration
buffers	Data capture
capture	Buffer configuration
mgts	MGT/link configuration and monitoring

`script`
: Runs list of `do` subcommands written down in a file (see the *Butler scripting interface* page for details).

There are also a few top-level options (if specified, these should go before the top-level subcommands, e.g. `empbutler -c CONNECTIONS.xml list`):

- `-c` / `--connections`: Relative/absolute path to your connections file, that specifies the communication protocol and address table files that will be used for each board/FPGA.
- `-t` / `--timeout`: Timeout for communicating with the FPGAs (unit: seconds)
- `-v` / `--verbose`: Increases logging verbosity
- `-q` / `--quiet`: Decreases log messsage verbosity
- `-s` / `--timestamp`: Include timestamp in console log messages

4.4 Butler scripting interface

When controlling the EMP firmware, you will typically need to run a sequence of several `empbutler` commands in order to carry out a test (e.g. playing data through your payload firmware, and capturing the outputs). If you need to repeat these procedures several times, running the same sequence of commands manually can become tedious (and unexpected behaviour can easily occur from human error). Instead, you can write down the list of `empbutler do` subcommands in a text file (without the `empbutler do` prefix), and then use the `empbutler script` command to execute those commands.

Specifically, the `empbutler script` command accepts one argument: the path to the file containing the `empbutler do` subcommands. In this file, lines that begin with `#` or only contain whitespace are ignored, and all other lines are assumed to contain subcommands of `empbutler do`.

4.4.1 Example: Link loopback pattern test

If you wanted to configure the EMP links in loopback mode, and play a simple pattern through them, you could run the following commands (replacing `CONNECTIONS_FILE.xml`, `DEVICE_ID` and `CHANNEL_LIST` with appropriate values):

```
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID reset internal
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID buffers tx PlayOnce -c CHANNEL_LIST --
↪inject generate://pattern
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID mgts configure tx -c CHANNEL_LIST --
↪loopback=nearPMA
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID mgts configure rx -c CHANNEL_LIST
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID mgts align -c CHANNEL_LIST
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID buffers rx Capture -c CHANNEL_LIST
empbutler -c CONNECTIONS_FILE.xml do DEVICE_ID capture --rx CHANNEL_LIST --tx CHANNEL_
↪LIST
```

If you or your group need to run this procedure repeatedly, you could instead save the list of subcommands in a text file (e.g. `myLoopbackCommands.txt`):

```
DEVICE_ID reset internal
DEVICE_ID buffers tx PlayOnce -c CHANNEL_LIST --inject generate://pattern
DEVICE_ID mgts configure tx -c CHANNEL_LIST --loopback=nearPMA
DEVICE_ID mgts configure rx -c CHANNEL_LIST
DEVICE_ID mgts align -c CHANNEL_LIST
DEVICE_ID buffers rx Capture -c CHANNEL_LIST
DEVICE_ID capture --rx CHANNEL_LIST --tx CHANNEL_LIST
```

And then execute all of these subcommands from that list by running the following `empbutler script` command:

```
empbutler -c CONNECTIONS_FILE.xml script myLoopbackCommands.txt
```

4.5 Buffer data files

The EMP datapath buffers can be used to capture data from the payload or RX link firmware, and to play user-defined data into the payload or TX link firmware. This page describes the format of the files that the EMP software uses to store this buffer data.

Note: In order to accommodate the new `lword` fields, the format of buffer data files has been updated in version 0.7.0. You can convert data files from the old format to new format using `convert-old-buffer-data-file.exe`, e.g:

```
convert-old-buffer-data-file.exe /path/to/old_format_data_file.txt /path/to/new_format_
↪data_file.txt
```

4.5.1 Format

The data is organised in a simple space-separated table format in plaintext files. Each column the table represents a different input/output channel, and each row represents the data for subsequent clock cycles, starting from 'frame 0' which is at the start of the buffer. The first line of the data file must start with ``ID: `` and should then specify an ID string for the captured/injected data (e.g. the name of the board/pattern). The second line specifies the encoding order of the metadata flags, and must be:

```
Metadata: (strobe,) start of orbit, start of packet, end of packet, valid
```

The column headings – the indices of the stored I/O channels – are then specified. For example:

```
056                     057                     058                     059
```

Finally the data frames themselves follow, with each line prefixed by `Frame NNNN ``. For channels on which the value of strobe is always high, the ``ldata` values are encoded as:

```
PSLV DDDDDDDDDDDDDDDD
```

where

- `P` is the value of the `start_of_packet` field (either `0` or `1`);
- `S` is the value of the `start` field (either `0` or `1`);
- `L` is the value of the `last` field (either `0` or `1`);
- `V` is the value of the `valid` field (either `0` or `1`); and
- `DDDDDDDDDDDDDDDD` is the value of the `data` field, in hex.

If the value of `strobe` is low for at least one clock cycle on a channel, then the value of `strobe` is added just before the other metadata fields. For example, the last word in a CSP packet (where `start_of_packet` and `start` would be low but `strobe, ``last` and `valid` high) with `data` set to 0x0123456789abcdef is stored as:

```
0011 0123456789abcdef
```

4.5.2 Example

The first 20 lines of a file containing incrementing counter data for four channels (indices 0, 1, 70 and 71), with packet and orbit starting on clock cycle 0 and ID `myData`, is shown below:

```
ID: myData
Metadata: (strobe,) start of orbit, start of packet, end of packet, valid

      Link              000                     001                     070              ␣
↪     071
Frame 0000    1101 0000000000000000  1101 0000000000000000  1101 0000000000000000  1101␣
↪0000000000000000
Frame 0001    0001 0000000000000001  0001 0000000000000001  0001 0000000000000001  0001␣
↪0000000000000001
Frame 0002    0001 0000000000000002  0001 0000000000000002  0001 0000000000000002  0001␣
↪0000000000000002
Frame 0003    0001 0000000000000003  0001 0000000000000003  0001 0000000000000003  0001␣
↪0000000000000003
Frame 0004    0001 0000000000000004  0001 0000000000000004  0001 0000000000000004  0001␣
```

(continues on next page)

(continued from previous page)

```
↪0000000000000004
Frame 0005    0001 0000000000000005  0001 0000000000000005  0001 0000000000000005  0001␣
↪0000000000000005
Frame 0006    0001 0000000000000006  0001 0000000000000006  0001 0000000000000006  0001␣
↪0000000000000006
Frame 0007    0001 0000000000000007  0001 0000000000000007  0001 0000000000000007  0001␣
↪0000000000000007
Frame 0008    0001 0000000000000008  0001 0000000000000008  0001 0000000000000008  0001␣
↪0000000000000008
Frame 0009    0001 0000000000000009  0001 0000000000000009  0001 0000000000000009  0001␣
↪0000000000000009
Frame 0010    0001 000000000000000a  0001 000000000000000a  0001 000000000000000a  0001␣
↪000000000000000a
Frame 0011    0001 000000000000000b  0001 000000000000000b  0001 000000000000000b  0001␣
↪000000000000000b
Frame 0012    0001 000000000000000c  0001 000000000000000c  0001 000000000000000c  0001␣
↪000000000000000c
Frame 0013    0001 000000000000000d  0001 000000000000000d  0001 000000000000000d  0001␣
↪000000000000000d
Frame 0014    0001 000000000000000e  0001 000000000000000e  0001 000000000000000e  0001␣
↪000000000000000e
Frame 0015    0001 000000000000000f  0001 000000000000000f  0001 000000000000000f  0001␣
↪000000000000000f
```

4.6 Testing the firmware and software

The EMP test suite (implemented in Python, using `pytest`) validates that the framework firmware and software are functioning as expected, using 'null algo' builds of the EMP firmware. The tests are organised into many test cases, each of which tests a specific workflow or area of functionality with a specific set of parameters. These test cases are grouped into three test modules:

- `test_ttc` - TTC reset & configuration
- `test_datapath` - I/O buffers
- `test_links` - Backend links, including collection of eyescan/bathtub data (verified by configuring the MGTs in internal loopback mode)

4.6.1 Environment

```
export PATH=/opt/cactus/bin/emp:$PATH
export LD_LIBRARY_PATH=/opt/cactus/lib:$LD_LIBRARY_PATH
export PYTEST_ADDOPTS="--rootdir=."
```

The last line is required to set the location for `.pytest_cache`.

4.6.2 Running the test suite

The test suite can be run as follows (after replacing `CONNECTIONS.xml`, `DEVICE_ID` and `CHANNEL_LIST` with the appropriate values for your firmware build and board):

```
python -m emp.tests -v --conn CONNECTIONS.xml --device DEVICE_ID --channels CHANNEL_LIST␣
↪--repeat=1
```

For example, for channels 1, 2, 3, 4 and 12 on a daughter card at site X0 on a Serenity, you would run the following:

```
python -m emp.tests -v --conn connections.xml --device x0 --channels 1-4,12 --repeat=1
```

You can get the test suite to repeat each test case several times by changing the value passed to the `--repeat` flag.

Warning: If external TTC sources (e.g. an FC7 and/or DTH) are not present/configured, then add the `--no-external-legacyttc` and/or `--no-external-tcds2` flags to the end of the test command so the test suite knows that it should expect failures on those interfaces.

Note: If you want to only run a subset of tests, you can specify the name of the test module or test cases using the `-k` flag. For example, to only run the TTC tests:

```
python -m emp.tests -vk test_ttc --conn CONNECTIONS.xml --device DEVICE_ID --channels␣
↪CHANNEL_LIST --repeat=1
```

Or, to run the TTC and I/O buffer tests:

```
python -m emp.tests -vk "test_ttc or test_datapath" --conn CONNECTIONS.xml --device␣
↪DEVICE_ID --channels CHANNEL_LIST --repeat=1
```

CHAPTER

FIVE

REFERENCE BITFILES

The results from automated (GitLab CI) builds of the 'null algo' example designs for the latest tag can be downloaded from the following URLs:

- HiTech Global K800
- MPUltra
- Xilinx VCU118
- Apollo CMv1 VU7P
- Apollo CMv2 VU13P-1
- Serenity KU115 BM1/TM1
- Serenity KU115 SO1
- Serenity KU15P SM1 v1 (360MHz payload clock)
- Serenity KU15P SM1 v1 (240MHz payload clock)
- Serenity KU15P SM1 v1 (GBT build)
- Serenity KU15P SM1 v1 (lpGBT build)
- Serenity KU15P SM1 v2 (360MHz payload clock)
- Serenity KU15P SM1 v2 (240MHz payload clock)
- Serenity KU15P SO1 v1 (360MHz payload clock)
- Serenity KU15P SO1 v1 (240MHz payload clock)
- Serenity KU15P SO1 v1 (lpGBT build)
- Serenity KU15P SO2 v0 (360MHz payload clock)
- Serenity KU15P SO2 v0 (240MHz payload clock)
- Serenity VU7P SO1 v1.0
- Serenity VU7P SO1 v1.1
- Serenity VU9P SO1 v1.1
- Serenity VU9P FO2
- Serenity VU9P SO2
- Serenity VU13P FO2
- Serenity VU13P SO2 (360MHz payload clock)
- Serenity VU13P SO2 (360MHz payload clock, RX only)

- Serenity VU13P SO2 (360MHz payload clock, TX only)
- Serenity VU13P SO2 (240MHz payload clock)
- Serenity VU13P SO2 (lpGBT build)

The tarball downloaded from each of the above links contains the bitfile (`top.bit`) and all of the address tables required to communicate with the firmware (stored in the `addrtab` directory, top-level file `addrtab/top_emp.xml`).

CHAPTER

SIX

RELEASE NOTES

Contents:

6.1 Software

6.1.1 Version 0.8.2

Created on 30th July 2023.

- Backend links: Add support for interpreting new format for link ID word 1 (required for use with APx).

6.1.2 Version 0.8.1

Created on 11th July 2023.

- Python bindings: Block SIGBUS automatically when library loaded.

6.1.3 Version 0.8.0

Created on 6th July 2023.

Note: This release should only be used with v0.8.x firmware, not earlier firmware versions.

- Backend links: Add support for error injection, and for new status registers from v0.8 of firmware.

6.1.4 Version 0.7.6

Created on 13th January 2023.

- Added support for channels-specific TX CSP settings (in C++ library).

6.1.5 Version 0.7.5

Created on 13th January 2023.

- Butler, `mgts measure-clocks` command: Explicitly label unused refclocks.
- Added support for channel-specific values for link settings (in C++ library)

6.1.6 Version 0.7.4

Created on 4th January 2023.

- Added ARM64 YUM repositories.
- Butler: Add command for injecting PRBS errors.
- Butler, MGTs configure command: Add flag for using idle method 2.
- Add checks for QPLL lock and fundamental protocol errors at end of MGT configuration functions.

6.1.7 Version 0.7.3

Created on 17th November 2023.

- Rx MGT configuration: Fix for spurious errors.
- CI: Add hardware test jobs.

6.1.8 Version 0.7.2

Created on 16th October 2022.

- Added `convert-old-buffer-data-file.exe`, which converts buffer data files from the old to new format.

6.1.9 Version 0.7.1

Created on 15th October 2022.

- Fixed small bugs in board data I/O functions.

6.1.10 Version 0.7.0

Created on 13th October 2022.

Note: This release should only be used with v0.7.x firmware, not earlier firmware versions.

- Added support for lpGBT and GBT links
- Added support for SLink readout links
- BE MGT links: Updated to support firmware implementation of new protocol (CSP)
  - As a result, the `empbutler mgts set-board-id` command now has two arguments: the crate ID number and the logical slot number.
- BE MGT links: Added support for setting value of MGT Tx driver settings

- Added ability to suppress data on input channels during configuration of buffers and links, then enable all inputs in synchronised manner.
  - By default inputs will be enabled at the end of the `empbutler buffers rx` and `mgts configure rx` subcommands
  - If you need to configure both input buffers and the input links, then you should add the `--keep-suppressing-inputs` flag to first of the two corresponding commands, so that the inputs are only enabled after both buffers and links have been configured.
- Removed support for Python 2.7
- Datapath: Fixed bug that caused region-local BC0 to be many clock cycles later than BC0 in TTC block
- TTC interfaces: Added more monitoring counters

6.1.11 Version 0.6.8

Created on 24th January 2022.

- TCDS2 interface: Fix status-checking function so that it ignores PRBS registers in non-test mode.

6.1.12 Version 0.6.7

Created on 24th January 2022.

- TCDS2 interface: Support test mode.
- TCDS2 interface: Wait until both RX & TX MGT 'ready' registers are high in configuration procedure.
- Add `ttc-measure-clocks` and `tcds2-test` butler command.
- Butler `ttc-check` command: Add `repeat-every` option.

6.1.13 Version 0.6.6

Created on 15th December 2022.

- Test suite: Fix failing 'MGT power after reset' test.

6.1.14 Version 0.6.5

Created on 14th December 2022.

- Backend link (i.e. Hermes) engine: Enable power of MGTs before configuring them.
- MGT eyescan procedures: Minor improvements; enable eyescan by default when configuring links.

6.1.15 Version 0.6.4

Created on 29th November 2021

All of the improvements in this release are in the buffer configuration:

- Playback modes (`PlayOnce` and `PlayLoop`)
  - If the loaded data is shorter than then length of the buffers, the frame range is now set to the length of the loaded data by default (rather than the length of the buffers).
  - Strobe value is now taken from the data loaded in the buffers (previously, was always set to `1`)
- Capture mode
  - Values of `strobe`, `valid` and the data word are now recorded for every clock cycle
  - To reproduce the previous behaviour of only recording clock cycles in which `strobe == 1`, you can use the new buffer mode `CaptureWhenStrobeHigh`

Warning: When updating to this version of the software, if strobe signals in the payload firmware's output ports are not set to 1 for every clock cycle, then you will need to update to v0.6.3 of the firmware at the same time in order to ensure that the raw contents of captured data files will identical to those captured with previous v0.6.x releases. If you do not update to v0.6.3 of the firmware, and strobe signals from input links are forwarded to output links in your payload firmware, then the strobe signal may now be specified in your output files (as a `0s/1s` prefix) and set to `0` in the initial unused frames from captures.

6.1.16 Version 0.6.3

Created on 14th November 2021

- Moved eyescan/bathtub data-collection routines from Python modules to C++ library.
- Test suite
  - Updated TTC 'source absent' tests to use correct method names (wasn't updated when method names were last updated)
  - Put name of TTC source in names of TTC test cases, rather than TTC source index.

6.1.17 Version 0.6.2

Created on 20th September 2021.

- Butler, `capture` subcommand: Fixed bug in printing of exception message
- Butler, `buffers` subcommand: Fixed bug in handing of `--frame-range` message

6.1.18 Version 0.6.1

Created on 19th August 2021.

- TTC configuration functions: Explicitly issue reset after changing TTC source.
- Implemented functions for SYSMON-based temperature measurement; added this measurement to `empbutler info` subcommand.

6.1.19 Version 0.6.0

Created on 9th August 2021.

- Removed dependence of test suite on ZeroMQ Python package.
- Minor version number incremented to support new v0.6.x FW series

N.B. This release should only be used with v0.6.x firmware, not earlier firmware versions.

6.1.20 Version 0.5.4

Created on 26th June 2021.

- Updated to uHAL v2.8
- Switch to pybind11 for Python bindings

6.1.21 Version 0.5.3

Created on 20th May 2021.

- Added PCIe reconnect script and reference GitLab runner config (for development boards)

6.1.22 Version 0.5.2

Created on 22nd March 2021.

- Fixes bug in TTC configuration and monitoring functions for designs that do not instanitiate legacy TTC interface (e.g. K800 and VCU118 dev boards).

6.1.23 Version 0.5.1

Created on 6th March 2021.

- Fixes bug in parsing of default channel range (`all`) for `buffers` and `capture` commands.

6.1.24 Version 0.5.0

Created on 21st February 2021.

- Updated back-end link classes & functions to support Hermes v2 implementation.
  - Notably, in v2 of the Hermes protocol, there are new TX channel and board ID fields that can be used to automatically determine RX-TX channel mappings (regardless of the 'payload' data sent over the links). These can be printed out with the new `empbutler mgts read-link-info` command.
- Added support for MGT margin analysis - specifically, scripts for saving BER measurements to files, as well as for creating eyescan and bathtub plots from those files.
- Added support for the newly-integrated TCDS2 interface.
  - Included support for existing TTC masters (FPGA-internal and legacy-style TTC over LVDS)
  - The TCDS2 interface is only instantiated in the designs for two Serenity daughter cards so far: KU15P SO2 and KU15P SM1.
  - If you need it to use the TCDS2 interface on other daugther cards, please get in touch.
- Added more build metadata to the framework. Specifically, the following quantities are now embedded in the bitfiles (and shown by the `empbutler info` command):
  - Timestamp (start of synthesis)
  - git repositories: SHA; branch/tag name; flag indicating whether there are any uncommitted changes
  - GitLab CI variables: pipeline & job IDs

N.B. This release should only be used with v0.5.x firmware, **not** earlier firmware versions.

6.1.25 Version 0.4.6

Created on 25th August 2020.

- Adapted to change in datapath playback control registers introduced in v0.3.6 of firmware
- Extended coverage of the link test suite (now covers both `PlayOnce` and `PlayLoop` modes, and a wider range of packet structures)
- Added firmware address tables.

N.B. This release should only be used with v0.3.6 firmware, **not** earlier firmware versions.

6.1.26 Version 0.4.5

Created on 9th July 2020.

- Extended pattern generator URI syntax to specify packet length and gap (format: `generate://pattern:LENGTH:GAP`)

6.1.27 Version 0.4.4

Created on 13th June 2020.

- Added support for CentOS8
- Updated functions reading firmware version for compatibility with FW v0.3.5

6.1.28 Version 0.4.3

Created on 4th April 2020.

- Updated link clock measurement functions to match changes in frequency measurement indices from FW version 0.3.4

6.1.29 Version 0.4.2

Created on 25th March 2020.

- Test suite: Fixed expected latency for 16G GTY links

6.1.30 Version 0.4.1

Created on 24th March 2020.

- Removed 'CDR lock' register from link status checks
- Test suite: Add dedicated tests for TTC/clock reset
- Butler: Add `script` command
- Butler: Add payload clock frequency to output of `info` command

6.1.31 Version 0.4.0

Created on 5th March 2020.

- Updated to uHAL version 2.7

6.1.32 Version 0.3.7

Created on 23rd February 2020.

- Butler: Add `mgts measure-clocks` command, for measuring the frequency of the MGT reference & link clocks.
- Test suite: Skip board-to-board link ests if `--link-mappings` flag not used
- Test suite: Significantly reduced time taken in comparing data captured from buffers with reference values.

6.1.33 Version 0.3.6

Created on 5th January 2020.

- Remove duplicate `etc` directory in path of address tables (i.e. `/opt/cactus/etc/etc/emp` becomes `/opt/cactus/etc/emp`)
- Remove unused datapath configuration modes

6.1.34 Version 0.3.5

Created on 24th November 2019.

- Butler: Update `mgts configure` commands to run status checks (unless `--no-check` flag is used)
- Butler: Improve output of `mgts status` commands
- Butler: Add bash autocompletion for the `DEVICE_ID` argument

6.1.35 Version 0.3.4

Created on 16th November 2019.

- Separated control of RX and TX link firmware into separate procedures. As a result, the `empbutler do DEVICE_ID mgts configure` command has been replaced by two separate commands:
  - `empbutler do DEVICE_ID mgts configure tx` (with options `--invert` and `--loopback`)
  - `empbutler do DEVICE_ID mgts configure rx` (with options `--invert` and `--enable-dfe`)
- Updated source code to use new logging library (and updated default format for log messages printed by `empbutler`)
- `empbutler`: Reorganised internal implementation of some commands
- Test suite: Added board-to-board link tests.

N.B. Like the previous software version, this release should only be used with v0.3.1 or v0.3.2 firmware, **not** earlier firmware versions.

6.1.36 Version 0.3.3

Created on 17th October 2019.

- `empbutler`: Fixed bug that led to some error messages not being displayed correctly
- `empbutler`: Fixed bug that caused failure of `mgts configure` command.

N.B. Like the previous software version, this release should only be used with v0.3.1 or v0.3.2 firmware, **not** earlier firmware versions.

6.1.37 Version 0.3.2

Created on 16th October 2019.

- Updated link functions to disable DFE by default.
- `empbutler`: Fixed bug arising from use of `-v` flag.
- Cleaned up test suite's structure and command-line interface.

N.B. Like the previous software version, this release should only be used with v0.3.1 or v0.3.2 firmware, **not** earlier firmware versions.

6.1.38 Version 0.3.1

Created on 12th October 2019.

- `empbutler`: Adds `mgts align` command (and updates underlying alignment source code)
- `empbutler`: Adds region information to the output of the `info` command.
- Updates link control functions to account for CRC counter reset fixes in the v0.3.1 firmware release

N.B. This version of the software should be used with v0.3.1 firmware, **not** earlier firmware versions.

6.1.39 Version 0.3.0

Created on 28th September 2019.

- Removes dependency on MP7 libraries.
- `empbutler`: Adds control and monitoring commands for 16G and 25G links.

6.1.40 Version 0.2.8

Created on 9th July 2019.

- Re-adds `empbutler` script to RPMs (missing from 0.2.7 RPMs due to makefile bug)

6.1.41 Version 0.2.7

Created on 8th July 2019

- Adds control and monitoring commands for external TTC source

6.1.42 Version 0.2.6

Created on 11th April 2019.

- Adds CI configuration.

6.1.43 Version 0.2.5

Created on 21st February 2019.

- `empbutler`: Fixes mismatch in name of variable (bug only present for some `click` versions).

6.1.44 Version 0.2.4

Created on 2nd December 2018.

- `empbutler`: Fixes bug in conversion of 'frame range' argument for buffers command

6.1.45 Version 0.2.3

Created on 23rd November 2018.

- `empbutler`: Improved error reporting in case of capture failure

6.1.46 Version 0.2.2

Created on 21st November 2018.

- `empbutler` : Number of words captured from buffers can now be modified by command-line flag

6.1.47 Version 0.2.1

Created on 21st November 2018.

- `empbutler`: Updated buffer command to support frame-based ranges
- `empbutler`: Added flag to skip checks at end of `reset` command

6.1.48 Version 0.2.0

Created on 16th October 2018.

- Core library updated to support buffer firmware with 64-bit data words

6.1.49 Version 0.1.0

First tag; created 23rd May 2018

6.2 Firmware

6.2.1 Version 0.8.2

Created on 13th Feb 2024.

- lpGBT: Add simple combinatorial mux for driving IC frames over the EC channel.
- Disable PCIe AXI clock workaround from Vivado 2023.2 (since XDMA bug was resolved in the corresponding XDMA IP core version).
- Add assertion for availability of ref clock - so that there's a clear error message if one tries to instantiate link firmware in regions without the requisite ref clock type.
- CI: Update to IPBB 2023a.
- CI: Update gendecoders command to use check committed address decoder files in BE-only jobs.

6.2.2 Version 0.8.1

Created on 30th July 2023.

- Backend links: Updated TX core to prevent filler word being inserted between orbit tag and first valid word of packet.
- Backend links: Updated RX core to pass through all 32 bits from link ID word 1, to support new format for that word (required for use with APx).

6.2.3 Version 0.8.0

Created on 4th July 2023.

- Backend links: Updated encoding and improved robustness (as part of tests with APx) - notably:
  - Reversed bit ordering in scrambler
  - Added error injection capabilities
  - Aded new status registers and widened some existing counters (incl. in alignment monitoring block)
  - Re-wrote index correction mechanism entities to improve robustness
- Simulation testbench: Fixed phase relation of 40MHz, payload clock and payload reset so that now matches phase relation in synthesised firmware.
- Frontend links: Added support for lpGBT v1

6.2.4 Version 0.7.5

Created on 9th February 2023.

- Added build for Apollo Rev2 (VU13P-1)
- Fixed bugs in UDP-based simulation.
- SLink: Updated to newer version of IP core & protocol
- SLink: Fixed reset latency bug in CRC block.

6.2.5 Version 0.7.4

Created on 13th January 2023.

- Testbench (for payload testing): Fixed buffer file formatting bug

6.2.6 Version 0.7.3

Created on 14th October 2022.

- Backend links: Updated constraints to work in designs with TX- or RX-only RX CSP cores
- CI: Updated to IPBB 2022f

6.2.7 Version 0.7.2

Created on 23rd October 2022.

- Added `emp_slink_types` to package `.dep` files for various boards

6.2.8 Version 0.7.1

- Fixed Vivado 2022.1 errors in TCDS2 'false path' constraints

6.2.9 Version 0.7.0

Created on 13th October 2022.

- Added support for lpGBT and GBT links
- Added support for SLink readout links
- BE MGT links: Updated from Hermes to CSP protocol, including support for back-to-back packets
  - As part of this, added `last` and `start_of_orbit` fields to `lword` type. On input links, `start` and `last` will now be high on the first and last clock cycle of each packet, and `start_of_orbit` will be high on the first clock cycle of the packet that contains data from the BX0 collision. `valid` is still high inside packets, and low for any filler words in spaces between packets. These fields have the same meaning on output links. A more detailed description of these can be found in the … page.
- Added ability to suppress data on input channels during configuration of buffers and links, then enable all inputs in synchronised manner.
  - When inputs are suppressed, `start_of_orbit`, `start` and `last` will all be low, `strobe` will be high, and the value of `data` is undefined.

  - When inputs are enabled at the end of buffer/MGT configuration, the software writes to the 'input enable' register, and then for each input channel the suppression stops in the next orbit when `start_of_orbit` is high.
- BE MGT links: Added support for setting value of MGT Tx driver settings.
- Datapath: Fixed bug that caused region-local BC0 to be many clock cycles later than BC0 in TTC block.
- TTC interfaces: Added more monitoring counters
- TTC interfaces: Moved source select and 'has interface' registers off 40MHz clock, so that they still work if external TTC clock stops or TTC configuration fails.

Migration guide: 0.6.x to 0.7.0

- Add `use work.emp_slink_types.all;` to the top of your `emp_project_decl` VHDL file, and add the following `SLINK_CONF` constant to the `emp_project_decl` package:

```
constant SLINK_CONF : slink_conf_array_t := (
  others      => kNoSlink
  );
```

  You can keep this `others => kNoSlink` value unless you need to use SLink links, or are otherwise explictly testing readout firmware and hence need the SLink firmware to be instantiated.

- Add `use work.emp_slink_types.all;` to the top of your `emp_payload` VHDL file, and add the following three new `emp_payload` ports:

```
clk40        : in std_logic;
slink_q      : out slink_input_data_quad_array(SLINK_MAX_QUADS-1 downto 0);
backpressure : in std_logic_vector(SLINK_MAX_QUADS-1 downto 0)
```

- Also, note that the format of the buffer data files has changed in order to support new `lword` fields. The new format is summarised in the *Buffer data files* page.

6.2.10 Version 0.7.0, alpha pre-release

Note: This tag should only be used if you need to instantiate lpGBT links in your design.

Created on 27th March 2022.

- First implementation of lpGBT and GBT links included in framework.

6.2.11 Version 0.6.8

Created on 26th March 2022.

- Resolved bug in frequency of XDMA clocks in Vivado 2021.2 (likely affects all Vivado versions since 2020.2.2). This Vivado bug led to paths for those clocks being timed in with half of the frequency that actually exists in hardware - as a result, errors appear in the `reset` butler command for all TTC sources.
- BE links: Resolved error with setting `USER_CLOCK_ROOT` in Vivado 2019.2

6.2.12 Version 0.6.7

Created on 14th March 2022.

- Clocking: Declare the 40MHz 'internal' clocks (input to MMCM) from different sources as logically exclusive, so that Vivado doesn't time those in.
- TTC: Add signal property to ensure that `ttc/tcds2_interface_stat` is not renamed in optimisation.
- Refactored CI (merged package & publish steps into main build job)
- Serenity KU15P SO2: Added example design with 240MHz payload clock.

6.2.13 Version 0.6.6

Created on 2nd March 2022.

- Serenity CPLL-based PCIe designs: Add workaround for 'double requested freq' Vivado bug encountered from 2020.2.2.
- BE links: Set `USER_CLOCK_ROOT` for MGT clocks (for some builds, Vivado used an inappropriate clock region)
- BE links: Permanently power down RX/TX MGTs in regions where only one direction requested (to improve Vivado power estimates).
- TTC: Increase `TTC_DEL` constant, to support 240MHz payload clock on VU13P
- Serenity VU7P: Add BE MGT links to more regions in example design.

6.2.14 Version 0.6.5

Created on 6th February 2022.

- Apollo: Updated build to use `v1.2.2` tag of the Apollo `CM_FPGA_FW` repo (for consistency with current Zynq configuration on boards in TIF).

6.2.15 Version 0.6.4

Note: Firmware built using this release (and other v0.6.x firmware releases) must be controlled using either v0.6.5 of the software or higher.

Created on 23rd January 2022.

- TCDS2 interface: Combine BC0 signals from the two channels (as a simple protection against bitflips in TTC path from DTH to Serenity).

- TCDS2 interface: Invert RX p & n signals within `emp_ttc`, to compensate for inversion in `tcds2_interface_with_mgt`
- Serenity VU9P SO2/FO2, VU13P SO2/FO2 and KU15P SO2 daughter cards: Switch from 5G to 10G TCDS interface.
- Backend links (Hermes) engine: Connect MGT power enable ports to corresponding control bus registers (i.e. from this release, MGTs are initially powered off on programming the FPGA, and the MGT power for individual quads is enabled as needed, by the software).
- CI: Change Serenity KU15P SO2 from minimal to 'full' build.

6.2.16 Version 0.6.3

Created on 30th November 2021.

- Serenity Z1.2 daughter cards: Added 5G TCDS2 interface (alongside general support for 5G TCDS interfaces)
- Updated channel buffer to set strobe signed high inbetween playback windwos

6.2.17 Version 0.6.2

Created on 5th November 2021.

- Added design for Apollo - CM v1, VU7P
- Re-wrote several Hermes link FW components, to ensure that correct techniques used for clock-domain crossing, and reduce resource usage (number of CRC blocks per link reduced from two to one)
  - This resolves CRC errors observed with some links in some builds (depending on exact placement of compoonents)
- Vivado 2021.1: Fixed synthesis error
- Testbench: Added logic to increment TTC counter
- Added more pipelining to the signals driving the payload reset ports (to resolve timing failure in some designs)
- Updated DRP address table to be compatible with new C++ implementation of eyescan software

6.2.18 Version 0.6.1

Created on 20th August 2021.

- Added location constraint to use SYSMON from master SLR.

6.2.19 Version 0.6.0

Note: Firmware built using this release (and other v0.6.x firmware releases) must be controlled using a 0.6.n version of the software.

Created on 9th August 2021.

- Updated the aux clock configuration to support frequencies that are not a multiple of 40MHz.

  - The aux clock frequencies are set by specifying divisors of the MMCM VCO frequency, via the `CLOCK_AUX_DIV` constant of the `emp_project_decl` . I.e. the frequency of each aux clock is `CLOCK_COMMON_RATIO * 40MHz / CLOCK_AUX_DIV(n)`
  - Reminder: The frequency of the aux clocks was previously specified through the `CLOCK_AUX_RATIO` constant, with the frequency of each aux clock equal to `CLOCK_AUX_RATIO(n) * 40MHz`
- VCU118: Updated assignment of EMP datapath regions to quads in order to match the region-quad mapping of the Serenity VU9P builds
  - I.e. EMP region 0 is now constrained to clock region X5Y0 (previously was X5Y1), EMP region 14 is in clock region X5Y14 (previously in X0Y14), EMP region 15 is in clock region X0Y14 (previously in X0Y13), and EMP region 29 is in clock region X0Y0 (previously did not exist as there were only 28 regions).
- Serenity VU9P DC designs: Fixed PCIe channel assignment.
- CI: Updated to `dev/2021i` tag of IPBB

Migration guide: 0.5.x to 0.6.0

- Replace the `CLOCK_AUX_RATIO` constant in your `emp_project_decl` package with `CLOCK_AUX_DIV` changing the type to `clock_divisor_array_t`, and the values to `CLOCK_COMMON_RATIO / CLOCK_AUX_RATIO(n)`. For example, in a design with the following aux clock configuration (80, 160 and 360MHz):

```
constant CLOCK_COMMON_RATIO : integer             := 36;
constant CLOCK_RATIO        : integer             := 9;
constant CLOCK_AUX_RATIO    : clock_ratio_array_t := (2, 4, 9);
```

  In migrating from 0.5.x to 0.6.0 these settings would need to be changed to:

```
constant CLOCK_COMMON_RATIO : integer               := 36;
constant CLOCK_RATIO        : integer               := 9;
constant CLOCK_AUX_DIV      : clock_divisor_array_t := (18, 9, 4);
```

- VCU118 builds: Add one to each region index in the `REGION_CONF` constants from your `emp_project_decl` package

---

6.2.20 Version 0.5.8

Created on 7th July 2021.

- Build metadata: Fixed 1-hex-char shift in git repo SHA with newer git versions.
- Serenity VU7P & VU9P SO1 v1.1 DC: Update reference clock assignments to avoid using clock ping that is not physically connected.

6.2.21 Version 0.5.7

Created on 19th May 2021.

- Serenity-Z VU9P SO1.1 daughter card: Fixed location constraint for PCIe channel
- Added design for Serenity-Z VU7P SO1.1 daughter card
- Re-organised board-specific dependency files, and removed dependency of `emp_device_decl` on TCDS2 repositories, in order to simplify user-created testbench depfiles.
- Updated source code to resolve Vivado warnings.

6.2.22 Version 0.5.6

Created on 2nd May 2021.

- Update dependency files for compatibility with `dev/2021f` tag of IPBB
  - Note: In order to use that tag of IPBB, you will also need to use a new tag - `v1.9` - of the IPbus firmware repository (the build instructions have been updated to reference this new tag).

6.2.23 Version 0.5.5

Created on 24th April 2021.

- Added designs for Serenity VU9P and VU13P daughter cards, specifically:
  - VU9P SO1 v1.1 (uses the B2104 package)
  - VU9P SO2 (A2577 package; symmetric I/O)
  - VU9P FO2 (A2577 package; fan in & fan out)
  - VU13P SO2 (A2577 package; symmetric I/O)
  - VU13P FO2 (A2577 package; fan in & fan out)
- Serenity DC designs: Removed an MMCM from the clock paths for the legacy TTC interface.

6.2.24 Version 0.5.4

Created on 8th April 2021.

- Serenity KU15P SM1 daughter cards: Relaxed constraint on BUFG in TCDS2 interface (from `LOC` to clock region), in order to give Vivado more flexibility with clock routing (and thereby resolve placement errors in some designs). Also constrained the TCDS2 interface to relevant quad pblock.

6.2.25 Version 0.5.3

Created on 30th March 2021.

- Updated synthesis settings to prevent Vivado from flattening hierarchy
  - Up to and including version 0.5.2 of IPBB, it automatically applied this setting to prevert hierarchy flattening; we found that hierarchy flattening adversely affects timing closure in builds, and so updated the framework's TCL scripts so that hierarchy flattening is still prevented the latest releases of IPBB.

6.2.26 Version 0.5.2

Created on 17th March

- IPBB: Updated to tag `dev/2021b`
- BE MGT (Hermes) links: Updated timing constraints so that CDCs that were erroneously being timed in are not correctly declared as false paths.
- VCU118 designs: Move PCIe block in order to ease timing closure. As a result, input/output buffers should no longer be instantiated in datapath regions 4 and 5 (i.e. channels 16 to 23).
- Datapath: Added register for TTC commands, to ease timing closure.
- CI: Reorganised configuration

6.2.27 Version 0.5.1

Created on 25th February 2021

- Fixed edge-case bug in script that collects build metadata.
- Reformatted source code (to have consistent indentation etc).

6.2.28 Version 0.5.0

Created on 21st February 2021.

Note: Firmware built using this release must be controlled using version 0.5.0 of the software.

Backward-incompatible change

- We have removed the CRC field from the project declaration, since only one CRC type is supported for the back-end links
  - When updating to this tag, you will need to remove all instances of `u_crc32` from the `REGION_CONF` setting in your `emp_project_decl` packages - e.g. change:

```
constant REGION_CONF : region_conf_array_t := (
  0  => (gth16, u_crc16, buf, no_fmt, buf, u_crc16, gth16),
  1  => (gth16, u_crc16, buf, no_fmt, buf, u_crc16, gth16)
  );
```

    to:

```
constant REGION_CONF : region_conf_array_t := (
  0  => (gth16, buf, no_fmt, buf, gth16),
  1  => (gth16, buf, no_fmt, buf, gth16)
  );
```

- We renamed top-level address table (now `top_emp.xml`, was previously `top_emp_slim.xml`)

Main improvements

- Updated back-end link firmware to implement version 2 of the Hermes protocol, with 64b/67b encoding. Notably TX channel & board IDs are now sent automatically sent on the links (in control words).
- Added support for MGT margin analysis (i.e. measuring BERs with offsets, e.g. for eyescan and bathtub plots).
- Integrated TCDS2 interface into the framework
  - The framework also still supports the existing TTC masters (FPGA-internal and legacy-style TTC over LVDS)
  - The TCDS2 interface is only instantiated in the designs for two Serenity daughter cards so far: KU15P SO2 and KU15P SM1.
  - If you need it to use the TCDS2 interface on other daugther cards, please get in touch.
- Added more build metadata to the framework. Specifically, the following quantities are now embedded in the bitfiles (and shown by the `empbutler info` command):
  - Timestamp (start of synthesis)
  - git repositories: SHA; branch/tag name; flag indicating whether there are any uncommitted changes
  - GitLab CI variables: pipeline & job IDs

6.2.29 Version 0.3.6

Created on 25th August 2020.

Note: Firmware built using this release must be controlled using version 0.4.6 of the software.

- Updated the datapath logic to specify a 'quiet' period at the end of the orbit, such that buffer playback stops several BX before the end of the orbit
- Updated the alignment block to only use the first alignment signal rising edge from each orbit

6.2.30 Version 0.3.5

Created on 2nd July 2020.

Note: Firmware built using this release must be controlled using version 0.4.4 (or higher) of the software.

- Updated BE MGT link firmware to improve timing closure, and fix high-severity 'no clock' warnings in the CRC error counter process
- Resolved timing warnings for top-level ports
- Updated to latest tag of IPbus firmware repository: version 1.8

- Board-specific changes:
  - Added design for Serenity VU7P daughter card
  - Serenity KU15P DCs: Increased width of quad pblocks on left-hand side to match resources in pblocks on right-hand side
  - Serenity SM1 v1 daughter card: Added 240MHz example design
  - Updated PCIe constraints in several designs, to use consistent strategy on all boards

6.2.31 Version 0.3.4

Created on 3rd April 2020.

- Updated framework to support two categories of link reference clocks - sync and async - in preparation for lpGBT links.
- New board design: Serenity KU15P SO1 daughter card
- Serenity daughter cards: Updated ref clocks used for BE MGT links in several datapath regions (part of preparation for lpGBT links)
- All board designs: Added diagrams of the mapping of EMP datapath region indices to clock regions and Xilinx quad numbers.

6.2.32 Version 0.3.3

Created on 23rd February 2020.

- Update to v1.7 tag of IPbus firmware (which fixes a Vivado 2019.2 build error in PCIe-based Ultrascale+ designs)
- MGT links: Connect link clocks to frequency measurement block, and fix logic errors in signal assignment for ref clock frequency measurements.
- Testbench example: Fix bug in `emp_project_decl` package

6.2.33 Version 0.3.2

Created on 1st December 2019.

- Updated link logic to use full depth of rx alignment FIFO (512 rather than 128 words, no impact on resource usage).
- Serenity KU115 daughter card: Fixed temperature grade listed in configuration files.
- Serenity KU115 daughter card: Remove erroneous instantiation of MGT logic in regions without MGTs.

6.2.34 Version 0.3.1

Created on 11th October 2019.

- Fixed bugs in link CRC counter reset logic
- Fixed bug in link alignment logic (alignment marker signal not driven) and alignment address table.
- Added pin constraints and example projects for v2 design of Serenity KU15P daughter card.
  - **N.B.** `boards/serenity/dc_ku15p` directory was reorganised, so if building for the Serenity KU15P SM1 daughter card, you will need to change `include -c boards/serenity/dc_ku15p` in your dependency files to:
    - `include -c boards/serenity/dc_ku15p dc_ku15p_sm1_v1.dep` for the v1 daughter card design
    - `include -c boards/serenity/dc_ku15p dc_ku15p_sm1_v2.dep` for the v2 daughter card design

6.2.35 Version 0.3.0

Created on 26th September 2019.

- Added 16Gbps (GTH & GTY) and 25Gbps (GTY) link firmware
- Fixed datapath logic errors encountered with payload clock > 320MHz
- Now using IPbus-PCIe interface from main *ipbus-firmware* repo (removed local implementation of that module)

6.2.36 Version 0.2.5

Created on 16th April 2019.

- Renamed top-level address table (now `top_emp_slim.xml`)

6.2.37 Version 0.2.4

Created on 11th March 2019.

- Payload-only simulation: Added 'playback loop' mode (input file contents played into algo continually, in a loop)

6.2.38 Version 0.2.3

Created on 22nd February 2019.

- New board designs: Serenity KU15P and KU115 daughter cards
- Reorganised and simplified the area constraints

6.2.39 Version 0.2.2

Created on 16th November 2018.

- Resolved clash in device IDs for two designs (MPUltra and sim)
- Updated Vivado and modelsim versions used for testing: 2018.2 and 10.6c
- Payload-only simulation: Pattern reader updated to support 'others' column (allows one to specify default data values for I/O channels)

6.2.40 Version 0.2.1

Created on 30th October 2018.

- Fixed bug in boundaries of payload area constraints on MPUltra
- Payload-only simulation: Added several new top-level generics

6.2.41 Version 0.2.0

Created on 16th October 2018.

Differences w.r.t. v0.1.2:

- Data width for payload I/O channels increased from 32 bits to 64 bits (so that at 240MHz each channel now corresponds to a 16Gbps link)
- Implemented example designs for Xilinx VCU118 dev board (VU9P)
- Updated testbench implementation, including support for 'sparsely populated' files
- Name of payload entity changed from `mp7_payload` to `emp_payload`
- Data types for the payload I/O channels are specified in `emp_data_types` package, which replaces the `mp7_data_types` package

---

6.2.42 Version 0.1.2

Created on 18th June 2018.

Differences w.r.t. v0.1.1

- Introduced injection/capture testbench support.

6.2.43 Version 0.1.1

Created on 29th May 2018.

- Renamed packages:
  - `emp_fwk_decl` becomes `emp_framework_decl`
  - `emp_design_types` becomes `emp_device_types`
  - `emp_design_decl` becomes `emp_device_decl`
  - `emp_proj_decl` becomes `emp_project_decl`

- Fixed bug in top-level sim files (were still referencing DAQ-related firmware that was recently removed from main framework).

6.2.44 Version 0.1.0

First tag; created on 23rd May 2018.

Notably:

- The *Walkthroughs for First Users* page explains how to integrate custom "*physics algorithm*" payload firmware into the framework.
- The *Testing payload firmware in a single FPGA* page explains how to use the software to:
  1. load data onto the input buffers;
  2. "play" it through the payload firmware; then
  3. capture and read back the contents of the output buffers.