A new Linux kernel subsystem for JESD204 multichannel RF Transceiver Systems

Michael Hennerich

Systems Development Group

AHEAD OF WHAT'S POSSIBLE™

Abstract

- Many applications need multiple channels of phase and frequency synchronization and coherency. Applications like Direction of Arrival (DOA) accuracy is directly related to the number of channels and the synchronization between these channels.
  However, synchronizing multiple high-speed RF transceiver systems is a challenging task from both hardware and software point of view. In most cases these systems need to scale both vertically and horizontally.
  This presentation will introduce a new yet to be mainlined JESD204 Linux kernel subsystem, which greatly simplifies configuration, bring-up and synchronization of multi topology converter systems and clocking trees. We will cover the basic architecture and concepts, the stack-up and components, how they typically interact with each other, the type of (inter)dependencies that exist and finally how easy it is to utilize, compose and scale such a system.

Introduction

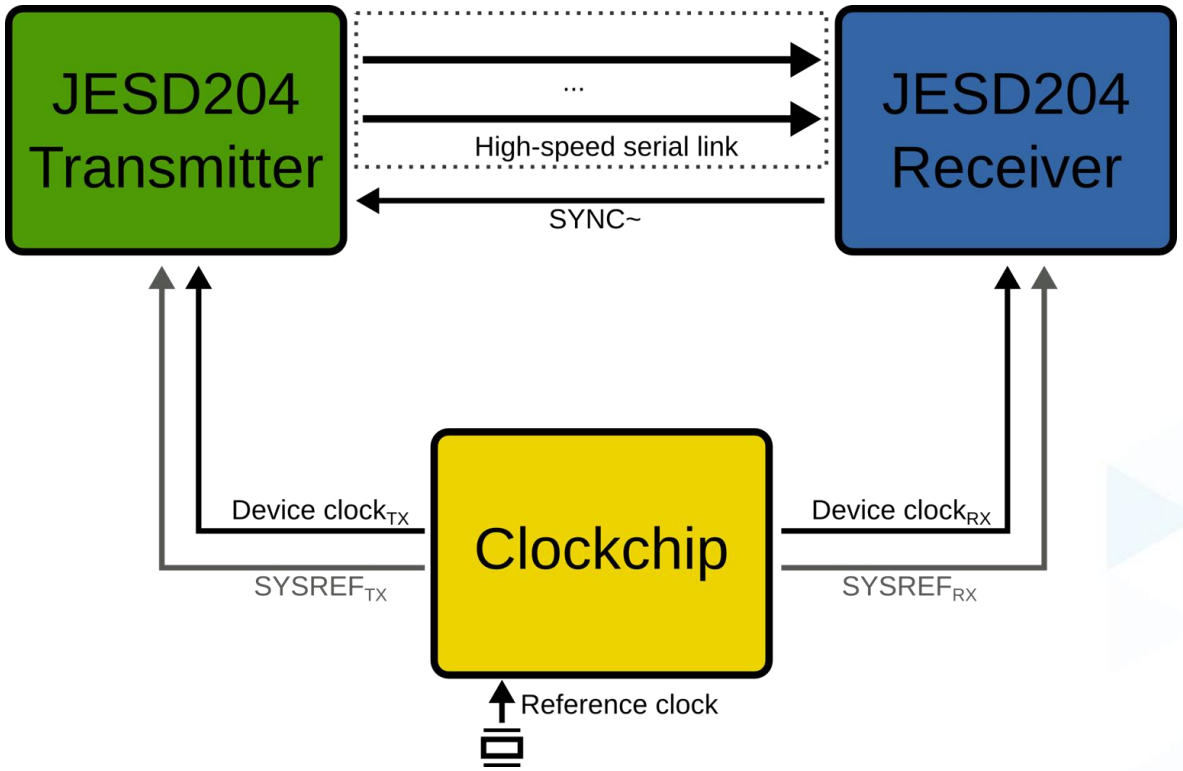
JESD204 Standard

- **Designed as high-speed serial data link between converter (ADC, DAC) and logic device**
  - Up to 32 lanes per link
  - Up to 32 Gbps (raw) per lane
- Describes data mapping and framing
- Multi-chip synchronization
- Deterministic latency

- 2006: JESD204
  - 1 lane, 3.125Gbps
- 2008: JESD204A
  - Multi-lane, 3.125 Gbps
- 2012: JESD204B
  - Multi-lane, 12.5 Gbps
  - Deterministic latency
    - Subclass 0, 1, 2
  - More flexible clocking scheme
- 2017: JESD204C
  - Multi-lane, 32 Gbps
  - 64b/66b and 64b/80b encoding
  - Forward Error Correction

JESD204 Architecture

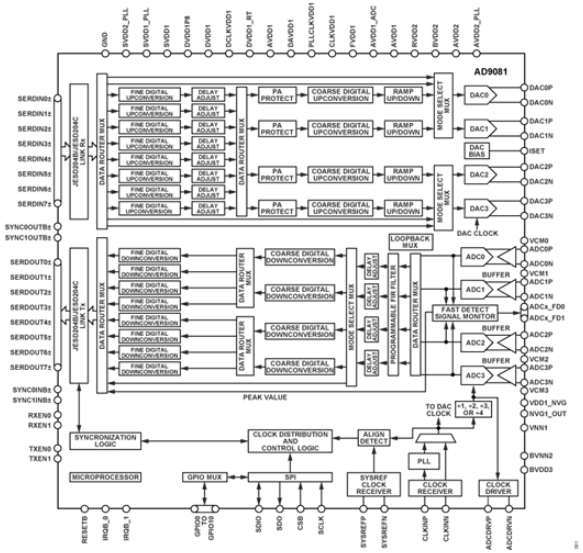
Overview



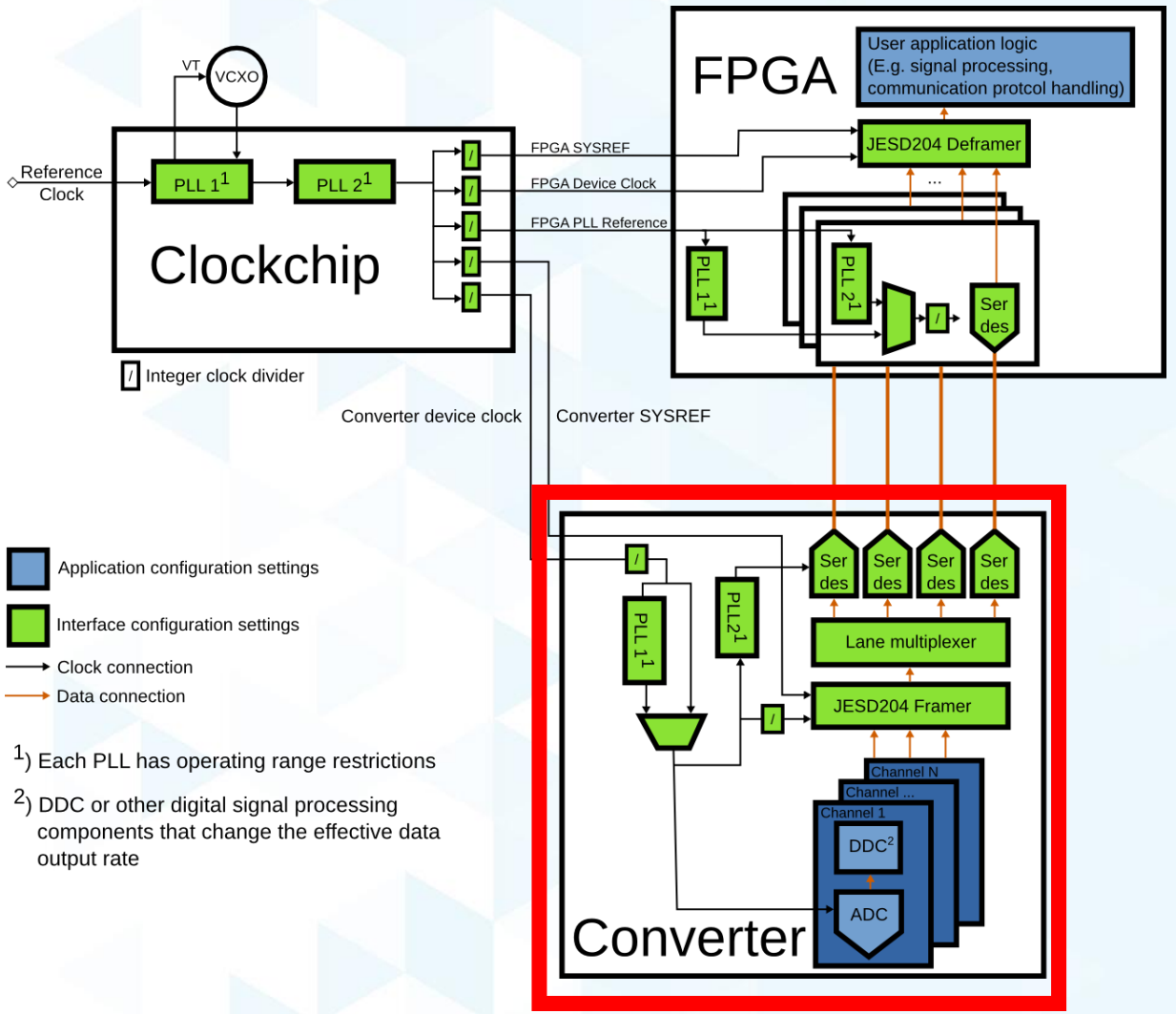
Layers

Application	Application specific processing
Transport	Sample framing Lane mapping
Link	Scrambling Character replacement 8b/10b encoding
Physical	High-speed SerDes Clock recovery Signal shaping

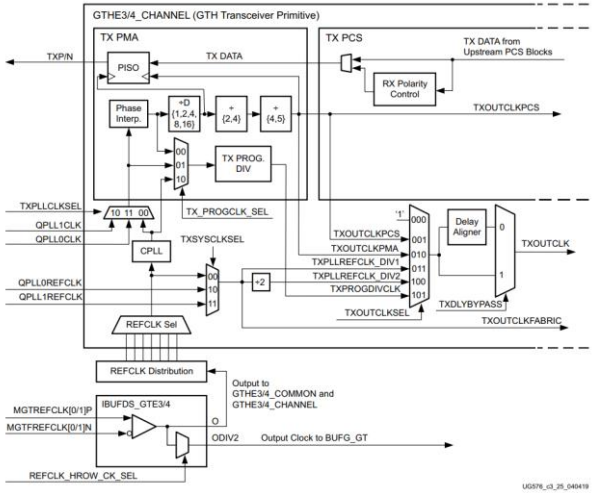
Converter Device



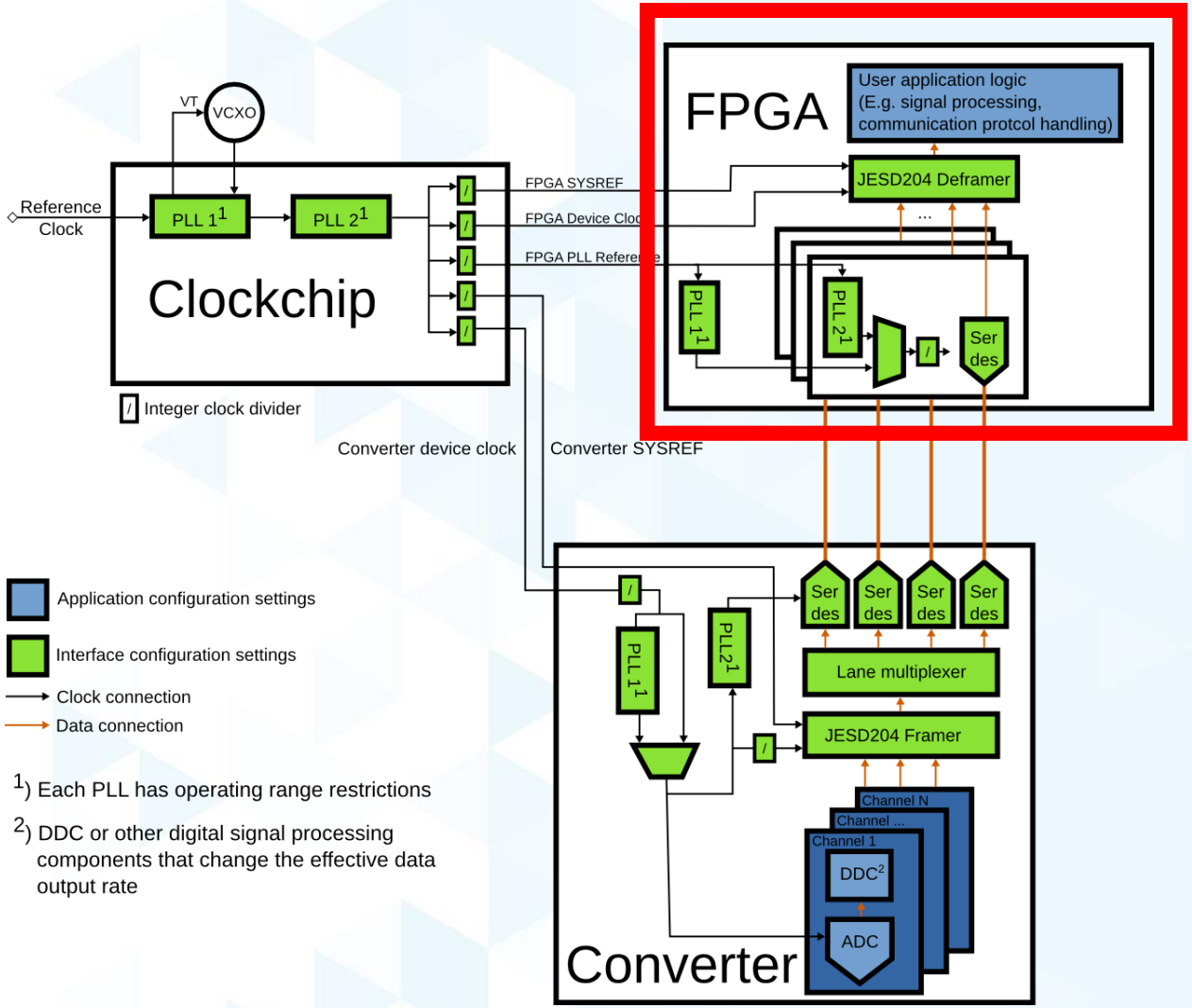
- Does either A2D or D2A conversion, sometimes both.
- Contains one or more converters
  - All synchronous
- Modern converter devices often include digital processing (DDC, DUC, NCOs, etc.)



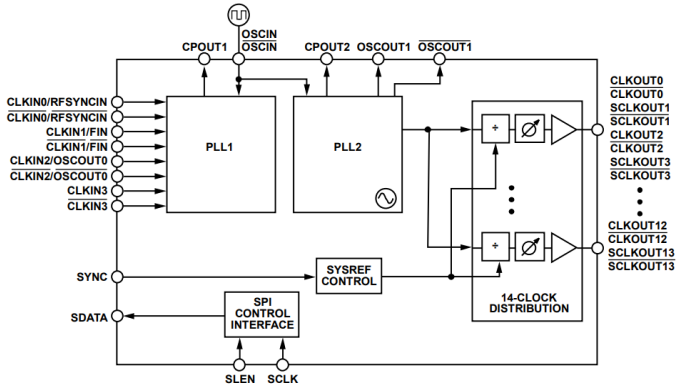
Logic Device



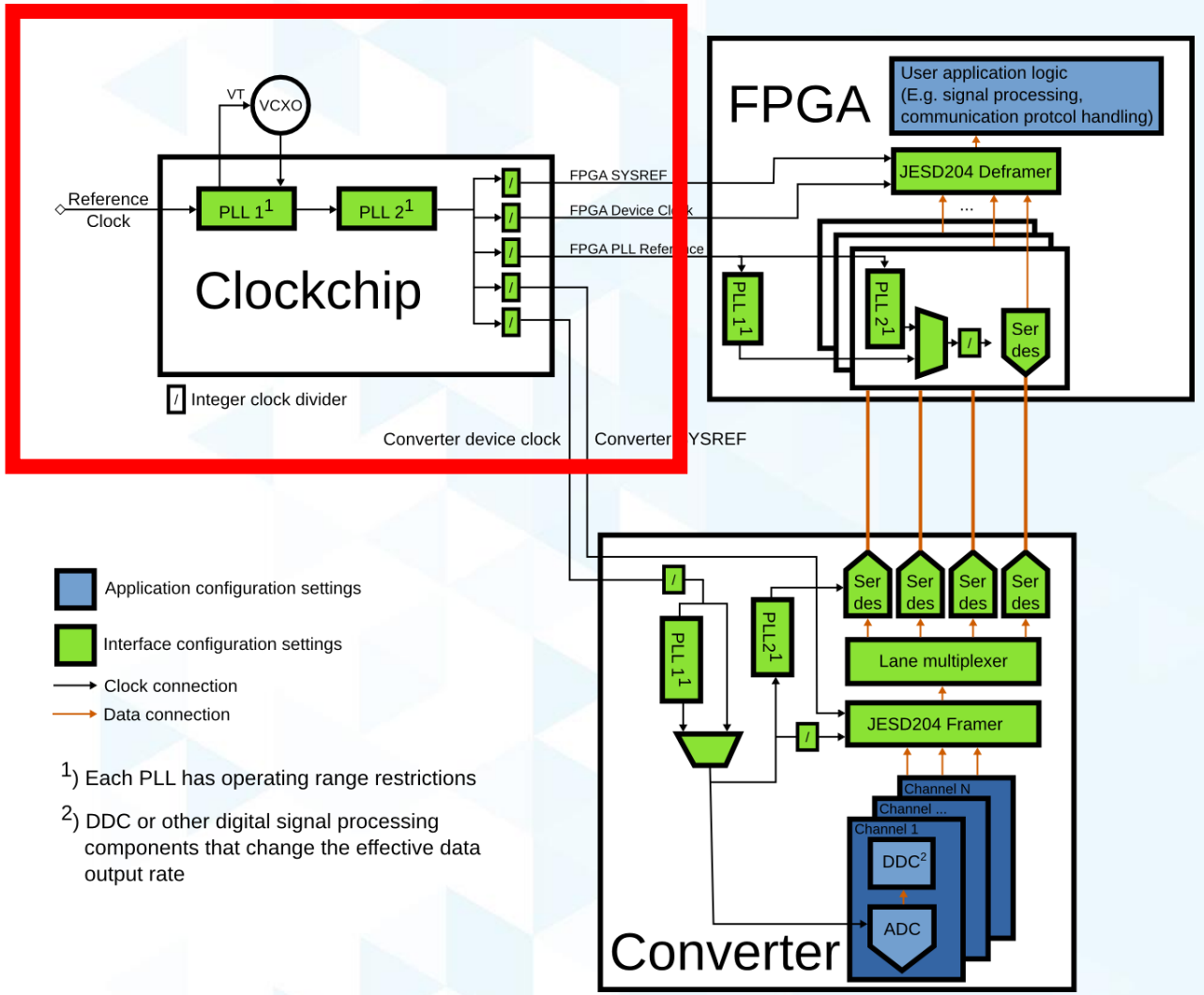
- Implements digital signal processing
- Often implemented in a FPGA
- One logic device can interface multiple synchronous converter devices
  - Multi-point link



Clock Generator

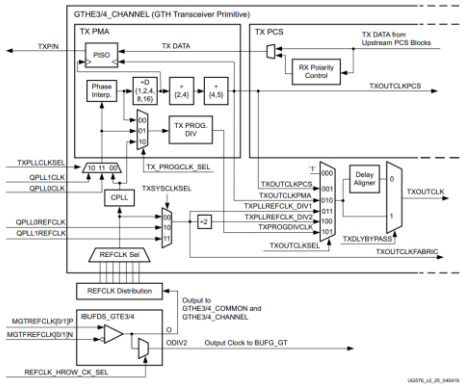


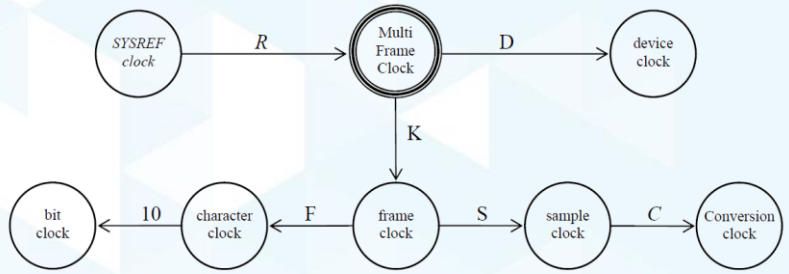
- A circuit used to generate synchronous, phase aligned clocks to various devices in the JESD204 system.
  - Device clocks
  - Link clocks
  - SERDES reference clocks
  - SYSREF
  - Sampling clocks



Challenges

- **Logic Device PHY Layer**
  - SERDES, Transceivers
  - Different
    - Vendors (Xilinx, Intel, etc.)
    - Architectures, Generations (GTX, GTH, GTY)
    - PLL blocks (QPLL, CPLL, ATX, CDR)
    - Speed grades
    - VCO bands
  - Each PLL has own math & constrains.
  - Sharing PLLs between links
  - Control JESD204 link lane rates based on JESD204 Link parameters

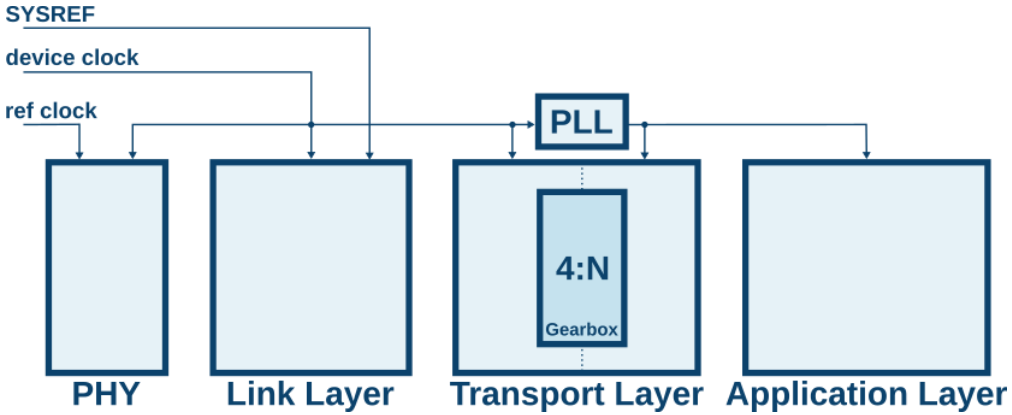




- **Clock Generator**
  - Multiple phase aligned clocks
    - Link configuration (LR/40, LR/66, LMFC/x, etc.)
  - Hand in hand with the PHY Layer PLLs and capabilities
  - JESD204 Subclass 0, 1
    - Single shot pulses or pulse trains
    - SYSREF Timing
  - Multilevel clock trees
  - VCO Bands, Output dividers constrains
  - Phase noise, Jitter considerations

Needs modular Software device drivers to abstract HW implementation details

Challenges



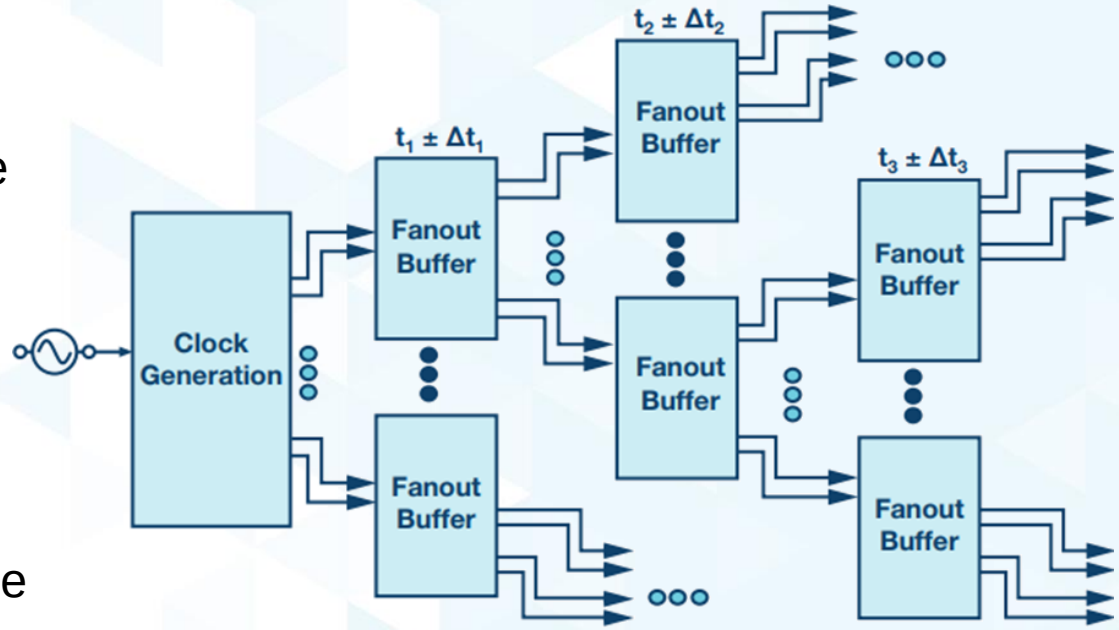
- **Link, Transport and Application Layer**
  - Matching link parameters need to be programmed on both sides of the link
  - Certain modes might require additional Gearbox handling. (more complex clocking)
  - Certain modes might not be supported by HW
  - JESD204B
    - 8b/10b Encoding
    - CGS, ILAS via SYNC strobe
  - JESD204C
    - 64b/66b or 64b/80b Encoding
    - Application Layer Sync handled in SW

Parameter*	Description
DID	Device identification
LID	Lane identification
F	Octets per frame
K	Frames per multi-frame
L	Number of lanes per converter device
N	Converter resolution
N'	Number of bits per sample (recommended to be multiple of 4)
SCR	Scrambling enabled/disabled
HD	High-density (Single sample split over multiple lanes)
JESDV	JESD204 Version (JESD204A, JESD204B, JESD204C)
SUBCLASSV	JESD204B Subclass (0, 1, 2)

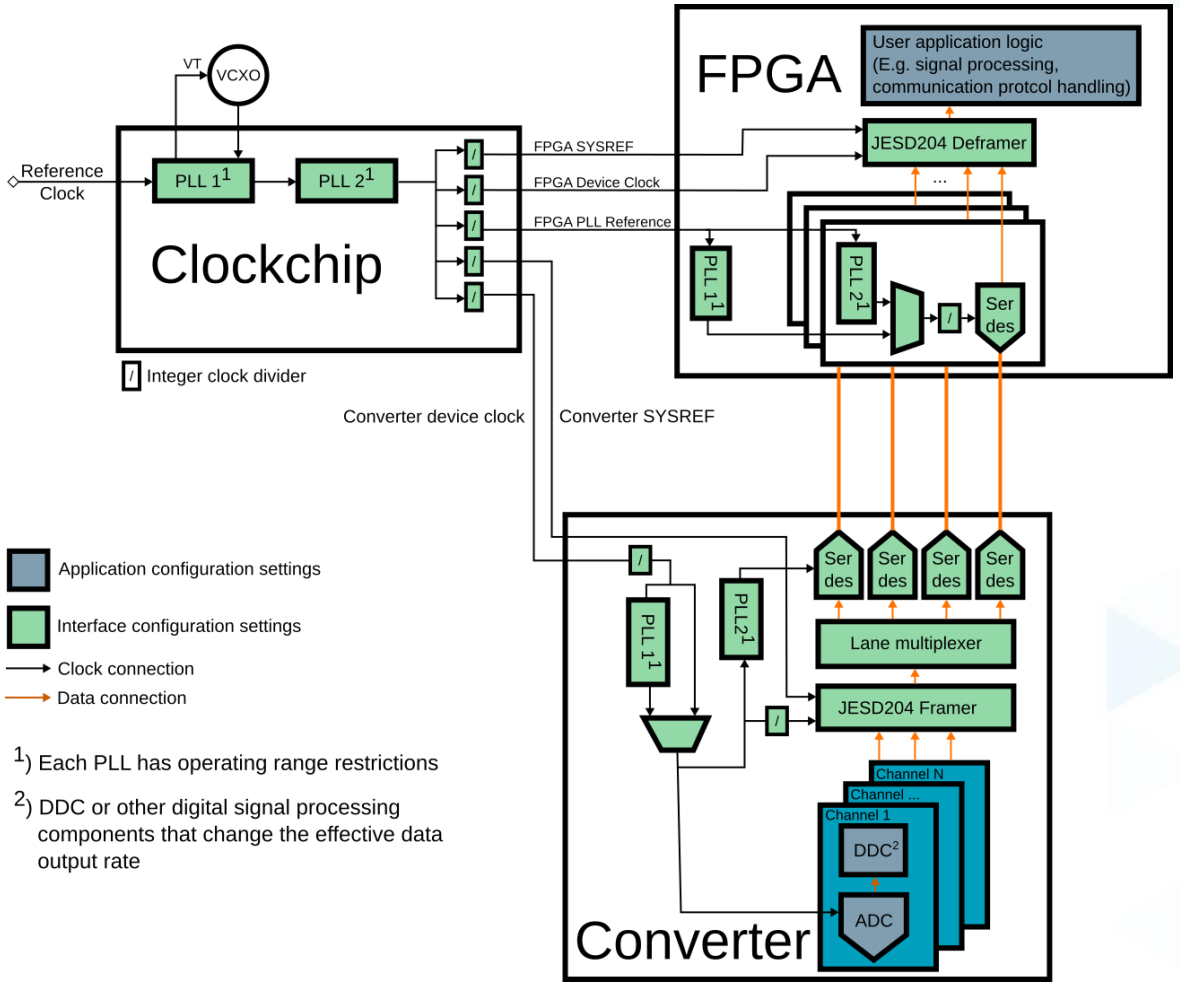
* Table is a excerpt of the most important parameters

Distributed Multichip Clock Synchronization

- **Requirements**
  - Deterministically align clocks and minimize channel skews across all outputs and layers in a clocking tree
  - Meet setup and hold times for SYSREF signals relative to the device clock at each data converter (TRX)
- **Problems**
  - Large number of clocks required
  - Maintain deterministic phase accuracy across multiple levels of clock expansion
  - Maintain clock jitter, phase noise performance across all tree layers and avoid extra clock spurs
  - Mechanics to synchronize multiple-device clock tree structures which are not based on simple clock buffers.



Challenges of high-speed links



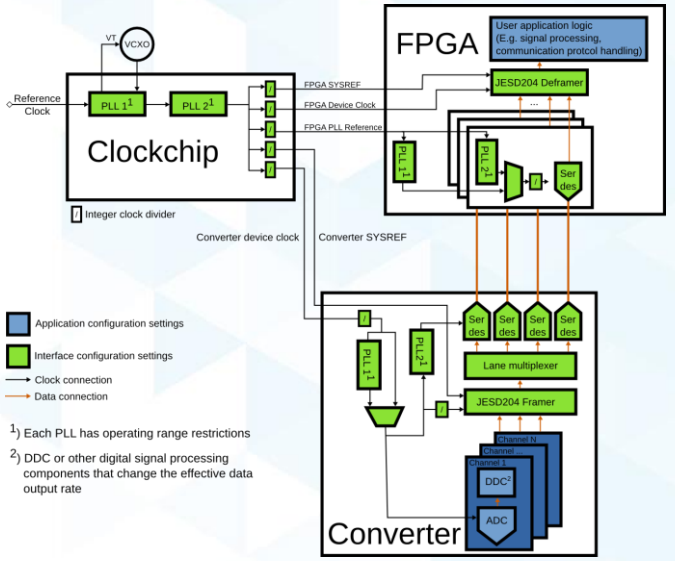
- High-speed serial links are **complex**
  - Lots of interconnected components
    - Clockchip, Converter, FPGA
  - Many PLLs and clock dividers
    - With different constraints on the operating range
  - Hundreds of registers to program
  - Transceivers in FPGAs differ between vendors and generations
  - If any component along the link fails, the whole link fails
    - Difficult to find initial working configuration set
    - Difficult to debug
- **Challenge:** Alleviate complexity allowing end users to manage the system from an application perspective

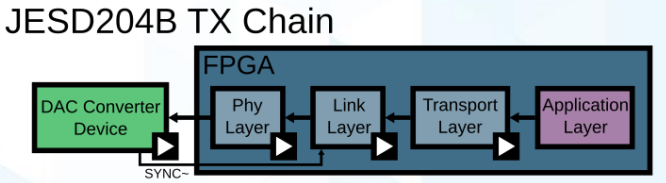
JESD204 Interface Framework

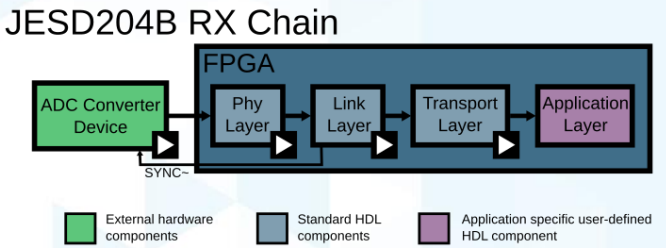
ADI-JESD204
ADI JESD204 INTERFACE FRAMEWORK

ANALOG DEVICES
AHEAD OF WHAT'S POSSIBLE™

- System-level integrated HDL and software framework covering the whole stack
  - Hardware: Reference and rapid prototyping systems
  - HDL: Components for JESD204 protocol handling
  - Software: Drivers to manage clock-chips, converters and HDL IP Cores
- Components have been co-designed for improved interoperability
- Key features
  - Automatic interface configuration based on application settings
    - High-level API
  - Dynamic re-configuration
  - Improved diagnostics
- ADI provides full stack reference designs
  - Works out of the box
  - Starting point for development of custom designs







Third Party Tool Integration
Matlab/Simulink Python
GNU Radio

Software Reference Designs
Configures drivers for HDL and hardware reference design

Software System Libraries
Provides unified interface (API)
Implements common system tasks

Software Drivers
Manages Hardware Components
Manages HDL Components

HDL Reference Designs
Instanciates HDL IP for hardware reference design

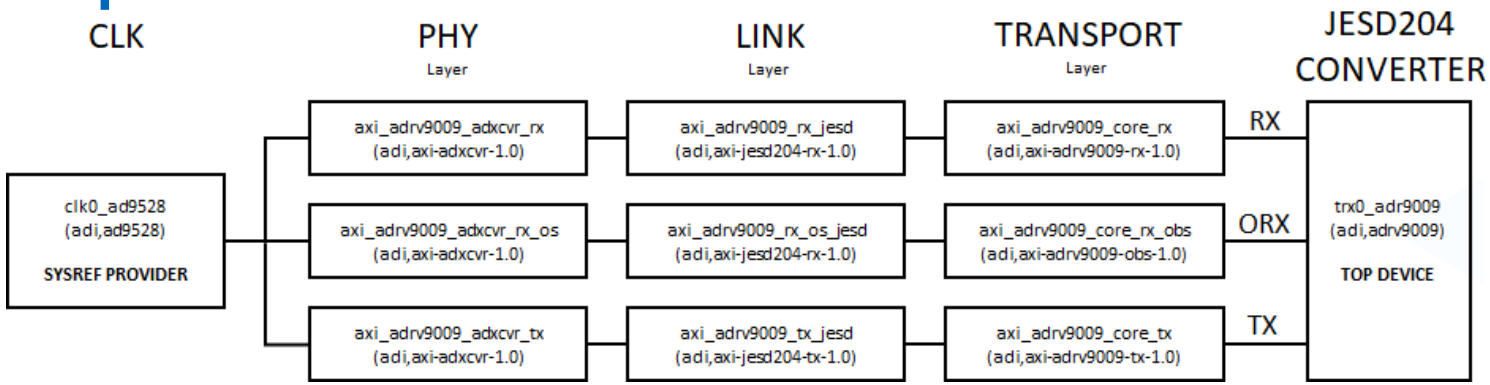
HDL IP Components
Phy Layer Transport Layer
Link Layer DMA

Hardware Reference Desgins
Rapid prototyping boards
Evaluation boards

Hardware Components
Converter Power
Clockchip Analog Frontend

GRCon

Scope & Requirements



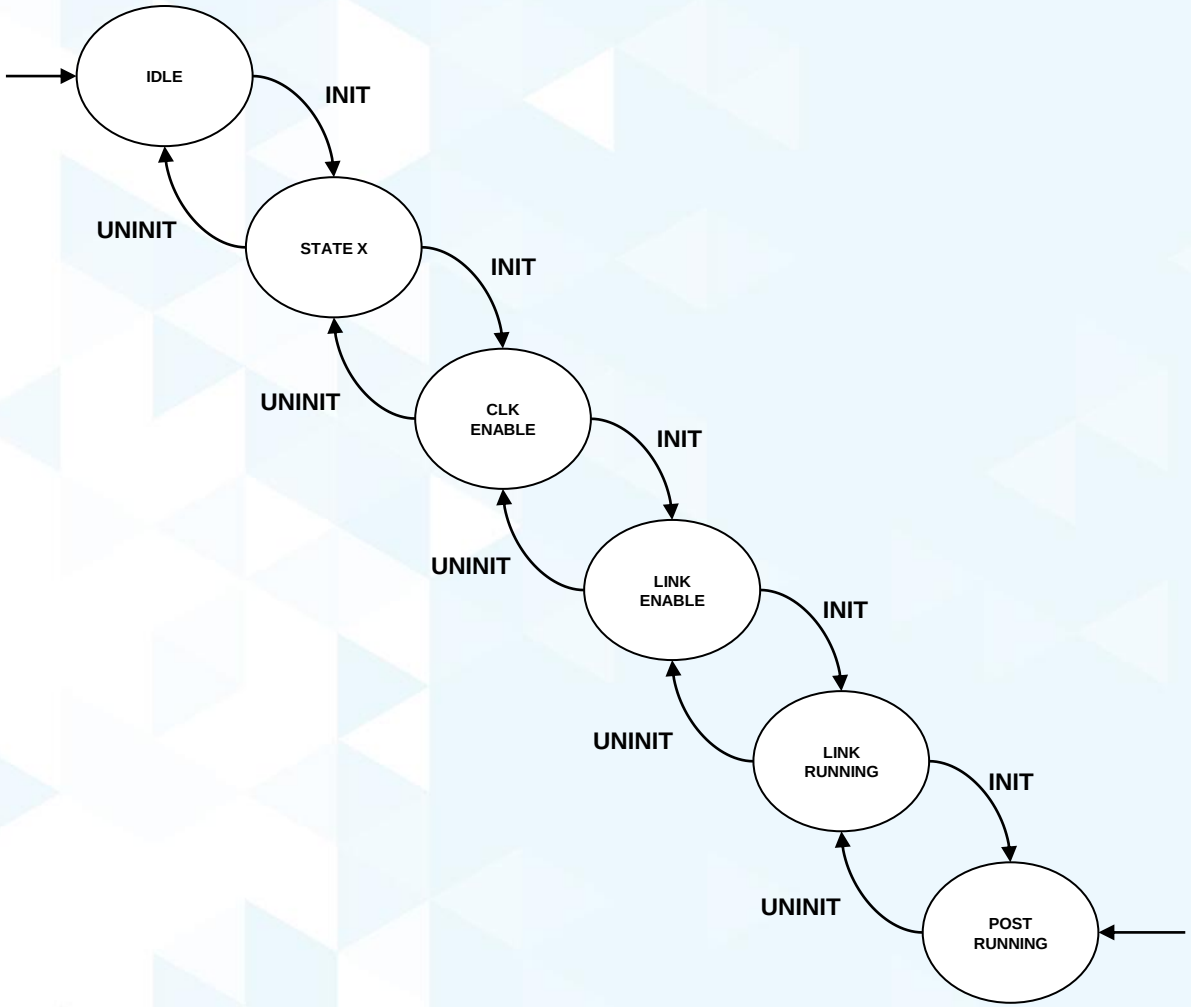
- Multiple interdependent device drivers instantiated via different busses
  - All must be fully probed before the link can start
- Link establishment is a sequence of finite steps with all link components involved at certain times
- Link parameter propagation
- Error reporting and handling
- Avoid boilerplate code that can be shared across component drivers
- Multi-chip synchronization requirements beyond Subclass1 deterministic latency
- Simple way to describe links, flowgraphs and dependencies

Modular design requires
A New JESD204 Linux Kernel Framework

A New JESD204 FSM Linux Kernel Framework

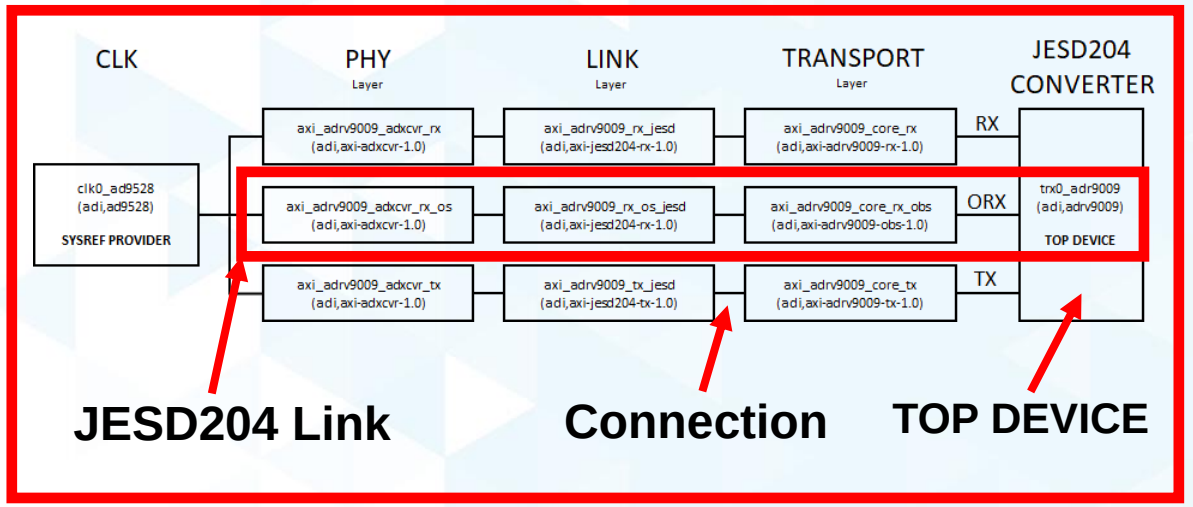
JESD204 FSM Linux Kernel Framework

- The JESD204 Linux Kernel Framework is a Finite State Machine (FSM)
- Synchronize multiple Linux device drivers to be able to properly bring-up & manage a single or multiple JESD204 links
- The JESD204 link bring-up and management can be complicated
- Requires that many actors (device drivers), be in sync with each other, in various link bring-up states/stages
  - Not just at boot-time, but also during run-time, in case a link is going to be reconfigured or breaks and must recover
- Typical components of an JESD204 link are
  - Physical layer (PHY)
  - Link layer (LL)
  - Transport layer (TPL)
  - High speed converter device (PHY, LL, TPL)
  - Clocking layer
  - Application layer
- Manage constrains and inter-dependencies



JESD204 FSM Linux Kernel Framework

- JESD204 Linux Kernel Framework
  - Hooks into all the drivers that participate in the link management (bring-up/bring-down)
  - Each driver provides a set of callbacks for each state that it supports
- The relationship between the devices is defined in the device-tree
- The relationship is called a **connection**
  - Not to re-use the term link, which can cause confusion with the term link from the JESD204 standard
- The whole group of devices, is actually a graph (or **topology**), with a single **top-level** device



TOPOLOGY

Design Principles

- A group of devices shall be named a **topology** (or informally graph or tree)
- Each device driver must **register with the JESD204 framework** to be able to take part in a topology
- A device may only be part of a topology, if it is defined in the device-tree (or other configuration mechanism) via a '**jesd204-device**' node/definition.
- Each topology shall have a single **top-level device**
  - For a multi-chip topology, one will be picked to be the top-level one.
  - For IIO devices, it is assumed that this device will also register the IIO buffer.
- A top-level device may be defined via '**jesd204-top-device = <ID>**'
  - ID is a number defining the topology ID, to be able to specify more topologies
- The top-level device defines the JESD204 link IDs in the device-tree
  - Using '**jesd204-link-ids**' array property
  - The order in this array, is the order in which the JESD204 links are initialized

```
// SPDX-License-Identifier: GPL-2.0
/*
 * Analog Devices ADRV9009 (via jesd204-fsm)
 * https://wiki.analog.com/resources/eval/user-guides/adrv9009
 * https://wiki.analog.com/resources/tools-software/linux-drivers/iio-transceiver/adrv9009
 * https://wiki.analog.com/resources/tools-software/linux-software/adrv9009_advanced_plugin
 *
 * hdl_project: <adrv9009/zc706>
 * board_revision: <>
 *
 * Copyright (C) 2020 Analog Devices Inc.
 */

#include "zynq-zc706-adv7511-adrv9009.dts"

#include <dt-bindings/iio/adc/adi,adrv9009.h>

&trx0_adrv9009 {
    jesd204-device;
    #jesd204-cells = <2>;
    jesd204-top-device = <0>; /* This is the TOP device */
    jesd204-link-ids = <DEFRAMER_LINK_TX FRAMER_LINK_RX FRAMER_LINK_ORX>;

    jesd204-inputs =
        <&axi_adrv9009_rx_jesd 0 FRAMER_LINK_RX>,
        <&axi_adrv9009_rx_os_jesd 0 FRAMER_LINK_ORX>,
        <&axi_adrv9009_core_tx 0 DEFRAMER_LINK_TX>;

    /delete-property/ interrupts;
};

&axi_adrv9009_core_tx {
    jesd204-device;
    #jesd204-cells = <2>;
    jesd204-inputs = <&axi_adrv9009_tx_jesd 0 DEFRAMER_LINK_TX>;
};

&axi_adrv9009_rx_jesd {
    jesd204-device;
    #jesd204-cells = <2>;
    jesd204-inputs = <&axi_adrv9009_adxcvr_rx 0 FRAMER_LINK_RX>;
};
```

```
&axi_adrv9009_rx_os_jesd {
    jesd204-device;
    #jesd204-cells = <2>;
    jesd204-inputs = <&axi_adrv9009_adxcvr_rx_os 0 FRAMER_LINK_ORX>;
};

&axi_adrv9009_tx_jesd {
    jesd204-device;
    #jesd204-cells = <2>;
    jesd204-inputs = <&axi_adrv9009_adxcvr_tx 0 DEFRAMER_LINK_TX>;
};

&axi_adrv9009_adxcvr_rx {
    jesd204-device;
    #jesd204-cells = <2>;
    jesd204-inputs =  <&clk0_ad9528 0 FRAMER_LINK_RX>;
    clocks = <&clk0_ad9528 1>;
    clock-names = "conv";
};

&axi_adrv9009_adxcvr_rx_os {
    jesd204-device;
    #jesd204-cells = <2>;
    jesd204-inputs =  <&clk0_ad9528 0 FRAMER_LINK_ORX>;
    clocks = <&clk0_ad9528 1>;
    clock-names = "conv";
};

&axi_adrv9009_adxcvr_tx {
    jesd204-device;
    #jesd204-cells = <2>;
    jesd204-inputs =  <&clk0_ad9528 0 DEFRAMER_LINK_TX>;
    clocks = <&clk0_ad9528 1>;
    clock-names = "conv";
};

&clk0_ad9528 {
    jesd204-device;
    #jesd204-cells = <2>;
    jesd204-sysref-provider;

    adi,sysref-pattern-mode = <SYSREF_PATTERN_NSHOT>;
    /delete-property/ adi,sysref-request-enable;
};
```

Design Principles (cont.)

- All devices in a topology must go through the **same states together** when bringing up a link and in the same order reverse in reverse when bringing down or rolling back
  - Example: all 8 devices must go from S0 to S9 together, and S9 to S0 together
- When going through each state, **each device-driver will provide its own set of callbacks** for what to do in each state
  - If a callback it is not provided, it is assumed that the device-driver doesn't care about that specific state, and the transition will continue
- When an **error occurs** in any of the states, the states should automatically be **rolled back** from the state that has errored back to the initial/idle state
  - Going from S0 to S9 and S3 faults, the transition will be S0, S1, S2, S3, S2, S1, S0 (in perfect symmetry)
- **Rolling back doesn't stop** even when any of the states errors out
  - It is of higher priority to reach back to IDLE state, than to stop when rolling back

Design Principles (cont.)

- Each callback (in the driver) must return either:
  - **JESD204_STATE_CHANGE_DONE** (value 1) or
  - **JESD204_STATE_CHANGE_DEFER** (value 0) or
  - **JESD204_STATE_CHANGE_ERROR** or any other negative value.
- The decision was made for **JESD204_STATE_CHANGE_DONE** to be 1, so that when a new driver implements a callback for a framework, “return 0” doesn't mean “DONE”
  - i.e. accidental/unwanted state transitions
- The **JESD204_STATE_CHANGE_DEFER** is important if a state should stop (but not rollback) and wait for an external call (a thread/retry mechanism) to restart the FSM and continue from the current state
  - When transitioning from state S0 to S9, and S4 calls for a DEFER, the FSM will stop at S4, and an external entity (retry loop, workq, interrupt, etc) would call the FSM to continue the transition up to S9
  - The DEFER mechanism/logic allows us to pause a transition of states if any device (in the topology) calls for it (because it isn't ready yet)
- For any particular state, the callbacks of the top-level device must be called last
- There can be only a **single** device that can act as a **SYSREF provider** in a topology
  - Defining more than one will fail the initialization of the topology

TL;DR - show me the code

- The current source code of the JESD204 Linux framework resides in
  - https://github.com/analogdevicesinc/linux/blob/master/drivers/jesd204/

- It is comprised of the current source files:
  - jesd204-core.c the core file of the framework - it reads the device-tree, constructs the topology
  - jesd204-fsm.c the entire FSM logic
  - jesd204-sysfs.c the Linux sysfs code to export files for debug/control/etc under /sys/bus/jesd204/devices/jesd204:X
  - jesd204-priv.h internal framework structures/functions to be shared inside the framework
  - include/linux/jesd204/jesd204.h API definitions to be used by drivers registering with the framework

How does it work?

- A typical driver needs to provide some data to the framework
- The driver needs to call **devm_jesd204_dev_register().**

```
jdev = devm_jesd204_dev_register(&spi->dev, &jesd204_adrv9009_init);
if (IS_ERR(jdev))
    return PTR_ERR(jdev);
```

- All drivers must finally call the **jesd204_fsm_start()** on their object from the framework. This is true for all devices, even the ones that are not top-level devices.

```
ret = jesd204_fsm_start(jdev, JESD204_LINKS_ALL);
```

- There's an equivalent **jesd204_fsm_stop()** that will stop the FSM.

```
ret = jesd204_fsm_stop(jdev, JESD204_LINKS_ALL);
```

- Proper function of the FSM relies on the drivers correctly using the framework and that that connections between devices be properly defined in the device-tree.

```
static const struct jesd204_dev_data jesd204_adrv9009_init = {
    .state_ops = {
        [JESD204_OP_DEVICE_INIT] = {
            .per_device = adrv9009_jesd204_uninit,
        },
        [JESD204_OP_LINK_INIT] = {
            .per_link = adrv9009_jesd204_link_init,
        },
        [JESD204_OP_LINK_SETUP] = {
            .per_device = adrv9009_jesd204_link_setup,
            .mode = JESD204_STATE_OP_MODE_PER_DEVICE,
            .post_state_sysref = true,
        },
        /* SNIP */
        [JESD204_OP_CLOCKS_ENABLE] = {
            .per_link = adrv9009_jesd204_clks_enable,
        },
        [JESD204_OP_LINK_ENABLE] = {
            .per_link = adrv9009_jesd204_link_enable,
            .post_state_sysref = true,
        },
        [JESD204_OP_LINK_RUNNING] = {
            .per_link = adrv9009_jesd204_link_running,
        },
        [JESD204_OP_OPT_POST_RUNNING_STAGE] = {
            .per_device = adrv9009_jesd204_post_running_stage,
            .mode = JESD204_STATE_OP_MODE_PER_DEVICE,
        },
    },

    .max_num_links = 3,
    .sizeof_priv = sizeof(struct adrv9009_jesd204_priv),
};
```

The initialization data

- A **SYSREF** provider hooks itself with the ***sysref_cb*** hook, but there also must be a device-tree property to mark that this is the SYSREF provider used in the topology.
- Optionally a driver may reserve some memory for private state data via **sizeof_priv** and can be obtained via a **jesd204_dev_priv(jdev)**
- The **links** field is used to define JESD204 links in a static manner in the driver
  - May go away if there aren't any clear use-cases for them; but it could be that some devices allow only a fixed configuration, so these could be useful in those cases
- **max_num_links** - maximum number of JESD204 links that this device supports
  - The actual number will be configured from the device-tree, but it shouldn't exceed this number
- **num_retries** - number of retries in case of error during an FSM start/link-bring-up
- **state_ops** - more below

```
/**
 * struct jesd204_dev_data - JESD204 device initialization data
 * @sysref_cb:          SYSREF callback, if this device/driver supports it
 * @fsm_finished_cb:    callback for when the FSM finishes a series of transitions
 * @sizeof_priv:        amount of data to allocate for private information
 * @links:              JESD204 initial link configuration
 * @max_num_links:      maximum number of JESD204 links this device can support
 * @num_retries:        number of retries in case of error (only for top-level device)
 * @state_ops:          ops for each state transition of type @struct jesd204_state_op
 */
struct jesd204_dev_data {
        jesd204_sysref_cb                       sysref_cb;
        jesd204_fsm_finished_cb                 fsm_finished_cb;
        size_t                                  sizeof_priv;
        const struct jesd204_link               *links;
        unsigned int                            max_num_links;
        unsigned int                            num_retries;
        struct jesd204_state_op                 state_ops[__JESD204_MAX_OPS];
};
```

The State Operations callbacks()

```
/**
 * struct jesd204_state_op - JESD204 device per-state op
 * @mode:               mode for this state op, depending on
 *                      this @per_device or @per_link is called
 * @per_device:         op called for each JESD204 **device**
 *                      during a transition
 * @per_link:           op called for each JESD204 **link**
 *                      individually during a transition
 * @post_state_sysref:  true if a SYSREF should be issued after the state change
 */
struct jesd204_state_op {
        enum jesd204_state_op_mode      mode;
        jesd204_dev_cb                  per_device;
        jesd204_link_cb                 per_link;
        bool                            post_state_sysref;
};
```

- Each driver hooks its callback by adding the proper entry in the **state_ops** array.
- During a state-transition a state callback will be called:
  - Once for each JESD204 link if the mode is default **JESD204_STATE_OP_MODE_PER_LINK**
    - In this case the **per_link** callback is called
  - Once for each device (regardless of the number of JESD204 links per device) if mode is **JESD204_STATE_OP_MODE_PER_DEVICE**
    - in this case the **per_device** callback is called
- Optionally, each state may request a SYSREF call, by setting **post_state_sysref** to true.

```
static const struct jesd204_dev_data jesd204_adrv9009_init = {
    .state_ops = {
        [JESD204_OP_DEVICE_INIT] = {
            .per_device = adrv9009_jesd204_uninit,
        },
        [JESD204_OP_LINK_INIT] = {
            .per_link = adrv9009_jesd204_link_init,
        },
        [JESD204_OP_LINK_SETUP] = {
            .per_device = adrv9009_jesd204_link_setup,
            .mode = JESD204_STATE_OP_MODE_PER_DEVICE,
            .post_state_sysref = true,
        },
        /* SNIP */
        [JESD204_OP_CLOCKS_ENABLE] = {
            .per_link = adrv9009_jesd204_clks_enable,
        },
        [JESD204_OP_LINK_ENABLE] = {
            .per_link = adrv9009_jesd204_link_enable,
            .post_state_sysref = true,
        },
        [JESD204_OP_LINK_RUNNING] = {
            .per_link = adrv9009_jesd204_link_running,
        },
        [JESD204_OP_OPT_POST_RUNNING_STAGE] = {
            .per_device = adrv9009_jesd204_post_running_stage,
            .mode = JESD204_STATE_OP_MODE_PER_DEVICE,
        },
    },

    .max_num_links = 3,
    .sizeof_priv = sizeof(struct adrv9009_jesd204_priv),
};
```

JESD204-FSM link states in a nutshell

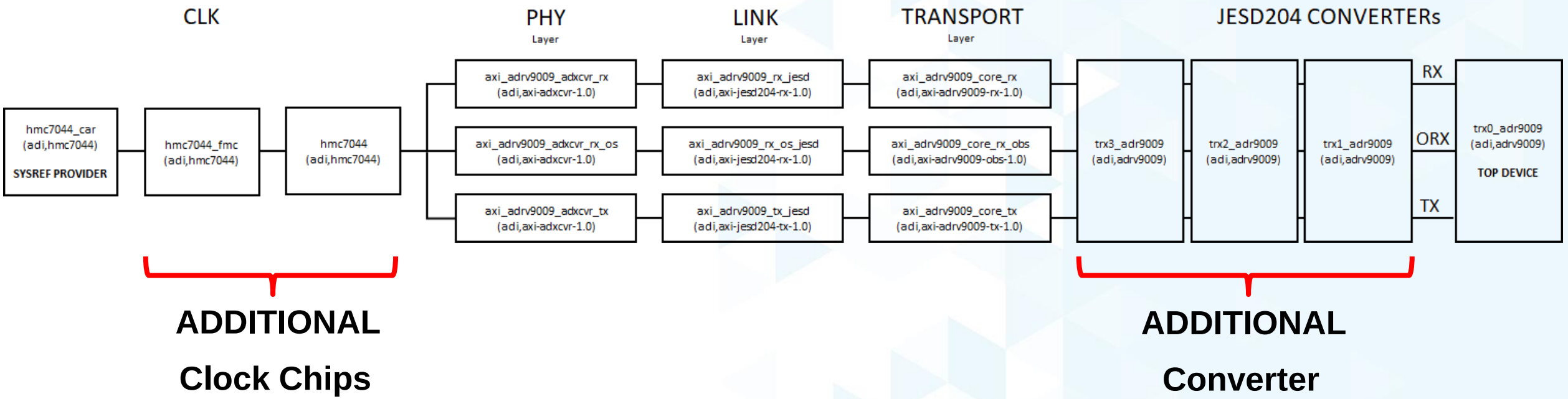
- **LINK_INIT**
  - The JESD204-FSM calls this callback for each JESD204 link defined in the device-tree
  - The TOP device fills in the (struct jesd204_link) parameter for each link.
  - These parameters include all the JESD204 link parameters, the sample rate and SYSREF mode settings, such as continuous or pulsed SYSREF operation.
- **LINK_SUPPORTED**
  - During this state the FSM core, calls for each link of a topology into each device in order to query if this configuration is supported. Whether a configuration is supported or not depends on a number of constrains and synthesis parameters.
  - In case all devices support the configuration, the FSM moves on to the next state. The clock chip drivers use this state to compute the LMFC/LEMC of all links and find it's GCD, so that a common SYSREF frequency can be computed which satisfies all links requirements.
- **LINK_PRE_SETUP**
  - Typically, this state is used by the CLK chip drivers to configure the output channels dedicated as SYSREF, applies the previously computed SYSREF frequency and configures the mode.
- **LINK_SETUP**
  - In this state all actors (link devices) are setup and configured, based on the mode and configuration previously validated.
  - A lot of devices typically found on multi-chip setups, require additional synchronization steps such as MCS (Mult-chip Sync), Phase/NCO Sync, or calibrations which take a lot of time and would benefit from being done in parallel to save some time.
  - For those cases **OPT_SETUP_STAGE1 to OPT_SETUP_STAGE5** can be used to implement these High-Speed converter device specific configuration, synchronization and calibration steps.

JESD204-FSM link states in a nutshell

- **Optional CLK_SYNC_STAGEx**
  - These states can be used by clock chip drivers to implement a Clock Tree Synchronization mechanism. This is typically device specific or might not be supported by the clock chip in question. Right now, there are 3 states reserved for this. Based on the HMC7044 example this is what happens in each state.
  - **CLK_SYNC_STAGE1**
    The SYNC provider and consumers are configured to generate or receive a synchronization request.
  - **CLK_SYNC_STAGE2**
    The synchronization request is issued at the TOP most device in the clock tree. For the HMC7044, this is the SYSREF PROVIDER.
  - **CLK_SYNC_STAGE3**
    In this last CLK SYNC state, the SYNC status of each CLK device is validated.
- **CLOCKS_ENABLE & LINK_ENABLE**
  - Depending on the direction of the JESD204 link, different link components may implement different things. However, as the name implies it's about enabling the JESD204 links.
  - This includes taking the Link Layer cores out of RESET, enabling the SYSREF receivers for SUBCLASS 1 operation, requesting a SYSREF pulse, etc.
- **LINK_RUNNING**
  - In this state all links of a topology assumed running. This is typically being checked in this state. Drivers can also use this state to complete setup and configuration which is required after the JESD204 links are running. In case another state is required drivers can also implement the optional **OPT_POST_RUNNING_STAGE** for these purposes.

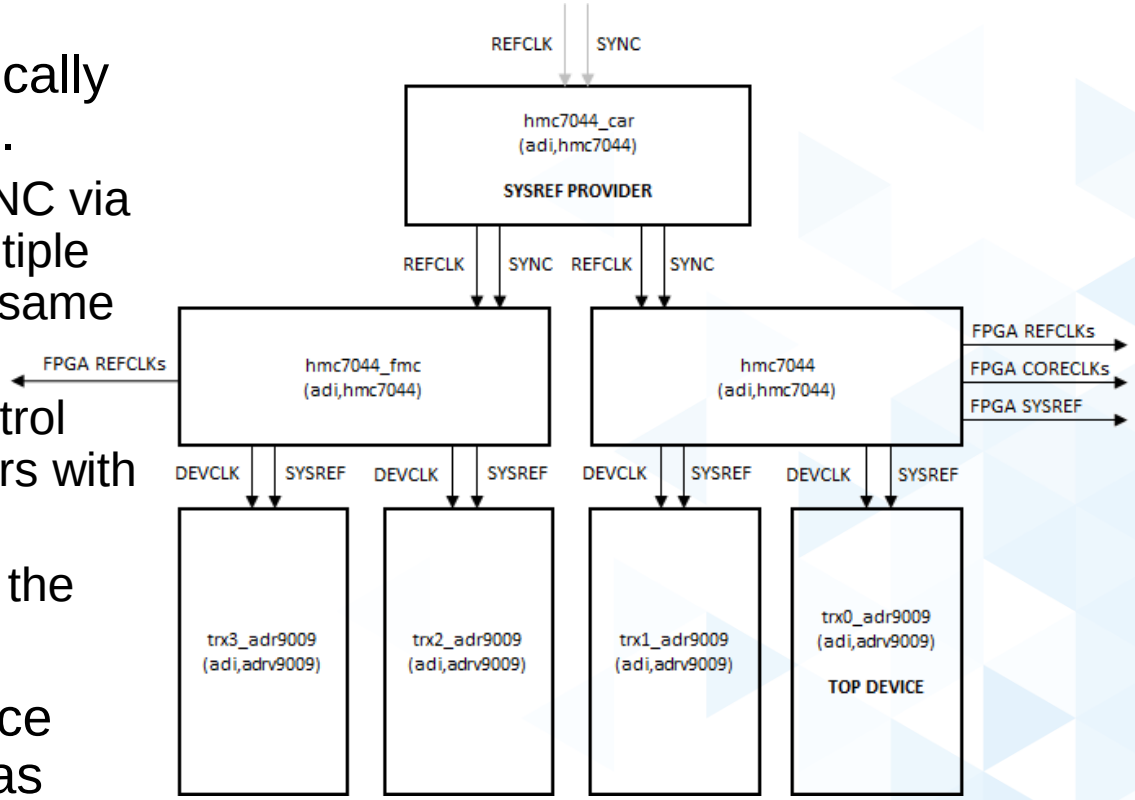
4x ADRV9009 Example Block Diagram

- Below example is taken from ADRV9009-ZU11EG System on Module on the ADRV2CRR-FMC carrier board with an additional FMCOMMS8 FMC board connected.
- In total there are 4x ADRV9009 RF transceivers in this design, with a two-level clock tree.
- The **JESD204-FSM** topology and input connections can be found below.



Clock Tree Setup and Synchronization for 8x8 Channels

- The HMC7044 which is used in this example is specifically designed to offer features to address MCS challenges.
  - An external reference-based synchronization feature (SYNC via PLL2 or RF SYNC only in fanout mode) synchronizes multiple devices, that is, it ensures that all clock outputs start with same rising edge.
  - This operation is achieved by rephasing the SYSREF control unit deterministically, and then restarting the output dividers with this new desired phase.
  - A SYSREF/PULSOR request issued at the TOP device in the clocking tree will propagate down in the hierarchy.
- The SYSREF signal acts as the master timing reference and aligns all the internal dividers from device clocks as well as the local multiframe clocks in each JESD204 transmitter and receiver. SYSREF helps to ensure deterministic latency through the system.



Kernel Boot Messages (devicetree parsing)

```
[    2.504336] jesd204: created con: id=0, topo=0, link=0, /fpga-axi@0/axi-jesd204-tx@84a30000 <-> /fpga-axi@0/axi-adrv9009-tx-hpc@84a04000
[    2.516027] jesd204: created con: id=1, topo=0, link=0, /fpga-axi@0/axi-adxcvr-tx@84a20000 <-> /fpga-axi@0/axi-jesd204-tx@84a30000
[    2.527704] jesd204: created con: id=2, topo=0, link=0, /amba/spi@ff040000/hmc7044@2 <-> /fpga-axi@0/axi-adxcvr-tx@84a20000
[    2.538774] jesd204: created con: id=3, topo=0, link=2, /fpga-axi@0/axi-adxcvr-rx-os@84a60000 <-> /fpga-axi@0/axi-jesd204-rx@84a70000
[    2.550712] jesd204: created con: id=4, topo=0, link=2, /amba/spi@ff040000/hmc7044@2 <-> /fpga-axi@0/axi-adxcvr-rx-os@84a60000
[    2.562043] jesd204: created con: id=5, topo=0, link=1, /fpga-axi@0/axi-adxcvr-rx@84a40000 <-> /fpga-axi@0/axi-jesd204-rx@84a50000
[    2.573721] jesd204: created con: id=6, topo=0, link=1, /amba/spi@ff040000/hmc7044@2 <-> /fpga-axi@0/axi-adxcvr-rx@84a40000
[    2.584797] jesd204: created con: id=7, topo=0, link=1, /amba/spi@ff040000/hmc7044-car@3 <-> /amba/spi@ff050000/hmc7044-fmc@2
[    2.596038] jesd204: created con: id=8, topo=0, link=2, /amba/spi@ff040000/hmc7044-car@3 <-> /amba/spi@ff050000/hmc7044-fmc@2
[    2.607284] jesd204: created con: id=9, topo=0, link=0, /amba/spi@ff040000/hmc7044-car@3 <-> /amba/spi@ff050000/hmc7044-fmc@2
[    2.618543] jesd204: created con: id=10, topo=0, link=1, /fpga-axi@0/axi-jesd204-rx@84a50000 <-> /amba/spi@ff050000/adrv9009-phy-d@1
[    2.630385] jesd204: created con: id=11, topo=0, link=2, /fpga-axi@0/axi-jesd204-rx@84a70000 <-> /amba/spi@ff050000/adrv9009-phy-d@1
[    2.642237] jesd204: created con: id=12, topo=0, link=0, /fpga-axi@0/axi-adrv9009-tx-hpc@84a04000 <-> /amba/spi@ff050000/adrv9009-phy-d@1
[    2.654556] jesd204: created con: id=13, topo=0, link=1, /amba/spi@ff050000/adrv9009-phy-d@1 <-> /amba/spi@ff050000/adrv9009-phy-c@0
[    2.666385] jesd204: created con: id=14, topo=0, link=2, /amba/spi@ff050000/adrv9009-phy-d@1 <-> /amba/spi@ff050000/adrv9009-phy-c@0
[    2.678241] jesd204: created con: id=15, topo=0, link=0, /amba/spi@ff050000/adrv9009-phy-d@1 <-> /amba/spi@ff050000/adrv9009-phy-c@0
[    2.690069] jesd204: created con: id=16, topo=0, link=1, /amba/spi@ff050000/hmc7044-fmc@2 <-> /amba/spi@ff040000/hmc7044@2
[    2.701050] jesd204: created con: id=17, topo=0, link=2, /amba/spi@ff050000/hmc7044-fmc@2 <-> /amba/spi@ff040000/hmc7044@2
[    2.712036] jesd204: created con: id=18, topo=0, link=0, /amba/spi@ff050000/hmc7044-fmc@2 <-> /amba/spi@ff040000/hmc7044@2
[    2.723063] jesd204: created con: id=19, topo=0, link=1, /amba/spi@ff050000/adrv9009-phy-c@0 <-> /amba/spi@ff040000/adrv9009-phy-b@1
[    2.734892] jesd204: created con: id=20, topo=0, link=2, /amba/spi@ff050000/adrv9009-phy-c@0 <-> /amba/spi@ff040000/adrv9009-phy-b@1
[    2.746750] jesd204: created con: id=21, topo=0, link=0, /amba/spi@ff050000/adrv9009-phy-c@0 <-> /amba/spi@ff040000/adrv9009-phy-b@1
[    2.758620] jesd204: created con: id=22, topo=0, link=1, /amba/spi@ff040000/adrv9009-phy-b@1 <-> /amba/spi@ff040000/adrv9009-phy@0
[    2.770273] jesd204: created con: id=23, topo=0, link=2, /amba/spi@ff040000/adrv9009-phy-b@1 <-> /amba/spi@ff040000/adrv9009-phy@0
[    2.781956] jesd204: created con: id=24, topo=0, link=0, /amba/spi@ff040000/adrv9009-phy-b@1 <-> /amba/spi@ff040000/adrv9009-phy@0
[    2.793666] jesd204: /amba/spi@ff040000/adrv9009-phy@0: JESD204[0] transition uninitialized -> initialized
[    2.803204] jesd204: /amba/spi@ff040000/adrv9009-phy@0: JESD204[1] transition uninitialized -> initialized
[    2.812805] jesd204: /amba/spi@ff040000/adrv9009-phy@0: JESD204[2] transition uninitialized -> initialized
[    2.822405] jesd204: found 14 devices and 1 topologies
```

Kernel Boot Messages (FSM start)

```
[    5.643202] axi_adxcvr 84a40000.axi-adxcvr-rx: AXI-ADXCVR-RX (17.04.a) using CPLL on GTH4 at 0x84A40000. Number of lanes: 8.
[    5.656624] axi_adxcvr 84a60000.axi-adxcvr-rx-os: AXI-ADXCVR-RX (17.04.a) using CPLL on GTH4 at 0x84A60000. Number of lanes: 8.
[    5.669150] axi_adxcvr 84a20000.axi-adxcvr-tx: AXI-ADXCVR-TX (17.04.a) using QPLL on GTH4 at 0x84A20000. Number of lanes: 16.
[    5.680916] axi-jesd204-rx 84a50000.axi-jesd204-rx: AXI-JESD204-RX (1.07.a) at 0x84A50000. Encoder 8b10b, width 4/4, lanes 8, jesd204-fsm.
[    5.693744] axi-jesd204-rx 84a70000.axi-jesd204-rx: AXI-JESD204-RX (1.07.a) at 0x84A70000. Encoder 8b10b, width 4/4, lanes 8, jesd204-fsm.
[    5.706522] axi-jesd204-tx 84a30000.axi-jesd204-tx: AXI-JESD204-TX (1.06.a) at 0x84A30000. Encoder 8b10b, width 4/4, lanes 16, jesd204-fsm.
[    5.799989] cf_axi_adc 84a00000.axi-adrv9009-rx-hpc: ADI AIM (10.01.b) at 0x84A00000 mapped to 0x(____ptrval____), probed ADC ADRV9009-X4 as MASTER
[    5.831861] cf_axi_dds 84a04000.axi-adrv9009-tx-hpc: Analog Devices CF_AXI_DDS_DDS MASTER (9.01.b) at 0x84A04000 mapped to 0x(____ptrval____), probed DDS ADRV9009-X4
[    5.911723] jesd204: /amba/spi@ff040000/adrv9009-phy@0,jesd204:0,parent=spi1.0: JESD204[0] transition idle -> device_init
[    5.922668] jesd204: /amba/spi@ff040000/adrv9009-phy@0,jesd204:0,parent=spi1.0: JESD204[1] transition idle -> device_init
[    5.933612] jesd204: /amba/spi@ff040000/adrv9009-phy@0,jesd204:0,parent=spi1.0: JESD204[2] transition idle -> device_init
[    5.944613] jesd204: /amba/spi@ff040000/adrv9009-phy@0,jesd204:0,parent=spi1.0: JESD204[0] transition device_init -> link_init
[    5.955998] jesd204: /amba/spi@ff040000/adrv9009-phy@0,jesd204:0,parent=spi1.0: JESD204[1] transition device_init -> link_init
[    5.967378] jesd204: /amba/spi@ff040000/adrv9009-phy@0,jesd204:0,parent=spi1.0: JESD204[2] transition device_init -> link_init
[    5.978818] jesd204: /amba/spi@ff040000/adrv9009-phy@0,jesd204:0,parent=spi1.0: JESD204[0] transition link_init -> link_supported
[    5.990459] jesd204: /amba/spi@ff040000/adrv9009-phy@0,jesd204:0,parent=spi1.0: JESD204[1] transition link_init -> link_supported
[    6.002099] jesd204: /amba/spi@ff040000/adrv9009-phy@0,jesd204:0,parent=spi1.0: JESD204[2] transition link_init -> link_supported
[    6.018987] jesd204: /amba/spi@ff040000/adrv9009-phy@0,jesd204:0,parent=spi1.0: JESD204[0] transition link_supported -> link_pre_setup
[    6.031067] jesd204: /amba/spi@ff040000/adrv9009-phy@0,jesd204:0,parent=spi1.0: JESD204[1] transition link_supported -> link_pre_setup
[    6.043147] jesd204: /amba/spi@ff040000/adrv9009-phy@0,jesd204:0,parent=spi1.0: JESD204[2] transition link_supported -> link_pre_setup
[    6.055227] jesd204: /fpga-axi@0/axi-jesd204-rx@84a50000,jesd204:8,parent=84a50000.axi-jesd204-rx: Possible instantiation for multiple chips; HDL lanes 8, Link[1] lanes 2
[    6.070438] jesd204: /fpga-axi@0/axi-jesd204-rx@84a70000,jesd204:10,parent=84a70000.axi-jesd204-rx: Possible instantiation for multiple chips; HDL lanes 8, Link[2] lanes 2
[    6.085839] jesd204: /fpga-axi@0/axi-jesd204-tx@84a30000,jesd204:12,parent=84a30000.axi-jesd204-tx: Possible instantiation for multiple chips; HDL lanes 16, Link[0] lanes 4
[   21.792984] jesd204: /amba/spi@ff040000/adrv9009-phy@0,jesd204:0,parent=spi1.0: JESD204[0] transition clocks_enable -> link_enable
[   21.804711] jesd204: /amba/spi@ff040000/adrv9009-phy@0,jesd204:0,parent=spi1.0: JESD204[1] transition clocks_enable -> link_enable
[   21.816436] jesd204: /amba/spi@ff040000/adrv9009-phy@0,jesd204:0,parent=spi1.0: JESD204[2] transition clocks_enable -> link_enable
[   21.863266] jesd204: /amba/spi@ff040000/adrv9009-phy@0,jesd204:0,parent=spi1.0: JESD204[0] transition link_enable -> link_running
[   21.874907] jesd204: /amba/spi@ff040000/adrv9009-phy@0,jesd204:0,parent=spi1.0: JESD204[1] transition link_enable -> link_running
[   21.886553] jesd204: /amba/spi@ff040000/adrv9009-phy@0,jesd204:0,parent=spi1.0: JESD204[2] transition link_enable -> link_running
[   22.000757] adrv9009 spi2.1: adrv9009_info: adrv9009 Rev 192, Firmware 6.0.2 API version: 3.6.0.5 successfully initialized via jesd204-fsm
[   22.115736] adrv9009 spi2.0: adrv9009_info: adrv9009 Rev 192, Firmware 6.0.2 API version: 3.6.0.5 successfully initialized via jesd204-fsm
[   22.230474] adrv9009 spi1.1: adrv9009_info: adrv9009 Rev 192, Firmware 6.0.2 API version: 3.6.0.5 successfully initialized via jesd204-fsm
[   22.345219] adrv9009 spi1.0: adrv9009_info: adrv9009-x4 Rev 192, Firmware 6.0.2 API version: 3.6.0.5 successfully initialized via jesd204-fsm
[   22.357904] jesd204: /amba/spi@ff040000/adrv9009-phy@0,jesd204:0,parent=spi1.0: JESD204[0] transition link_running -> opt_post_running_stage
[   22.370504] jesd204: /amba/spi@ff040000/adrv9009-phy@0,jesd204:0,parent=spi1.0: JESD204[1] transition link_running -> opt_post_running_stage
[   22.383099] jesd204: /amba/spi@ff040000/adrv9009-phy@0,jesd204:0,parent=spi1.0: JESD204[2] transition link_running -> opt_post_running_stage
```

PHY

LINK

TPL

FSM Start

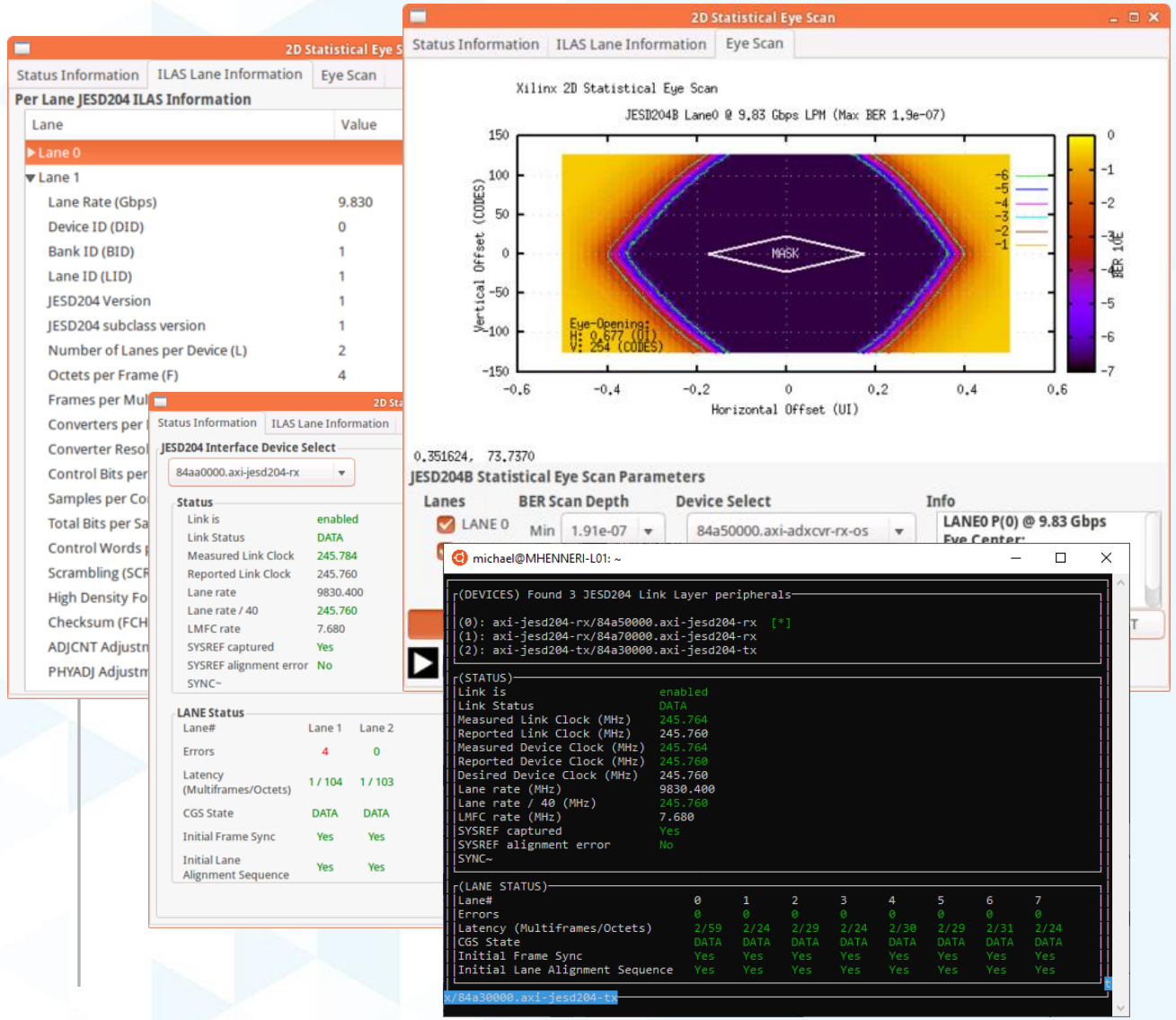
FSM Done

JESD204 Diagnostic Tools

ADI-JESD204
ADI JESD204 INTERFACE FRAMEWORK

ANALOG DEVICES
AHEAD OF WHAT'S POSSIBLE™

- Statistical Eyescan, Bit-error-rate monitoring
  - Detect electrical signal integrity issues
- Clock rate monitoring for all system clocks
  - Detect bad clock wiring
  - Detect clock failures
- Initial lane sequence monitoring and verification
  - Detect lane swaps
- Lane arrival monitoring (relative to SYSREF)
  - Detect potential sources of non-deterministic latency
- SYSREF alignment monitoring
  - Detect SYSREF timing and configuration issues
- Continuous monitoring
  - Application is notified as soon as failure occurs



GRCon

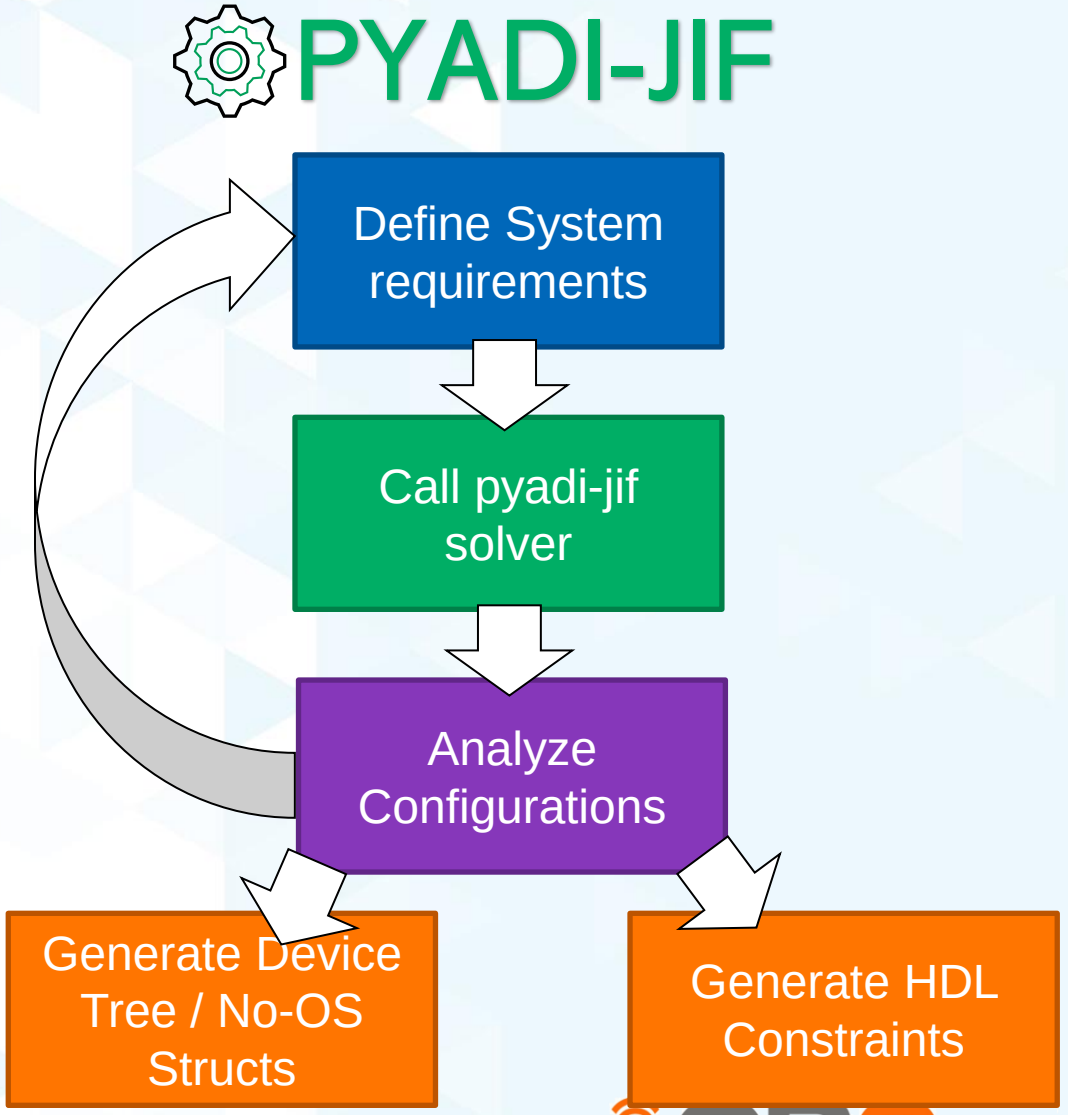
Next Steps

- Standardize and export all Link status and debug information from the framework core
- Convert our connector mechanism to OF graph
- Infrastructure allowing to add auxiliary user/application specific states
- Make FSM by default less verbose
- Centralized
  - PHY layer PRBS testing methods via sysfs
  - FSM start/stop/restart/resume via sysfs
- Better PHY layer resource management.
  - Sharing PLLs between links when possible
  - Auto-switching assigned PLLs

Mainline / Upstream

pyadi-jif: JESD Framework Configuration Tool

- Configuration tooling for the ADI JESD Interface Framework and clock infrastructure
- Manages
  - Converter SERDES, datapath, and clock requirements
  - FPGA transceiver clocking requirements
  - Clock chip constraints and references
- Leverages non-linear solvers to quickly solve, validate, and optimize configuration across the entire system
- Includes python library, CLI, and GUIs
- Drives device tree, HDL constraints, and baremetal drivers

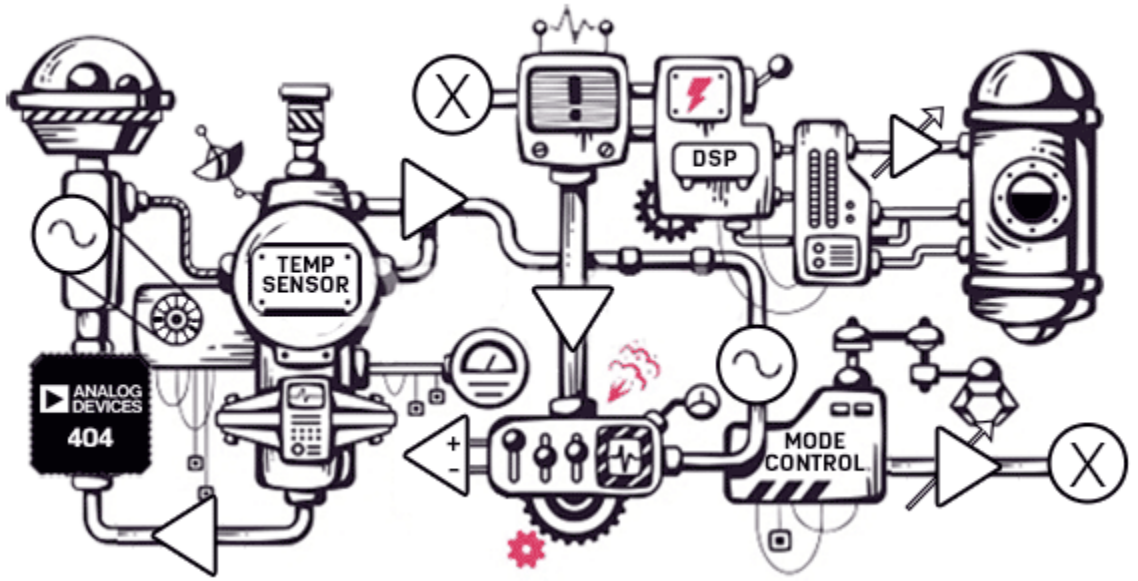


Conclusions

- Standardizing interfaces and modular design
  - Allows different objects to interact
  - Allows code reuse
  - Hides implementation details
  - Limits complexity
  - Provides interoperability, scale and maintainability
- The jesd204-fsm kernel framework/subsystem greatly simplifies jesd204 converter systems
- Porting new devices to the framework is a straightforward exercise.
- Free and Open-Source Software and Ecosystems

References

- **Framework**
  - https://wiki.analog.com/resources/fpga/peripherals/jesd204
  - https://wiki.analog.com/resources/tools-software/linux-drivers/jesd204/jesd204-fsm-framework
- **Link Layer drivers**
  - https://wiki.analog.com/resources/tools-software/linux-drivers/jesd204/axi_jesd204_tx
  - https://wiki.analog.com/resources/tools-software/linux-drivers/jesd204/axi_jesd204_rx
- **PHY Layer driver**
  - https://wiki.analog.com/resources/tools-software/linux-drivers/jesd204/axi_adxcvr
- **Transport Layer drivers**
  - https://wiki.analog.com/resources/tools-software/linux-drivers/iio-dds/axi-dac-dds-hdl
  - https://wiki.analog.com/resources/tools-software/linux-drivers/iio-adc/axi-adc-hdl
- **Tools**
  - https://wiki.analog.com/resources/tools-software/linux-software/jesd_eye_scan
  - https://wiki.analog.com/resources/tools-software/linux-software/jesd_status
  - https://github.com/analogdevicesinc/pyadi-jif
- **Examples**
  - https://wiki.analog.com/resources/eval/user-guides/adrv9009-zu11eg
  - https://wiki.analog.com/resources/eval/user-guides/quadmxfe



Ahhh, technology. We can't find that page.

Thanks

Q & A