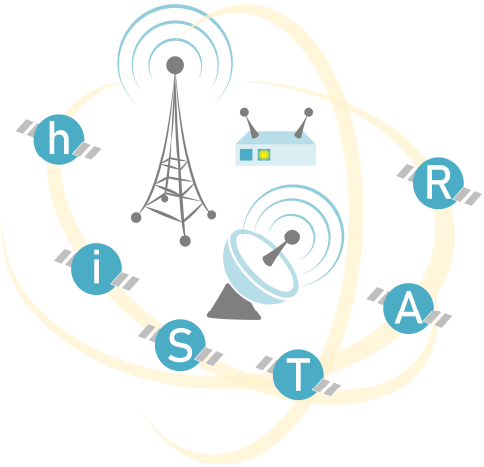
Science Fund
of the Republic of Serbia

Grant Agreement No.: 7750824

Call: IDEJE

Hybrid Integrated Satellite and Terrestrial Access Network



D3.2: FPGA-based accelerators for high-performance 5G/Sat transceivers

Work package	WP 3
Subactivity	T3.2
Due date	31.12.2023.
Submission date	01.01.2024.
Deliverable lead	ETF
Version	1.0
Authors	Dragomir El Mezeni, Vladimir Petrović, Haris Turkmanović
Reviewers	Lazar Saranovac, Dejan Drajič, Goran Đorđević

Document Revision History

Version	Date	Description of change	List of contributor(s)
V1.0	01/01/2024	1st version of D3.2	

ACKNOWLEDGMENT

Science Fund of the Republic of Serbia

This deliverable has been written in the context of hi-STAR project who has received funding from the Science Fund of the Republic of Serbia, Programme IDEJE under grant agreement n° 7750284.

EXECUTIVE SUMMARY

The hi-STAR project addresses one of the most critical challenges for the next generation wireless networks, which is integration of non-terrestrial networks with terrestrial 5G network. The general objective of the project is to develop flexible framework for integrated terrestrial 5G and Low-Earth-Orbit (LEO) satellite networks, where traffic management is performed with assistance of newly developed artificial intelligence methods.

This deliverable is a result of the work done in the context of WP3 Subtask T3.2 – 5G and DVB-S2X protocol implementation using HW/SW co-design. Deliverable D3.2 presents developed FPGA hardware accelerators for both physical layers.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

ASIC	**Application Specific Integrated Circuit**
BCH	**Bose–Chaudhuri–Hocquenghem**
C2V	**check to variable node messages**
CC	**Control Channel**
DSC	**Data Stream Channel**
DMA	**Direct Memory Access**
FEC	**Forward Error Correction**
FPGA	**Field Programmable Gate Array**
GUI	**Graphical User Interface**
HUT	**Hybrid User Terminal**
HW	**Hardware**
LDPC	**Low Density Parity Check**
LFSR	**Linear Feedback Shift Register**
LLR	**log-likelihood ratio**
OS	**Operating System**
QAM	**Quadrature Amplitude Modulation**
RTL	**Register Transfer Level**
SDR	**Software defined radio**
SoC	**System on Chip**
SW	**Software**
TCP	**Transmission Control Protocol**
UDP	**User Datagram Protocol**
UART	**Universal asynchronous receiver-transmitter**
V2C	**variable to check node messages**
WP	**Work Package**

SECTION 1 - INTRODUCTION

This deliverable is a result of the work done in the context of WP3 Subtask T3.2 – 5G and DVB-S2X protocol implementation using HW/SW co-design. Deliverable D3.2 presents developed FPGA hardware accelerators for both physical layers.

This deliverable is structured as follows: In Section 2 overall system architecture is presented. It addresses the main system components, hardware architecture and communication mechanism between different devices in the system. In Section 3 hardware accelerators are described. Most critical parts of processing such as data encoding and decoding are accelerated by implementation of dedicated FPGA-based hardware units. Each accelerator chain and components within each chain are described in details. Software architecture and developed applications that enable data creation, processing and checking are described in Section 4.

SECTION 2 - SYSTEM ARCHITECTURE

The AMD-Xilinx RF-SoC development board ZCU111 (later called just RF-SoC) is about to serve as an accelerator card in a SW-defined radio system whose features are implemented on a general purpose PC with time-critical tasks implemented on an RF-SoC. The communication with the PC is obtained via ethernet.

The RF-SoC has a processing system with ARM Cortex-A9 processors that can run Linux and work as an independent computer. Such a system is used for an interface between the outer world (e.g. HUT's PC) and the RF-SoC, and the acceleration functionalities, which are implemented in programmable logic with a significant amount of FPGA resources.

The most complex part of the development in accelerator implementation is in RTL design of individual IP cores (acceleration functionalities). In the previous work it has been analyzed that the forward error correction tasks are the most critical in terms of complexity. Therefore, the FEC IP cores are implemented as mandatory. However, since other different functionalities are potentially needed, and since different standards require different algorithms and IP cores, the accelerator design is developed to share the same top-level architecture regardless of the final acceleration functionalities. This way, the flexibility is achieved.

The block diagram of the accelerator's top-level architecture is shown in Figure 1. It is intended that the ARM CPU (with Linux running) receives and sends the data from and to the PC using the ethernet interface. The data is unpacked from ethernet packets in SW and sent to the FPGA logic using the DMA controller. The hardware component should unpack the sent packet and identify control data, separate it from the user data, and pass it to the acceleration functionalities (Figure 2). Acceleration functionalities are developed as a chain of multiple components, which in parallel has a control stream derived from the control data received in the control header. The control stream is controlled by controlling components, which parse the control header and synchronize the appropriate control signals for individual acceleration IP cores. The general structure is shown in Figure 3. The accelerator chain returns processed data and status signals. The status is packed and padded to the user data at the end of the data block (Figure 2). Such a block is then sent to the DMA controller which passes the data to the software memory space. Now, the software can pack the data into the ethernet packets and send it to the PC.

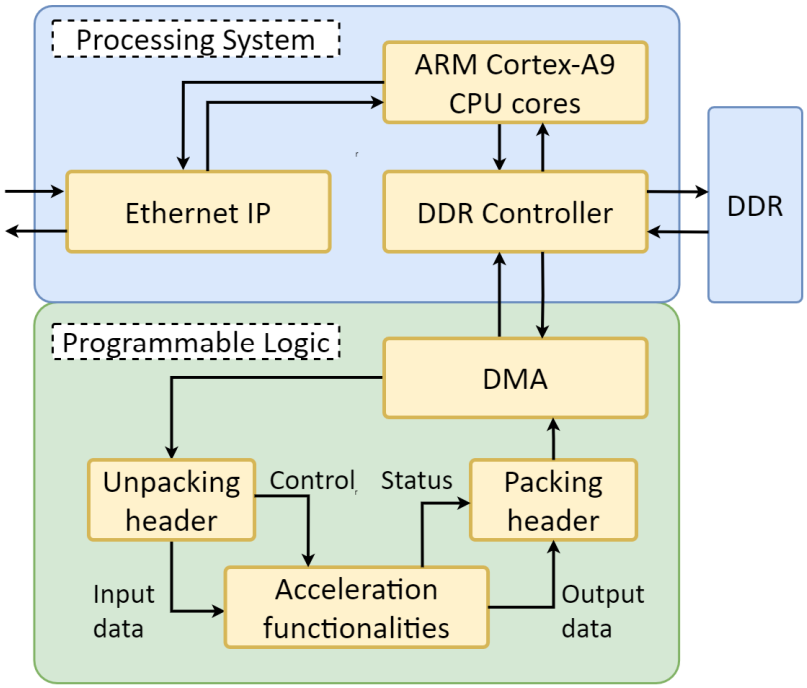


Figure 1 - General accelerator architecture

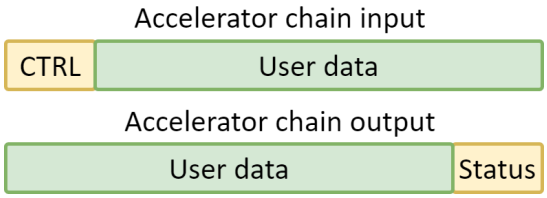


Figure 2 - A general format of the HW acceleration chain data

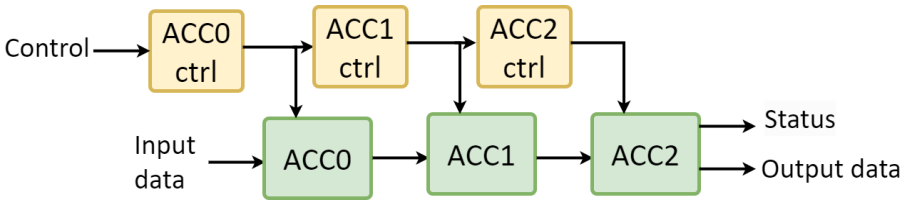


Figure 3 - A general accelerator chain with controlling components synchronized with the user data

Single RF-SoC device can have multiple acceleration chains. In this project, there are four acceleration chains identified:

1) 5G NR TX accelerator chain - acceleration IP cores used for 5G NR transmitter
2) 5G NR RX accelerator chain - acceleration IP cores used for 5G NR receiver
3) DVB-S2X TX accelerator chain - acceleration IP cores used for DVB-S2X transmitter
4) DVB-S2X RX accelerator chain - acceleration IP cores used for DVB-S2X receiver

It is up to the system integrator to decide which functionalities will be implemented on a single board. There are several options that can be configured. If two boards are used, the functionalities can be separated into two independent parts. The most reasonable feature sets for two independent cards can be divided by: 1) standard (5G NR and DVB-S2X) or 2) transmission side (transmitter and receiver). In the first scenario, a single board would have both the parts of transmitter and the receiver for 5G NR on a single card, and both the parts of transmitter and the receiver for DVB-S2X on another card. In the second scenario, one card can be used only for transmitter functionalities but for both standards, while the second card can be used only for receiver functionalities. A third option where all functionalities are supported on a single card is also a legitimate option, which can be useful if the second card must be used for channel modeling (if needed). However, in this case the FPGA resources might become a bottleneck.

Communication between different devices is depicted in Figure 4. Configuration of communication parameters, generation of different types of data and final checking is controlled trough the custom developed GUI console application. Accelerators chain configuration and control is done from custom user application that is built under Linux OS which is run on the ARM CPU inside processing system.

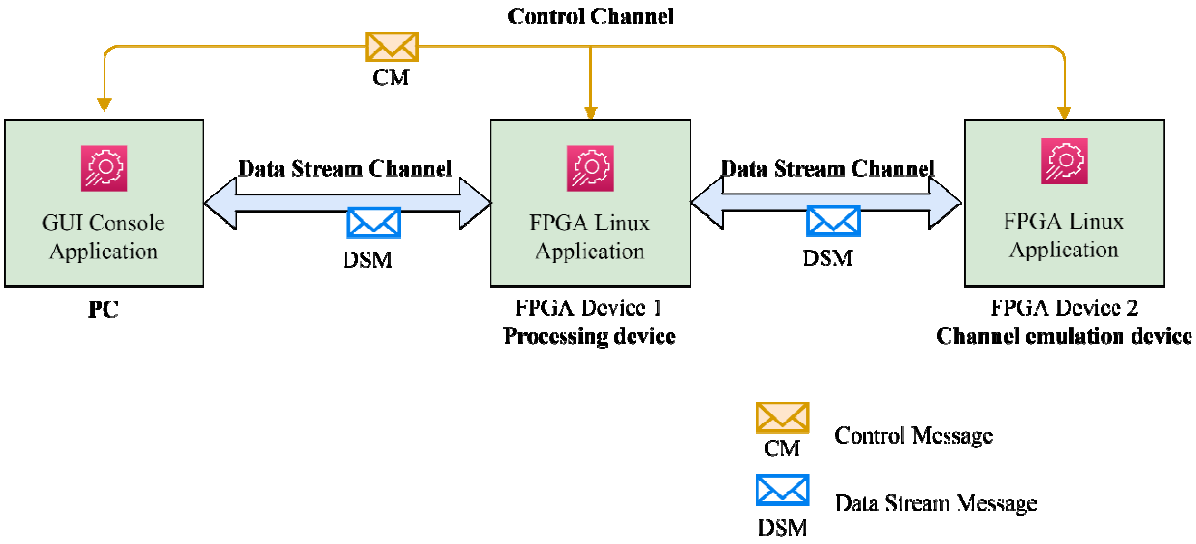


Figure 4 - Communication channels between different devices in the system

There are two types of channels that can be used for communication between devices:

- Control Channel (CC)
- Data Stream Channel (DSC)

The primary function of the Control channel is to facilitate the configuration of FPGA devices through the GUI Console application. Control messages are transmitted through this channel,

which is established using the TCP protocol. These messages are implemented as ASCII string messages. The existing software version, encompassing the hi-STAR Console Application and FPGA Linux Application, supports a limited number of commands. Additional control commands will be incorporated in subsequent milestones.
Data Stream Channel is used to exchange user data over UDP between devices. Data exchanged over Data Stream Channel is encapsulated inside Data Stream Message illustrated in Figure 5.

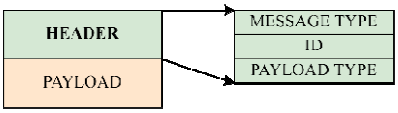


Figure 5 - Data Stream Message structure

A data stream message contains a message header and a message payload. The message header is composed of three fields:

- **Message type**

  There are two message types: Request and Response

- **ID**

  Each message has unique ID that enables tracking of messages
  When a request is generated, a unique ID is assigned to that message which stay until response, that corresponds to that request, is completely processed on GUI Console application.

- **Payload Type**

  Identify the type of payload which is used by Data processing logic to forward receive packets to corresponding processing algorithms: DVBS2X or 5G.

The Data Stream Channel enables the exchange of two message types: Request and Response. Data stream is initiated by sending Request message to the appropriate FPGA accelerator card, each Data Stream Message is encoded and forwarded to the channel emulator. After channel emulation data is sent to accelerator FPGA card for decoding. The message type of decoded data is changed to Response and data is forwarded to the GUI Console application. This is illustrated in Figure 6.

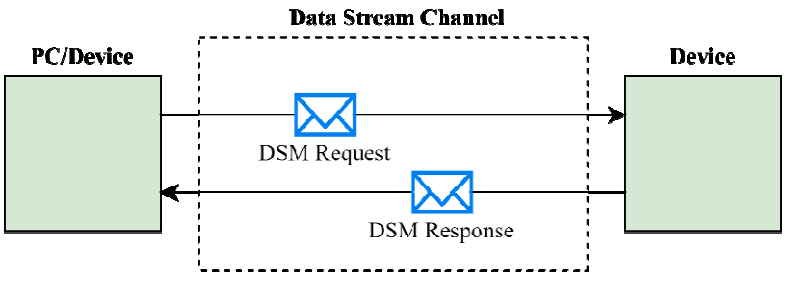


Figure 6 - Data Stream Channel

SECTION 3 - ACCELERATION CHAINS

This section describes the main features of each of the developed acceleration chains. The most complex chains are briefly described and their architectures presented.

SECTION 3.1 - 5G NR TRANSMITTING CHAIN

IP cores developed for 5G NR transmitter include:

- LDPC encoder (including puncturing and filler bit insertion) soft IP core
- Bit selection and interleaving IP core
- QAM mapper

All IP cores follow standardized AXI4-Stream interface for user data and parallel ports synchronized with the data stream for control and status. The detailed block diagram of all functionalities is shown in Figure 7. The control contains the information about modulation order, exact LDPC code identifier, and lengths of the information part of the code word and the entire code word necessary for proper bit selection.

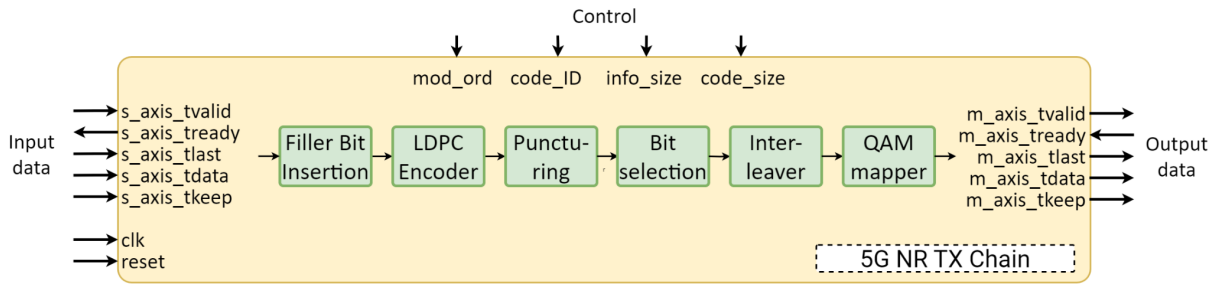


Figure 7 - Block diagram of 5G NR TX Accelerator Chain

The most challenging part of the 5G NR TX acceleration chain is the LDPC encoder. It is designed using the algorithm described in [1]. The algorithm is based on solving the linear equations system in GF(2) for syndrome calculation, i.e. by using the parity check matrix for encoding. The architecture of the LDPC encoder core is shown in Figure 8. It takes information bits from the input buffer. The bits are packed into groups of Z bits, where Z is the lifting factor of the code, defined by the code_ID. The information bits are firstly circularly shifted and cumulatively added to calculate GF(2) sums of the first four row groups (layers) of the parity check matrix. Those values are stored in four registers denoted with lambda. These intermediate values are then used to calculate the first four groups of parity bits. These bits are passed to the output, but are also used for further calculation of additional parity bits, together with some informational bits. The rationale for this approach can be further investigated in [1].

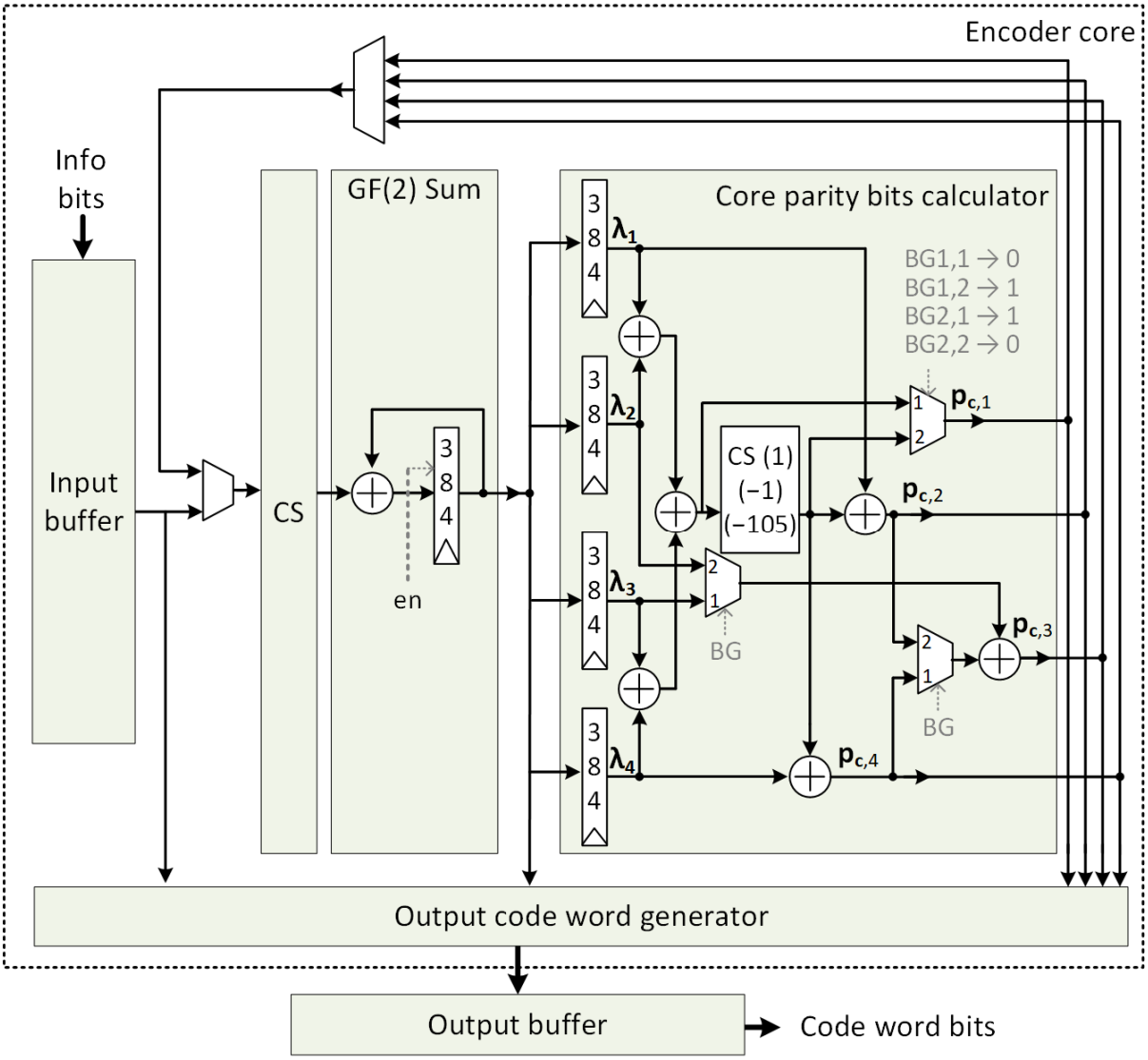


Figure 8 - 5G NR LDPC Encoder Architecture

Filler bit insertion is handled by the input module of the LDPC encoder core. It is used to pack the input stream into the format suitable for the LDPC encoder core. During this process, filler bits are added if needed. Puncturing is done by the LDPC encoder core by simply not writing the first two groups of information bits, as defined by the standard [2].

Bit-selection and interleaving are combined and performed together. The module takes a stream of encoded bits and packs them into a circular buffer. Bits are selected as defined by the standard and written to the buffer memory column-wise to be read row-wise. This way, the interleaving is performed. The interleaving is programmable so that all modulation orders supported by standard can be used.

QAM mapper is a simple module that unpacks a stream of bits into an array of groups of multi-bit symbols and maps them to QAM symbols using the simple lookup table method.

SECTION 3.2 - 5G NR RECEIVING CHAIN

IP cores developed for 5G NR receiver include:

- QAM demapper
- Rate-matching and deinterleaving IP core
- LDPC decoder (including de-puncturing and filler bit removal) soft IP core or wrapper IP core for Xilinx's SD-FEC hard IP available on RF-SoC

All IP cores follow standardized AXI4-Stream interface for user data and parallel ports synchronized with the data stream for control and status. The detailed block diagram of all functionalities is shown in Figure 9. The control contains the information about modulation order, exact LDPC code identifier, number of iterations that are available for iterative LDPC decoding, and length of the information part of the code word for proper filler bit removal.

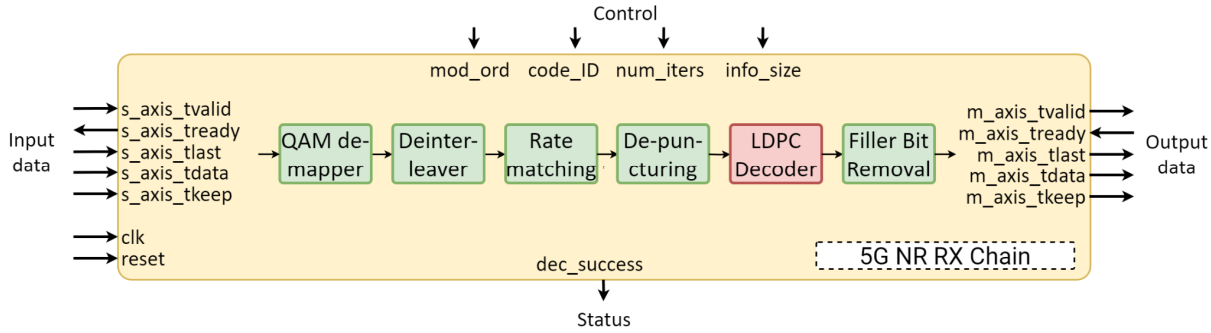


Figure 9 - Block diagram of 5G NR RX Accelerator Chain

QAM demapper is implemented by using simple decision threshold algorithm described in [3].

De-interleaving and rate-matching are functionalities inverse to the bit-selection and interleaving from the TX chain. Therefore, they are implemented similarly. The main difference is in the fact that the data is not a stream of bits, but a stream of LLRs (multi-bit probabilities that indicate the soft information whether the received bits are 0 or 1).

LDPC decoder is implemented in two ways: 1) fully in FPGA logic as an independently developed soft IP core, or 2) by wrapping the integrated SD-FEC IP core, available in ASIC on an RF-SoC device.

Soft IP core follows the hybrid decoding algorithm described in [4].

De-interleaving and rate-matching are functionalities inverse to the bit-selection and interleaving from the TX chain. Therefore, they are implemented similarly. The main difference is in the fact that the data is not a stream of bits, but a stream of LLRs (multi-bit probabilities that indicate the soft information whether the received bits are 0 or 1).

LDPC decoder is implemented in two ways: 1) fully in FPGA logic as an independently developed soft IP core, or 2) by wrapping the integrated SD-FEC IP core, available in ASIC on an RF-SoC device.

Soft IP core follows the hybrid decoding algorithm described in [4]. The hybrid decoding is a method based on the conventional layered decoding, but optimized for hardware implementations. Its advantage is in the possibility for removal of all stall cycles that happen because of the pipeline hazards. The decoder architecture is shown in Figure 7 and is briefly explained in the following text.

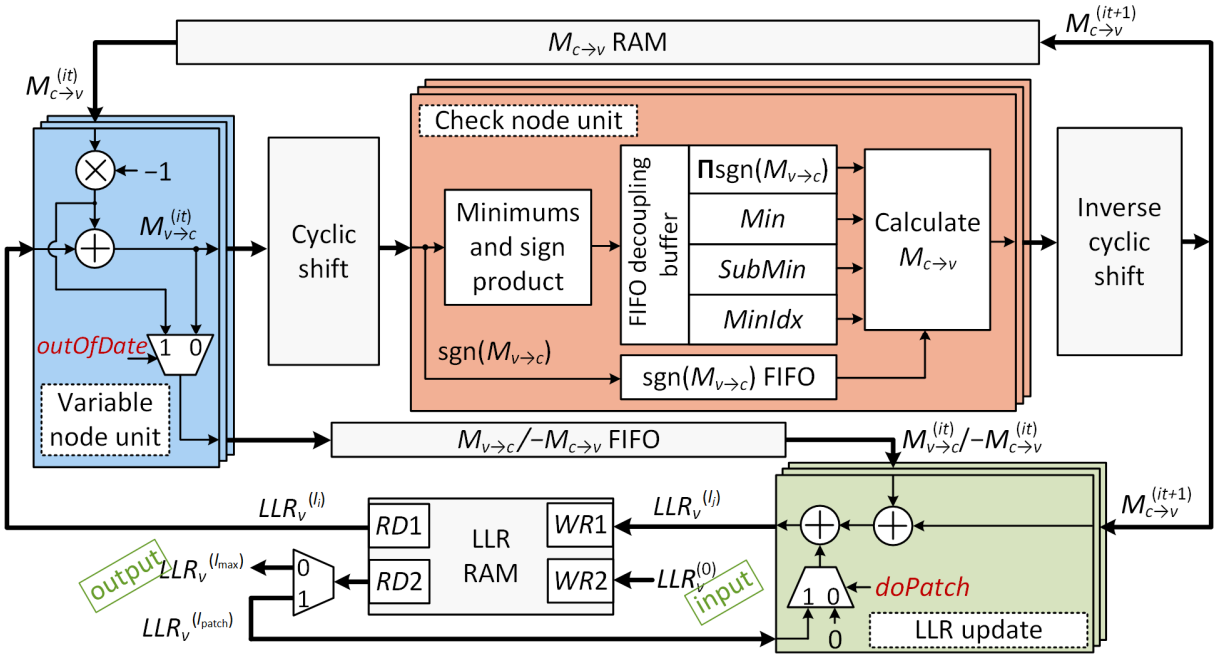


Figure 10 - The architecture of LDPC decoder IP core

The decoding is an iterative process where codeword LLRs are continuously updated based on the message passing algorithm. The algorithms calculate variable-to-check messages (messages that variable nodes in Tanner's graph send to check nodes) based on the current LLR values. Variable-to-check (V2C) messages are calculated in parallel for multiple variable nodes in so-called variable node units. Number of messages is equal to the number of check nodes in the parity check matrix layers. V2C messages are then circularly shifted in the flexible module for cyclic shift and passed to check node units. Cyclic shift is necessary to establish adequate connections between variable and check nodes in Tanner's graph of an LDPC code, since 5G codes have quasi-cyclic structure. Check node units calculate check-to-variable (C2V) messages based on the received V2C messages from a single layer. C2V messages are calculated based on the offset min-sum algorithm. Those messages are inversely shifted and passed to a set of parallel modules called LLR update units. LLR update units calculate new LLRs in conventional layered manner by summation of the V2C message and newly calculated C2V message. However, it is obvious that LLR RAM memory is shared for calculation of new V2C messages

(read) and for update of variable node state, i.e. writing the new LLR values (write). In such a pipelined architecture it is possible that variable node units need the new LLRs calculated in previous layers to calculate new V2C messages. In those cases, to avoid stall cycles, the hybrid algorithm reads the old LLR values, but in LLR update units, the update is done differently. The FIFO that is used to pass V2C messages to the LLR update units is now used to pass the old C2V messages (*outOfDate* signal in Figure 10), which are later used for the current layer's contribution calculation. This contribution is added to the LLR values stored in the memory (*doPatch* signal in Figure 10 is active).

The described architecture can be used for decoding any quasi cyclic (QC) LDPC code. However, it can be reused, with some additional changes to decode structured IRA codes such as those used in DVB-S2 and DVB-S2X standards.

De-puncturing and filler-bit removal are done inside the LDPC decoder soft IP core, when it is used for acceleration. However, to save FPGA resources and accelerator's power consumption, the LDPC decoding for 5G codes can be done in the integrated Xilinx SD-FEC IP core, which is available in some RF-SoC devices. In this case, the SD-FEC IP core must be wrapped into a soft IP core module that does de-puncturing and filler bit removal outside of the hardened IP core, since these features are not supported in the ASIC core. Since it is still unclear how the integration will finally happen and which acceleration features will remain on a single RF-SoC device, the SD-FEC wrapping is also done. The IP core is integrated in the accelerator chain and is readily available for acceleration of the 5G receiver.

SECTION 3.3 - DVB-S2X TRANSMITTING CHAIN

IP cores developed for DVB-S2X transmitter include:

- BCH encoder
- LDPC encoder
- Interleaver (reused architecture from 5G)

All IP cores follow standardized AXI4-Stream interface for user data and parallel ports synchronized with the data stream for control and status. The detailed block diagram of all functionalities is shown in Figure 8. The control contains the information about modulation order and exact BCH+LDPC code identifier.

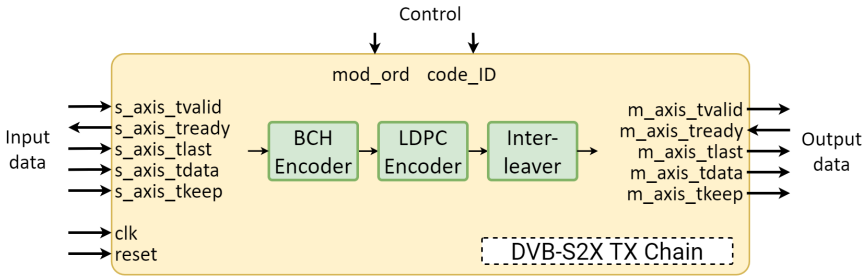


Figure 11 - Block diagram of DVB-S2X TX Accelerator Chain

The DVB-S2X standard introduces structured irregular repeat accumulate (IRA) LDPC codes as the main error protecting codes for cleaning the majority of errors that happen due to the imperfect transmission. However, the additional BCH code, as an outer code, is used to clear the remaining residual errors that can happen after the LDPC decoding.

The BCH encoding is a very simple operation performed by a simple linear feedback shift register (LFSR) architecture. The only challenge lies in the fact that for high speed applications, more than one bit per clock cycle must be calculated. However, the LFSR can be parallelized without complex procedures and algorithmic changes.

The LDPC encoding of the IRA codes is different from the 5G codes encoding. For this application, a method from [5] is used. Similarly to 5G, it is still based on solving the system of GF(2) equations and on using the parity check matrix for encoding. However, the equations are completely different and induce different hardware architecture.

The parity check matrix of structured IRA code such as those in DVB-S2X can be permuted in a way to look like a QC code. The example is shown in Figure 12 and in Figure 13. Firstly, the rows are permuted to get the QC structure in the left hand side of the matrix (the part of the matrix related to the information bits). This change does not influence the order of bits in the codeword. The right hand side of the matrix can be permuted column-wise to get QC structure. However, since columns are permuted, the parity bits must be permuted after the encoding to get the appropriate order of bits in the codeword.

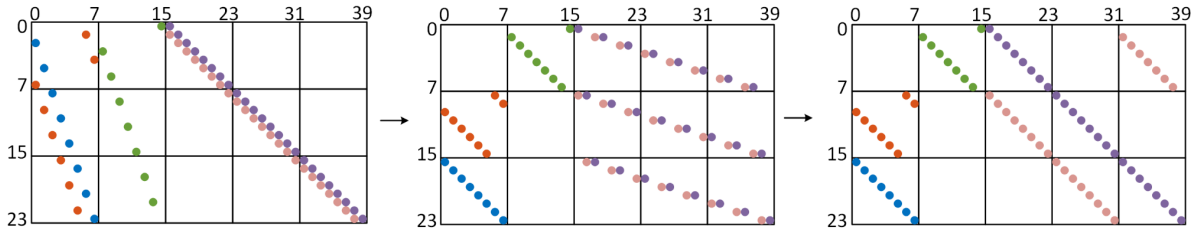


Figure 12 - Permutation of IRA code PCM to get QC-like PCM

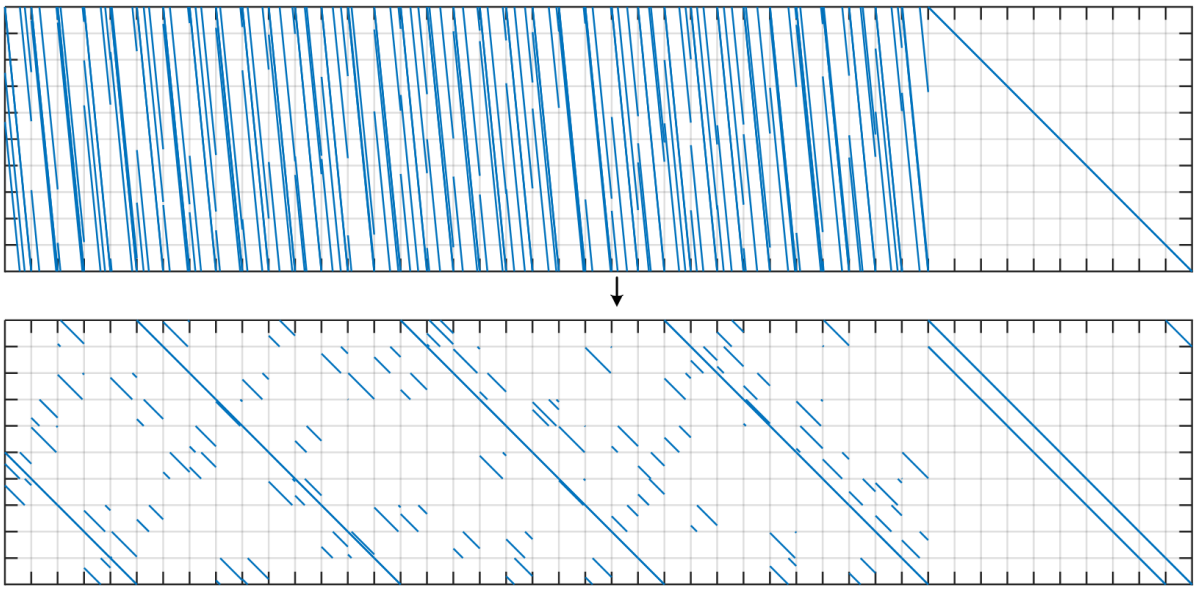


Figure 13 - The example of DVB-S2X code PCM permutation to get QC-like PCM

The presented structure is more intuitive and enables deriving the encoding algorithm. The architecture of the IRA LDPC encoder is shown in Figure 14. The first part of the algorithm consists of calculating intermediate sums of cyclically shifted information bits for each layer. If all parity check equations are summed, it is easy to conclude that by summing all of the intermediate results, the result is a cumulative sum of the last parity bits group and the last parity bits group shifted by one position. Therefore, the GF(2) sum post-processing module is used to obtain the last parity bits group from the total sum of the intermediate results. Now, all other parity bits are iteratively calculated in groups of 360 by adding already calculated parity bits to the intermediate results previously stored in RAM.

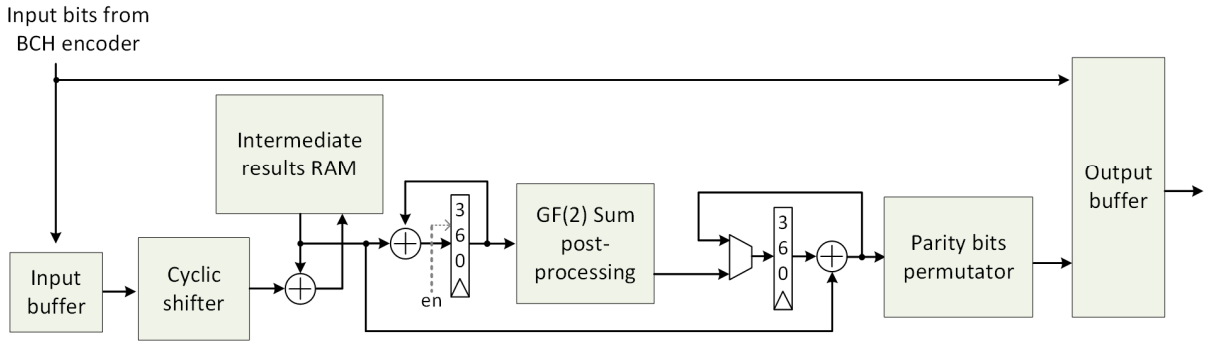


Figure 14 - LDPC encoder architecture for DVB-S2X LDPC codes

Interleaver architecture is reused from 5G.

SECTION 3.4 - DVB-S2X RECEIVING CHAIN

IP cores developed for DVB-S2X receiver include:

- Deinterleaver (reused architecture from 5G)
- LDPC decoder
- BCH decoder

All IP cores follow standardized AXI4-Stream interface for user data and parallel ports synchronized with the data stream for control and status. The detailed block diagram of all functionalities is shown in Figure 15. The control contains the information about modulation order, exact LDPC code identifier, and number of iterations that are available for iterative LDPC decoding.

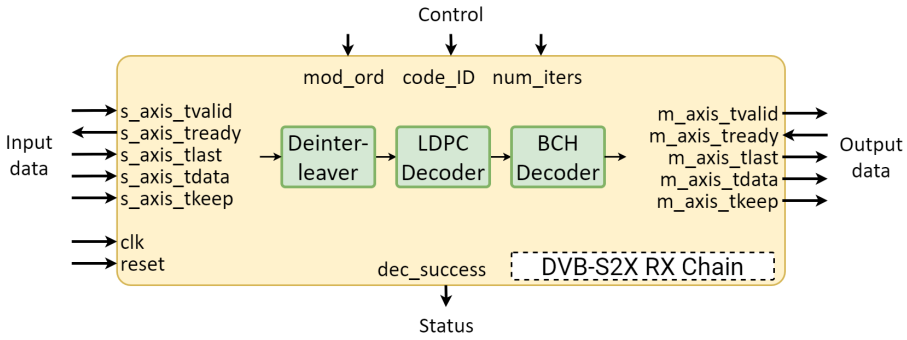


Figure 15 - Block diagram of DVB-S2X RX Accelerator Chain

Deinterleaver shares similar architecture as for 5G.

LDPC decoder for DVB-S2X codes fundamentally shares the architecture with the 5G soft IP core for LDPC decoding. However, as already seen in the encoder case, the DVB-S2X parity check matrix must be permuted to be used in QC LDPC decoder. Consequently, all the LLRs at the decoder input must be permuted to correspond to the permuted PCM. Therefore, the DVB-S2X LDPC decoder consists of two major parts: the input permutator and the LDPC decoding core similar to the 5G's decoder.

The input permutator works as a programmable interleaver that can change the interleaving pattern in runtime. Interleaving patterns are different for various codes defined in the standard. The decoder core works on parallelism of 360, which is the natural number of PCM rows in a single layer. Another key difference is in handling the upper-right entry in the parity check matrix. It can be seen that the upper-right square matrix is incomplete. It is an identity matrix shifted by 1 position to the left, but one entry is missing. Therefore, the connection between the last variable and the first check node must be removed in the hardware architecture. This is done by careful multiplexing in variable node units and LLR update units, to avoid impact of non-existing V2C message on the check node unit calculation, and to avoid the change in LLR memory for the last variable node.

BCH decoder is developed as a standard compliant decoder that fixes the residual errors that remain after LDPC decoding. The decoder uses conventional architecture, shown in Figure 16, which takes the codeword bits and calculates the syndrome. The syndrome is used to solve a system of equations which will give the polynomial that determines error locations. The codeword bits are stored in FIFO during the syndrome calculation to wait for error locating completion. After the errors are found, the codeword bits are corrected using typical XOR operation. However, the BCH codes defined in the standard can correct up to 12 errors. If there are more residual errors left after the LDPC decoding it cannot be guaranteed that the information bits are correct. This is the reason why the decoder has an additional syndrome checker. If the final syndrome is equal to zero, the BCH decoder will return the "decoding success" as a status.

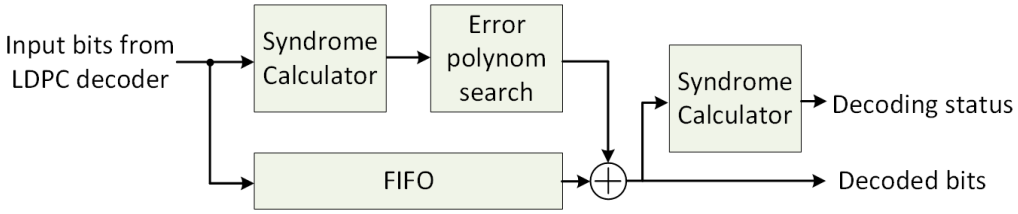


Figure 16 - Top level architecture of BCH decoder

SECTION 4 - SOFTWARE INFRASTRUCTURE

Two applications are developed to enable establishing of previously described data transfer path. These two applications are:

- hi-STAR GUI console applications created within Qt framework.
- Linux based application for FPGA based MPSoC.

4.1. LINUX BASED APPLICATION FOR FPGA

FPGA boards host System-on-Chip (SoC) instances based on application processors capable of running the Linux operating system. This setup facilitates convenient control of FPGA functionalities through a Linux application. In this project, an application has been developed, and its overall functionalities are depicted in the provided Figure 17.

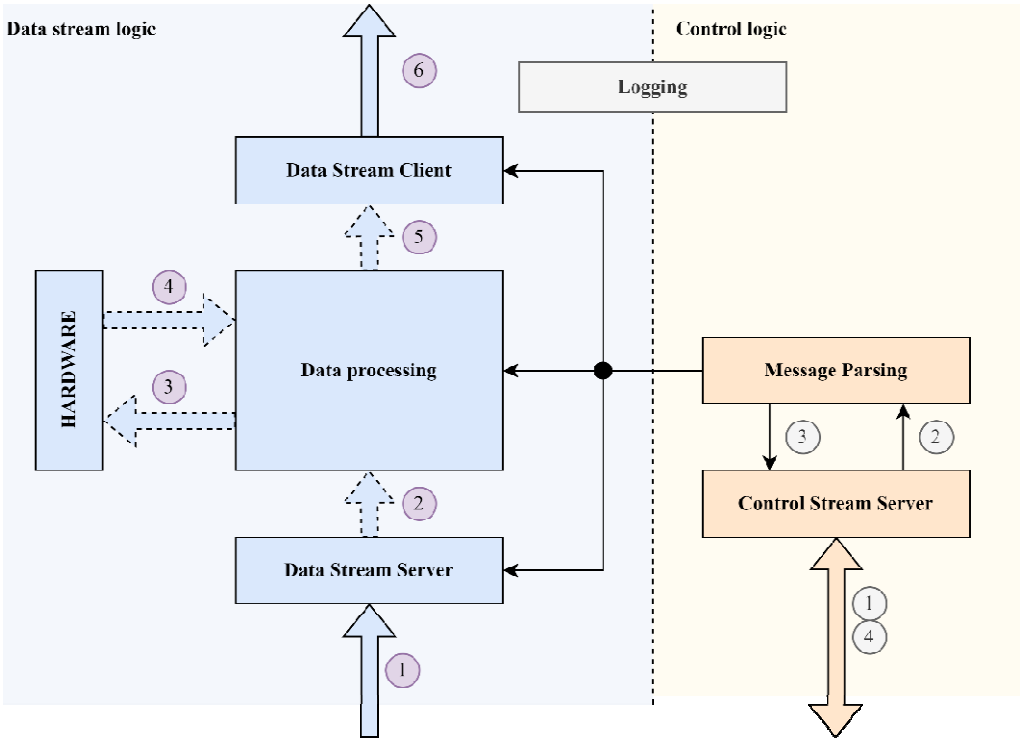


Figure 17 - Linux application architecture for FPGA boards

Complete application can be separated on two logical units:

- Data stream logic
- Control logic

Data stream logic supervises receiving, processing, and forwarding messages to previously configured destination server. Data, encapsulated within DSM format, are received on devices thought Data Stream Server (1). After data is successfully received, they are transferred to further processing inside Data Processing block (2). Inside this block is implemented logic which accelerates data processing by utilizing specific hardware blocks. Data is transferred to hardware (3) and from hardware (4) by utilizing DMA blocks. After data is processed, they are sent to the

previously configured destination server (5). This operation is under control of Data Stream Client logic block (6).

Control logic manages receiving, processing, and execution of control messages. Control Stream Server receives control messages on specific port (1) and forward them to the message parsing logic (2). This logic analyzes message content, extract command arguments from, execute commands by interacting with corresponding logic blocks and prepare response based on execution command status. Prepared response is returned to control stream server that return response to the client that send command (4). Based on response received on client side, it is possible to determine command execution status: successful or failed.

While executing the software, gaining insights is very useful feature. To facilitate this, logging logic that captures messages from all logic blocks, assigns them a timestamp and message type, and then transmits these logged messages over UART to the PC serial terminal is established. A portion of the logging messages is depicted in the Figure 18:

```
[1:8:27](Data Stream Client) - Sending data to IP: 192.168.1.110
[1:8:27](Data Stream Server) - Data received from source IP: 192.168.1.110
[1:8:27](Data Stream Server) - Packet counter: 28049408
[1:8:27](Data Stream Client) - Data successfully sent
[1:8:27](Data Stream Client) - Data received
[1:8:27](Data Stream Client) - Sending data to IP: 192.168.1.110
[1:8:27](Data Stream Server) - Data received from source IP: 192.168.1.110
[1:8:27](Data Stream Server) - Packet counter: 14483456
[1:8:27](Data Stream Client) - Data successfully sent
[1:8:27](Data Stream Client) - Data received
[1:8:27](Data Stream Client) - Sending data to IP: 192.168.1.110
[1:8:27](Data Stream Server) - Data received from source IP: 192.168.1.110
[1:8:27](Data Stream Server) - Packet counter: 37027840
```

Figure 18 - Portion of Linux application logging messages

4.2. HI-STAR GUI CONSOLE APPLICATION

Hi-STAR Console application represents interface to two FPGA boards that will be used within this project. This application enables easy configuration of those two FPGA boards and enables monitoring of logic execution.

After hi-STAR Console Application is launched, the initial GUI framework is opened. This initial GUI framework is presented in the following figure.

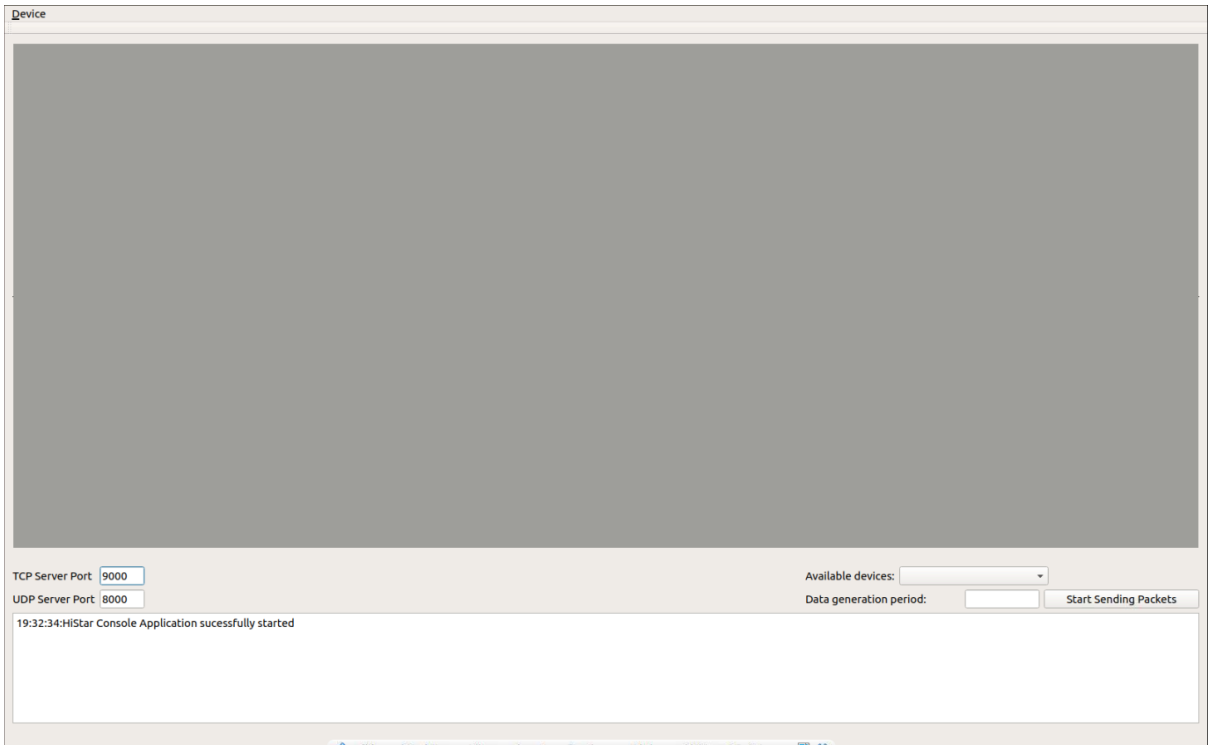


Figure 19 - GUI Console application start view

This framework consists of different logical GUI areas:

- Menu bar
- Connected devices overview.
- Server information
- Testing logic
- Logging interface

The menu bar features a "Device" menu, allowing users to establish connections with various FPGA boards where the FPGA Linux-based application has been initiated. Device options and status messages from the devices are displayed in dedicated sub-windows, instantiated within the connected devices overview logic unit.

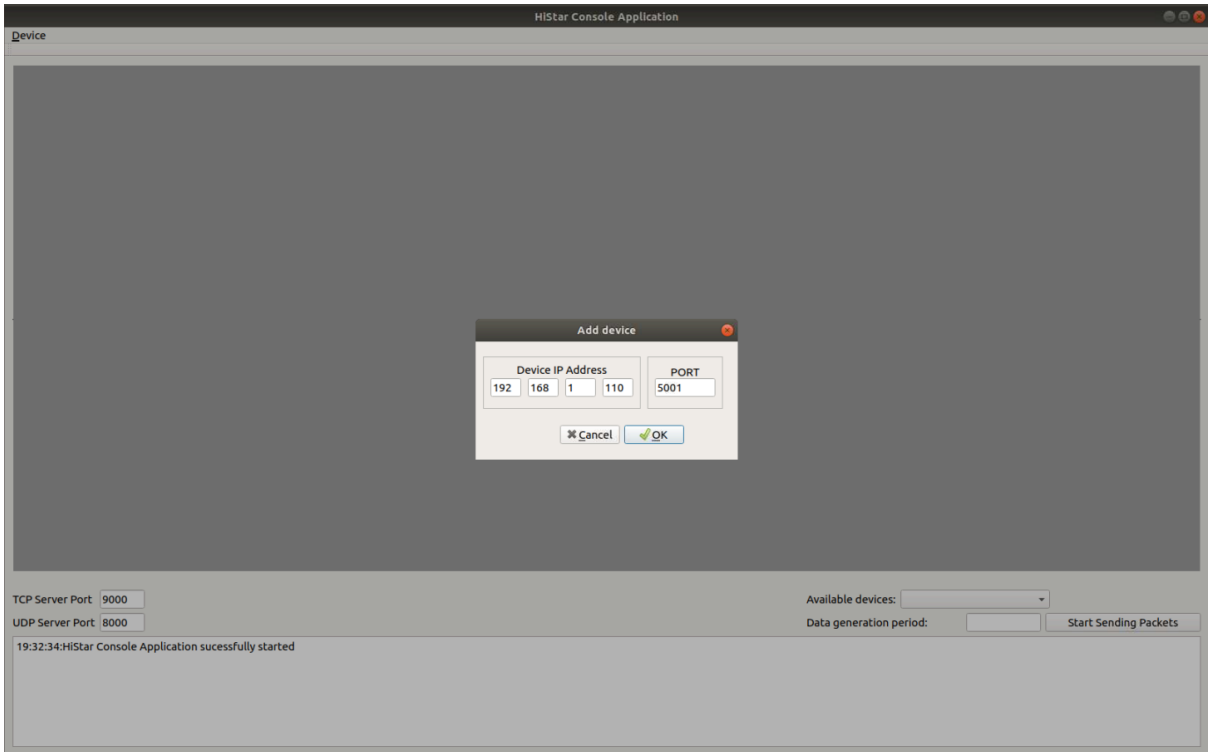


Figure 20 - Add device sub-window

Upon initiating the hi-STAR console application, UDP and TCP servers are launched in the background on specific port numbers. These port numbers are important for the end users to provide them to FPGA Linux application boards as data processing destination ports.

For performance testing, users can create different testing packages and send them through a previously established path with predefined generation periods. Within the Testing Logic section, users can select the desired device for communication and define the packet generation period in milliseconds.

It is always useful to support insight into application execution status and FPGA device execution status. This feature is integrated inside the logging interface area.

When the device's IP address is entered, and connection is established with Device, device sub windows is opened. This sub window layout is presented in Figure 21. In general case, multiple device sub-windows can be instantiated but, in our case, with system that includes two FPGA boards, usually two sub-windows are created. This is illustrated in Figure 22.

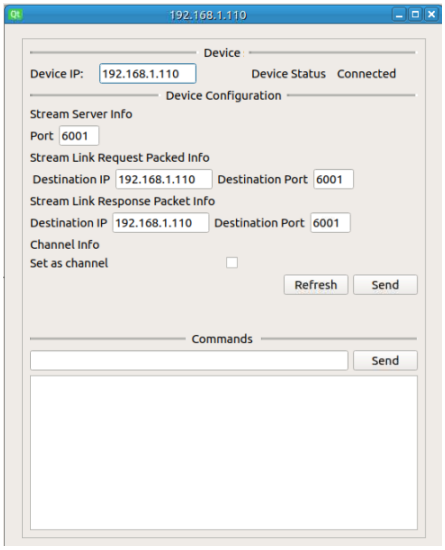


Figure 21 - Device sub-window

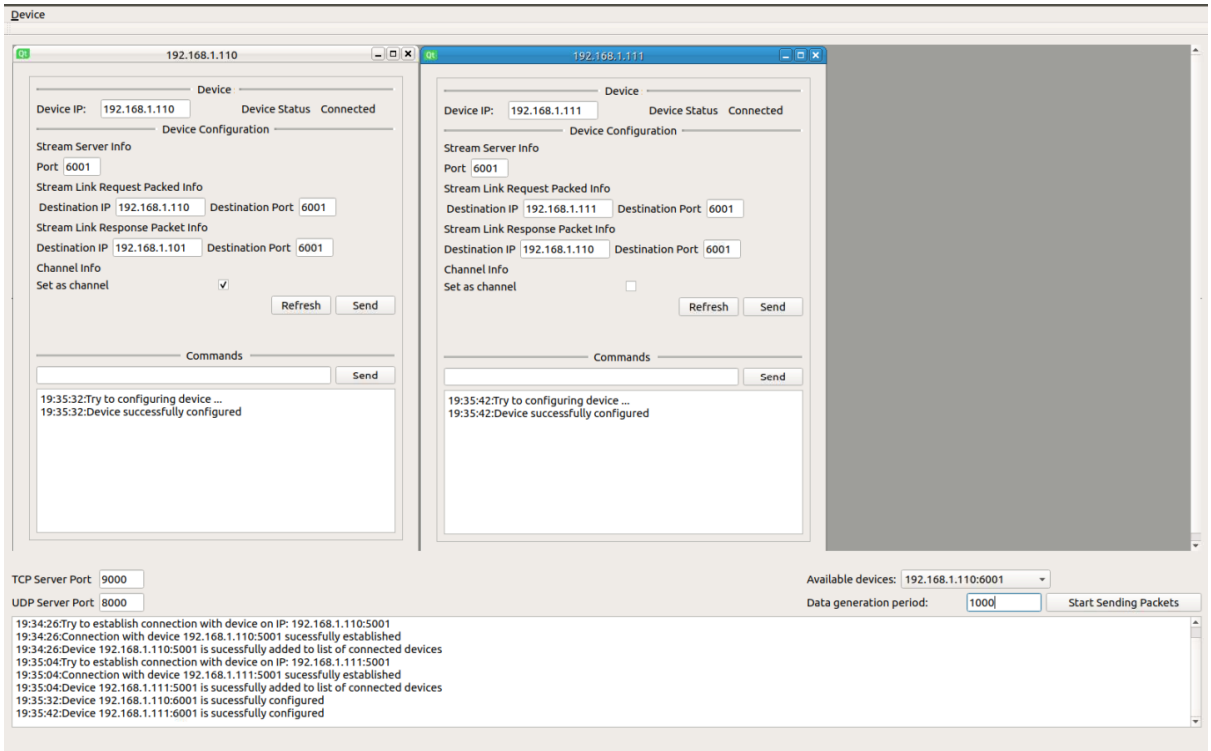


Figure 22 - GUI Console Application establish connection with two FPGA devices

CONCLUSIONS

This document D3.2 has described HW/SW co-designed system architecture for satellite and terrestrial communication that will be used for hybrid terminal demonstration. Developed accelerators are grouped into several communication chains 5G-NR transmitter and receiver as well as DVB-S2X transmitter and receiver. To efficiently control data flow two software applications have been developed. PC based GUI application enables control of overall data generation, transfer, processing and checking. Another application has been developed for FPGA based ARM CPU and it has been used for closer control of developed accelerator chains. Developed system make possible implementation of the final proof-of-concept that should enable usage of satellite and terrestrial communication standards in one unified hybrid communication solution.

REFERENCES

[1] Petrović, Vladimir L., Dragomir M. El Mezeni, and Andreja Radošević. 2021. "Flexible 5G New Radio LDPC Encoder Optimized for High Hardware Usage Efficiency" Electronics 10, no. 9: 1106

[2] 3rd Generation Partnership Project. Technical Specification Group Radio Access Network; NR; Multiplexing and Channel Coding (Release 16), 3GPP TS 38.212 V16.5.0 (2021-03); 3GPP: Valbonne, France, 2021.

[3] Ali, Imran, Uwe Wasenmüller and Norbert Wehn. “A high throughput architecture for a low complexity soft-output demapping algorithm.” Advances in Radio Science 13 (2015): 73-80.

[4] Petrović, Vladimir L., Milos M. Markovic, Dragomir M. El Mezeni, Lazar V. Saranovac and Andreja Radošević. “Flexible High Throughput QC-LDPC Decoder With Perfect Pipeline Conflicts Resolution and Efficient Hardware Utilization.” IEEE Transactions on Circuits and Systems I: Regular Papers 67 (2020): 5454-5467.

[5] Gomes, Marco Alexandre Cravo, Gabriel Falcão, Alexandre Sengo, Vitor Ferreira, Vítor Silva and Miguel Falcão. “High throughput encoder architecture for DVB-S2 LDPC-IRA codes.” 2007 Internatonal Conference on Microelectronics (2007): 271-274.