Employing GNU Radio for Robust Testing of the Novel DASH SoC

Drake Silbernagel DSILBERN@ASU.EDU
Prithvi Hemanth PHEMANTH@ASU.EDU
Alex Chiriyath ACHIRIYA@ASU.EDU
Saquib Siddiqui SASIDDI8@ASU.EDU
Wylie Standage-Beier WSTANDAG@ASU.EDU
Daniel Bliss DWBLISS@ASU.EDU

Center for Wireless Information Systems and Computational Architectures (WISCA), Arizona State University, Tempe, AZ, 85281, USA

Abstract

Through the patronage of the The Defense Advanced Research Projects Agency (DARPA) Domain-Specific System-on-Chip (DSSoC) program, we constructed the Domain-focused Advanced Software-reconfigurable Heterogeneous (DASH) system-on-chip (SoC), a coarse-scale heterogeneous SoC that breaks the trade-off in computational efficiency versus ease of reprogamability. The DASH SoC caters to the target domains of sophisticated RF processing for communications, radar, positioning, navigation, and timing (PNT), and spectral situational awareness. We also developed an FPGA-based DASH emulation platform to showcase the capabilities of the DASH SoC and aid in rapid testing and validation. In this paper, we use GNU Radio to rapidly test and validate the functionality of the DASH SoC's custom Forward error correction (FEC) accelerators with real data. We generate a realistic communications transmit chain with LDPC encoding in GNU Radio, and then feed the resulting output into the DASH SoC emulation framework to perform LDPC decoding via the FEC accelerator. We will show the results of the LDPC decoder, and showcase the functionality of the DASH SoC. Through GNU Radio, we are able to develop realistic communications transmit chains that scale up in complexity with great ease, enabling us to rapidly and robustly test the DASH SoC.

1. Introduction

Modern applications increasingly rely on multiple radio functionalities to provide more features and capabilities to users. These systems will require flexible computational architectures to perform multiple RF operations simultaneously and on demand. However, current implementations of such flexible systems have been limited traditionally by inflexible, rigid computational capabilities. Consequently, modern advancements in hardware will drive the design of future wireless systems with greater functionality and capabilities compared to current "stovepiped" systems.

Towards this end, as part of the DASH team, we built a framework to develop flexible, high-performance, low-power, domain-specific SoCs, while assuring non-expert programmability. The DASH SoC is an example SoC, developed via the aforementioned development framework, that targets the domain of software-defined RF systems: radios; radars; spectral awareness; positioning, navigation, and timing; and RF convergence. DASH allows RF system designers to escape the traditional power & development-cost limits to innovation (Bliss, 2020; Chiriyath et al., 2021). Our DASH framework and SoC breaks the trade-off between ease of reprogrammability, latency and power performance, as we highlight in Figure 1.

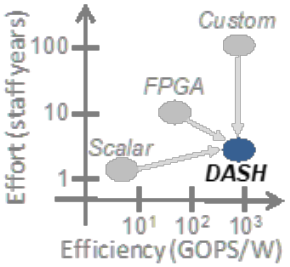

Figure 1. Trade-off between flexibility and performance.

In this paper, we venture down the path towards robust

Proceedings of the 1st GNU Radio Conference, Copyright 2016 by the author(s).

testing and validation of our DASH SoC with real data over-the-air (OTA) for the first time. More specifically, we employ GNU Radio to develop a realistic communications transmitter and receiver chains, while employing the DASH SoC to perform post processing in the form of soft demodulation and LDPC decoding. This preliminary system can be rapidly scaled up in complexity.

2. Background

2.1. DASH Development Framework

The DASH development framework is used to formally identify application domains and design DSSoCs that have high performance and can operate at lower power. This framework provides necessary capabilities to enable re-programmable and efficient processing of wide domain of embedded applications.

This framework has a ontological analysis (Uhrie et al., 2019), compilation and runtime workflow (Mack et al., 2021; 2020b;a), a flexible and robust scheduling scheme with tools to evaluate scheduling and resource allocation performance (Arda et al., 2019; Krishnakumar et al., 2020; Goksoy et al., 2021) along with high-performance network-on-chip. DASH SoC includes a heterogeneous cluster of accelerators which also include re-programmable systolic array accelerator referred to as domain adaptive processor (DAP)(Chen et al., 2022).

To enable rapid prototyping and extensive testing we have also developed an FPGA emulation of DASH SoC on Xilinx UltraScale+ VU19P evaluation board.

The emulation currently includes various specialized accelerators, ARM core clusters as well FEC and DAP processors.

2.2. CEDR

Compiler-integrated, extensible, DSSoC runtime (CEDR) (Mack et al., 2022b) is an open-source platform that allows users to remain in familiar programming environments while having the ability to automate and dynamically deploy applications to coarse scale heterogeneous compute systems which are comprised of cluster of general purpose processors and tailored accelerators.

CEDR integrates front-end compiler flow with a Linux-based runtime system. The front-end compilation workflow is used to convert C/C++/Python applications into CEDR compatible binaries. The runtime workflow is then used to parse, schedule, dispatch and execute those applications across a heterogeneous pool of resources that are offered by DSSoC.

The unique capability of CEDR is that it can be easily ported across any number of execution environments without requiring any major development changes hence remaining independent of any heuristic algorithm used for scheduling or hardware platform involved. Thus, the user can easily port any given scheduler to a new hardware platform or vice versa.

2.3. GR-CEDR

One of the key limitations of GNURadio has been the constraints on hardware resources that can be utilized and their selection at design time. By using GR-CEDR(Mack et al., 2022a) we can accelerate GNURadio blocks across a variety of available heterogeneous accelerators such as FPGA, GPU, custom accelerators such as FEC and re-programmable systolic array accelerators such DAP dynamically by utilizing intelligent runtime scheduler.

GR-CEDR uses the front-end compilation flow provided by CEDR to convert flow graphs to executable binaries which can be linked appropriately by minor tweaks to the building process. For C++ flow graphs this process is much simpler but for Python flow graphs, Cython3 is utilized as an intermediate step. Once the binaries have been compiled and linked, GNU Radio flow graph is launched from within CEDR as an application.

GR-CEDR has been demonstrated to work within GNU Radio Companion and as an Out-of-tree (OOT) module. Both of which enable the usage of API calls to the available pool of accelerators that CEDR provides.

3. LDPC Encoding and Decoding

In this section, the details of the low-density parity check (LDPC) encoding and decoding processes that are employed are discussed. The encoding process is done via GNURadio Companion and the DASH SoC. Specifically, the FEC Extended Encoder block is used to encode the message and the LDPC Encoder Definition (via Generator) block uses the stored generator matrix of the LDPC Generator Matrix block. The generator matrix is obtained from the DASH SoC after its own encoding procedure and fed into our subsequent experiments. The decoding, soft demodulation of symbols and LDPC decoding, process is done entirely on the DASH SoC FPGA emulation framework. The DASH SoC employs a custom FEC accelerator to perform the decoding with low latency and high energy efficiency (Yue et al., 2022).

This paper considers a specific class of LDPC codes, in which the parity check matrix is quasi-cyclic in nature. This class of codes is based on the 5GNR implementation of LDPC codes.

3.1. Encoding

The LDPC code is a systematic block code with a sparse parity check matrix. To achieve this type of encoding, various encoding schemes can be used such as Gaussian elimination, which is the one used. Gaussian elimination is agnostic to the structure of the parity check matrix, but it is also computationally intensive.

Given a shift matrix A, the parity check matrix H is formed by dividing the entire H matrix into chunks of ZxZ blocks, where Z is the lifting size. The matrix H is set to zero and based on the value of the shift matrix at $A[i][j]$, the corresponding ZxZ block in H is filled with a column-shifted identity matrix. The number of the column shifts is determined by the value at $A[i][j]$. If $A[i][j]$ is equal to -1 then the corresponding block is skipped i.e. the entire block is zero. This parity check matrix is then converted to an equivalent form using Gaussian Elimination. This equivalent form is used to construct the Generator matrix G which can be used to construct codewords.

3.1.1. Gaussian Elimination

In the Gaussian Elimination method, the generator matrix G is derived from the parity check matrix H. First, the parity check matrix is brought to the following form using Gauss-Jordan Elimination. Let n be the code length and k be the message length.

$$\mathbf{H}_{n-k \times n} = [\, \mathbf{P}_{n-k \times k} \;\; \mathbf{I}_{n-k \times n-k} \,] \tag{1}$$

Here, I is the identity matrix and P is the parity matrix. Next, the parity matrix is transposed, P', and the generator matrix, G, is constructed as described below.

$$\mathbf{G}_{k \times n} = [\, \mathbf{I}_{k \times k} \;\; \mathbf{P'}_{k \times n-k} \,] \tag{2}$$

Finally, the message bit string u is encoded by vector - matrix multiplication to get the codeword y. The bit strings are row vectors.

$$\mathbf{y}_{1 \times n} = \mathbf{u}_{1 \times k}\, \mathbf{G}_{k \times n} \tag{3}$$

This method is computationally intensive because the parity matrix P is dense. This implies an increase in the number of operations while multiplying.

3.2. Decoding

As evidenced by the inclusion within the 5G standard, a Quasi-Cyclic LDPC code is an increasingly favorable coding type. Moreover, it is advantageous for hardware implementation while still maintaining decoding performance to other standards. Given a codeword generated from the process described in the encoding section, a parity check matrix exists that satisfies $Hy^T = 0$ for a valid codeword. The parity check matrix, H, can be further described as a tanner graph with variable nodes (VN) describing the message bits and check nodes (CN) describing the parity bits. The FEC Accelerator of the DASH SoC employs a Min-Sum algorithm for LDPC decoding. This algorithm is proportional to others, like a sum-product approach, in a high SNR regime while being less complex (Anastasopoulos, 2001). As a result, the FEC Accelerator can maintain its unified structure with having the ability to switch between coding types without a large computational cost (Yue et al., 2022). The Min-Sum algorithm is an iterative process. In the min operation, η_{mn}, LLRs are calculated from check nodes, m, to variable nodes, n; conversely, the sum operation, λ_{nm}, computes LLRs from variable nodes, n, to check nodes, m.

$$\eta_{mn} = \prod_{k,\mathbf{H}(m,k)=1,k\neq m} sign(\lambda_{km}) \; \min_{k}(\lambda_{km}) \tag{4}$$

$$\lambda_{mn} = \sum_{k,\mathbf{H}(k,n)=1,k\neq n} \eta_{kn} + \eta_{0n} \tag{5}$$

The η_{0n} term within the sum operation is given by the LLR of the n^{th} bit received. The number of iterations is an adjustable parameter of the FEC Accelerator.

4. Experimental Results

In this section, an overview of the experimental setups are presented. The first experiment involves testing the FEC Accelerator under an AWGN channel utilizing GNU Radio Companion and validating it against internal unit tests. The other experiment is more involved, testing Over-the-Air data on the emulated DASH SoC. For this experiment, GNU Radio Companion provides a robust transceiver chain utilizing only native GNU Radio blocks and the DASH SoC provides the data processing component. Finally, the results of each experiment are presented with accompanying analysis.

4.1. Validation Setup

A simple experimental setup is constructed with GNU Radio Companion which provides a transmitted signal through an AWGN Channel. The received noisy symbols are then processed on the DASH SoC.

The following description explains the GNU Radio Companion Flowgraph and relevant parameters. A Vector Source sends out a random sequence of 324 bits unpacked as a byte. These bits are then encoded via the FEC Extended Encoder block. This block uses the LDPC Encoder Definition (via Generator) with an LDPC Generator Matrix that was constructed prior on the DASH SoC and written to a compatible format - alist. The LDPC encoding is a

rate 1/2 code which results in a 648 byte, of 1 significant bit, codeword. This codeword is then BPSK modulated with a Chunks to Symbols block. Finally, these symbols are summed with an AWGN noise source to provide noisy complex symbols. These symbols are then written to file and subsequently read and processed on the DASH SoC. Specifically, approximate log-likelihood ratios (LLRs) are computed and recast to integers for BPSK Soft Demodulation, and then the FEC Decoder API is called. This API requires information on the dimensions of the shift matrix, the lifting size, the shift matrix itself, and the fixed-point LLRs. The FEC Accelerator performs the Min-Sum algorithm described in the LDPC Decoding section with a user-set max iteration of eight. By manipulating the Noise variance of the AWGN, and therefore Noise Power, we compute a small sample Monte-Carlo. Since our signal is of unity gain (0 dB), this allows to vary SNR and test the resulting bit error rate.

4.2. Validation Results

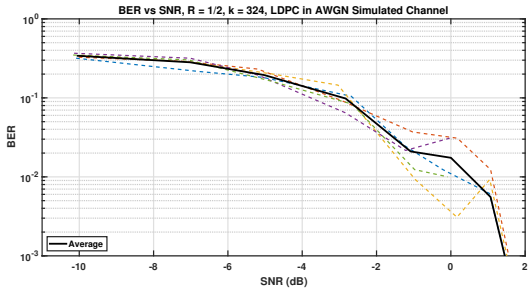

Figure 2. Bit-error rate vs. SNR for LDPC decoding via the DASH SoC's FEC accelerator. The bit-error rate clearly decreases as SNR increases, as expected. Moreover, the result follows the internal unit tests performed validating previous findings. This ultimately demonstrates the ability of GNU Radio Companion to act as a proper simulation test-bed for the DASH SoC.

In this section, we review the results of the simple test of the DASH SoC. This is done by plotting the bit-error rate of the DASH SoC FEC accelerator as a function of the Signal to Noise Ratio (SNR) in 2. As seen from Figure 2, we see the bit-error rate decreases as the SNR increases, which is as expected. When juxtaposed with internal unit testing of the same experiment, similar results are found. This concludes and shows the ability of GNU Radio Companion, and GNU Radio at large, to act as a user-friendly environment for simulated testing of the DASH SoC. It should be noted that these results are obtained over a Monte-Carlo simulation of five samples so randomness of the noise is not perfectly averaged. Additionally, the codeword and iterations are short thus not providing the lower BERs typically found at much higher SNRs. However, this unit test can be expanded to run over multiple Monte-Carlo simulations to alleviate these problems. Further integration between GNU Radio and our DASH SoC, via something akin to GR-CEDR (Mack et al., 2022a), will rapidly reduce the amount of effort required to do such testing.

4.3. Over-the-Air Setup

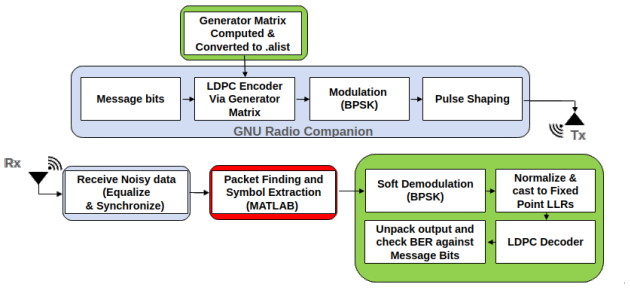

Figure 3. System Block Diagram presenting how each component is done.

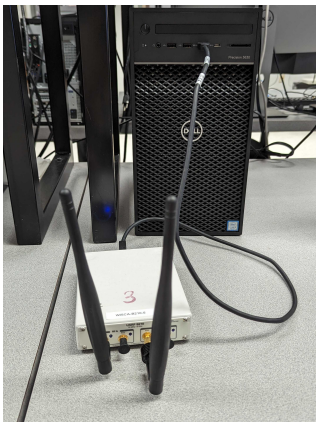

Figure 4. The physical setup of the USRP B210 Radio. Transmission from RF A (TX/RX) to reception of RF B (RX2) with 3 dBi omni-directional antennas. A 30 dB attentuator is placed on the receive antenna.

For the over-the-air component, we use GNU Radio, USRP B210 Software Defined Radio (SDR), MATLAB, and the DASH SoC to execute the experiment.

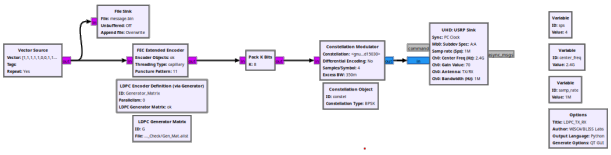

Figure 5. GNU Radio Companion Transmit Chain of OTA Component.

The entire transmit chain is implemented in GNU Radio Companion. A Vector Source block sends out a predefined message sequence on repeat. The message sequence is 324 bits in length. A 13-bit barker code is included at the beginning of the message to get a good correlation at the receiver. This sequence is then sent to the FEC Extended Encoder block. The encoding is done as mentioned in section 4.1. The encoded message is then packed to bytes by the Pack K Bits block. These packed bytes are fed to the Constellation Modulator block which modulates the encoded bytes to a BPSK signal. The block also performs pulse shaping at 4 samples per symbol. This pulse shaped BPSK signal is transmitted by the USRP B210 SDR at a gain of 70 dB.

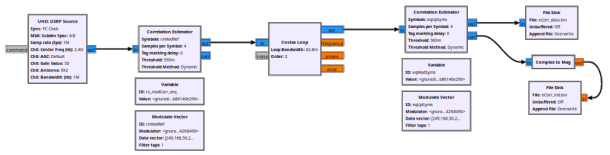

Figure 6. GNU Radio Companion Receive Chain of OTA Component.

The receiver chain contains these components - GNU Radio, MATLAB, and the DASH SoC. In the GNU Radio environment, the signal is received at a gain of 50 dB. The signal is also attenuated by a 30 dB attenuator attached to the receiver antenna. A Correlation Estimator block is used at this point to provide a phase estimate to the subsequent Costas Loop block. The signal is correlated over the entire modulated encoded sequence. The Costas Loop block uses the phase estimate to correct for the phase change thus equalizing the channel. A second Correlation Estimator block is used to locate the encoded sequence in the data stream. The outputs are written to a file for use in MATLAB. MATLAB uses the information about the correlation peaks to find the pulse shaped encoded message and extract the BPSK symbols from the signal. These symbols are written to rewritten to a binary file and supplied to the DASH SoC. The message sequence is computed by the FEC accelerator on the DASH SoC as explained in Section 4.1.

4.4. Over-the-Air Results

The following illustrates the results of the over-the-air exercise.

Figure 7 shows the received waveform, a pulse-shaped BPSK signal affected by the phase of the line of sight channel.

From the constellation plots before and after the Costas Loop, shown in Figures 8 and 9 respectively, we can see that the block uses the phase estimate from the Correlation Estimator to correct for the channel. Following equalization, the BPSK symbols remain on the real axis, as expected.

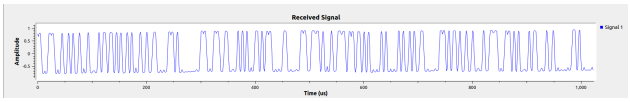

Figure 7. Received waveform of pulse-shaped data.

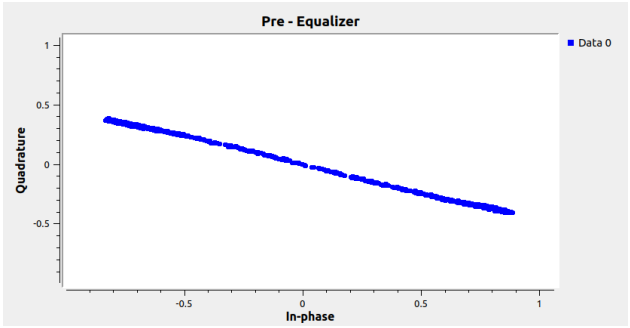

Figure 8. Received Constellation. Given the system parameters and setup, the constellation is a phase shifted BPSK.

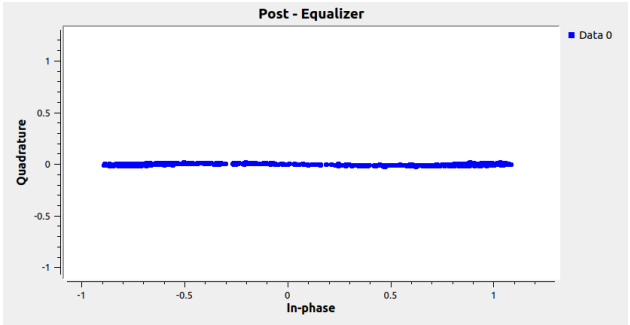

Figure 9. Constellation after Costas Loop providing a equalization. The initial correlation information tags are provided to the Costas Loop for a faster convergence using data.

After equalizing, the noisy symbols are further processed in MATLAB with windowing and down sampling to extract symbols of interest. This packet is then fed into the decoder of the DASH SoC's FEC Accelerator. The accelerator successfully computes the decoded bits with a bit error rate of zero. Given the system setup of a short link and input parameters of the USRP B210 TX/RX Gains, this result is anticipated.

5. Conclusion

In this paper, we showcase the functionality of the novel DASH SoC and process a test case of real over-the-air data. The simple test framework verifies the functional correctness of the DASH SoC's custom unified FEC accelerator. The over-the-air experiment demonstrates the practical capabilities of this accelerator and the DASH SoC in general. GNU Radio provides a platform of rapid testing for novice and knowledgeable users alike, either through simulation or assisting with over-the-air experiments. GNU Radio can also be used to easily scale up the complexity of either situation by quickly changing parameters such as the waveform or reception techniques. In combination with the prior work of GR-CEDR (Mack et al., 2022a), this joint work demonstrates the potential integration of GNU Radio with the DASH SoC to provide a platform of robust testing.

References

Anastasopoulos, A. A comparison between the sum-product and the min-sum iterative detection algorithms based on density evolution. In *GLOBECOM'01. IEEE Global Telecommunications Conference (Cat. No.01CH37270)*, volume 2, pp. 1021–1025 vol.2, 2001. doi: 10.1109/GLOCOM.2001.965572.

Arda, Samet E, NK, Anish, Goksoy, A Alper, Mack, Joshua, Kumbhare, Nirmal, Sartor, Anderson L, Akoglu, Ali, Marculescu, Radu, and Ogras, Umit Y. A Simulation Framework for Domain-Specific System-on-Chips: Work-in-Progress. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis Companion*, pp. 1–2, 2019.

Bliss, Daniel W. Development of next gen processors for rf systems. In *DARPA Electronics Resurgence Initiative Summit*, 2020.

Chen, Kuan-Yu, Yang, Chi-Sheng, Sun, Yu-Hsiu, Tseng, Chien-Wei, Fayazi, Morteza, He, Xin, Feng, Siying, Yue, Yufan, Mudge, Trevor, Dreslinski, Ronald, Kim, Hun-Seok, and Blaauw, David. A 507 gmacs/j 256-core domain adaptive systolic-array-processor for wireless communication and linear-algebra kernels in 12nm finfet. In *2022 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, pp. 202–203, 2022.

Chiriyath, Alex R., Herschfelt, Andrew, Srinivas, Sharanya, and Bliss, Daniel W. Technological advances to facilitate spectral convergence. In *2021 55th Asilomar Conference on Signals, Systems, and Computers*, pp. 623–628, 2021. doi: 10.1109/IEEECONF53345.2021.9723312.

Goksoy, A. Alper, Krishnakumar, Anish, Hassan, Md Sahil, Farcas, Allen J., Akoglu, Ali, Marculescu, Radu, and Ogras, Umit Y. Das: Dynamic adaptive scheduling for energy-efficient heterogeneous socs. *IEEE Embedded Systems Letters*, pp. 1–1, 2021.

Krishnakumar, Anish, Arda, Samet E., Goksoy, A. Alper, Mandal, Sumit K., Ogras, Umit Y., Sartor, Anderson L., and Marculescu, Radu. Runtime task scheduling using imitation learning for heterogeneous many-core systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):4064–4077, 2020.

Mack, Joshua, Kumbhare, Nirmal, and Akoglu, Ali. Automating Programming and Development of Heterogeneous SoCs with LLVM Tools. Talk at FOSDEM'20, Brussels, Feb 2020a.

Mack, Joshua, Kumbhare, Nirmal, NK, Anish, Ogras, Umit Y., and Akoglu, Ali. User-space emulation framework for domain-specific soc design. In *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 44–53, 2020b.

Mack, Joshua, , and Akoglu, Ali. Runtime Strategies and Task Scheduling of Software-Defined Radio on Heterogeneous Hardware. Talk at FOSDEM'21, Brussels, Feb 2021.

Mack, Joshua, Gener, Serhan, Akoglu, Ali, Holtom, Jacob, Chiriyath, Alex, Chakrabarti, Chaitali, Bliss, Daniel, Krishnakumar, Anish, Goksoy, Alper, and Ogras, Umit. Gnu radio and cedr: Runtime scheduling to heterogeneous accelerators. In *Proceedings of the GNU Radio Conference*, volume 7, 2022a.

Mack, Joshua, Hassan, Sahil, Kumbhare, Nirmal, Gonzalez, Miguel Castro, and Akoglu, Ali. Cedr - a compiler-integrated, extensible dssoc runtime. *ACM Trans. Embed. Comput. Syst.*, mar 2022b. ISSN 1539-9087. doi: 10.1145/3529257.

Uhrie, Richard, Bliss, Daniel W., Chakrabarti, Chaitali, Ogras, Umit Y., and Brunhaver, John. Machine understanding of domain computation for Domain-Specific System-on-Chips (DSSoC). In Suresh, Raja (ed.), *Open Architecture/Open Business Model Net-Centric Systems and Defense Transformation 2019*, volume 11015, pp. 180 – 187. International Society for Optics and Photonics, SPIE, 2019. doi: 10.1117/12.2519264. URL https://doi.org/10.1117/12.2519264.

Yue, Yufan, Ajayi, Tutu, Liu, Xueyang, Xing, Peiwen, Wang, Zihan, Blaauw, David, Dreslinski, Ronald, and Kim, Hun Seok. A unified forward error correction accelerator for multi-mode turbo, ldpc, and polar decoding.

In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, pp. 1–6, 2022.