IEEE Open Journal of
Circuits and Systems

Received 9 August 2020; revised 1 November 2020 and 11 December 2020; accepted 21 December 2020. Date of current version 28 January 2021.

*Digital Object Identifier 10.1109/OJCAS.2020.3047007*

# Channel Estimation for Advanced 5G/6G Use Cases on a Vector Digital Signal Processor

**STEFAN A. DAMJANCEVIC [1] (Graduate Student Member, IEEE), EMIL MATUS[1], DMITRY UTYANSKY[2], PIETER VAN DER WOLF[3], AND GERHARD P. FETTWEIS [1,4] (Fellow, IEEE)**

[1]Vodafone Chair Mobile Communication Systems, TU Dresden, 01187 Dresden, Germany

[2]Solutions Group, Synopsys Inc., 197022 St. Petersburg, Russia

[3]Solutions Group, Synopsys Inc., 5656 AE Eindhoven, The Netherlands

[4]Barkhausen Institut, TU Dresden, 01069 Dresden, Germany

This article was recommended by Associate Editor C. Studer.

CORRESPONDING AUTHOR: S. A. DAMJANCEVIC (e-mail: stefan.damjancevic@tu-dresden.de)

This work was supported in part by Synopsys Inc., under the Industry—University Cooperation Project "Efficient Implementation of 5G Baseband Kernels on a Vector Processor."

**ABSTRACT** As we target implementations of very high-end *3GPP 5$^{th}$ Generation New Radio* (5G NR) specifications and look towards the future, it becomes apparent that the stringent execution deadlines in *physical layer* (PHY) procedures are hard to satisfy using traditional algorithms optimised for high throughput. Hence, if the designer adheres to the same throughput efficient algorithm and simply scales up the *hardware* (HW), the device effectively becomes overprovisioned, costing more than it would if the designer opted for a latency efficient algorithm. However, latency efficient algorithms cost more operations per transmitted data item, and therefore consume more power compared to throughput efficient algorithms. Consequently, if the designer opts for latency efficient algorithms, the implementation would be power inefficient for all but the latency-critical use cases. We identify the use-cases where these problems occur in the *channel estimation* (CE) PHY procedure and propose a *software* (SW) implementation that can dynamically switch between latency and throughput efficient algorithms and thereby avoid both unnecessary HW overprovisioning and excess power consumption. In this article we demonstrate this with an example implementation of CE for the 5G NR high-end and quantify the benefits of this approach.

**INDEX TERMS** 5G, 6G, channel estimation, interpolation, workloads, HW, SW, requirements, MPSoC mapping, SIMD, VLIW, vector processor, DSP, implementation, latency, throughput, trade-off.

## I. INTRODUCTION

WHEN implementing *physical layer* algorithms the designer starts with having to make a couple of decisions, for which a set of questions can ease the decision making process.

First, what kind of mathematical operations are used in the targeted algorithm? It is essential to get the idea of what kind of software (SW) and hardware (HW) support is needed and if parallel processing or some form of a repeating pattern in processing can be exploited.

Second question: in addition to no dependencies on adjacent data items, can the same operation be applied to multiple data? If the answer is yes, which is true for many *physical layer* algorithms including channel estimation (CE), then the algorithm is suitable for specific type of fine-grained parallel processing called *vector processing*. In SW this means *single instruction, multiple data* (SIMD) style parallel processing is possible, and in HW algorithm implementations the designers will have more freedom to exploit parallelism in design of functional units and control of the data processing pipeline that can simplify the design. If an algorithm can be structured as a many iteration loop with its data items independent between the loop iterations, then such an algorithm is well suited for *vector processing*.

The third question, after deciding if *vector processing* can be used is: do you want to implement the algorithm in SW or in HW? Technically, everything can be done in SW or everything in HW, but there are some benefits and drawbacks to both. SW is flexible, easy to maintain and update, and is friendly to HW reuse by running different SW kernels[1] in a time multiplexed fashion on the same processor. Other times when we want to opt for a SW implementation is when we expect the kernel functionality to change over time or when for the same functionality there are changing conditions, e.g., CE under different channel conditions. From another perspective the processors on which this SW would run have a larger footprint compared to a specific dedicated kernel HW, hence, they are more expensive per unit and probably not as energy efficient as the dedicated HW kernel for the specific task. By implementing a kernel as a well optimised HW module, the kernel could save power, cost per unit and potentially have higher throughput relative to a processor. But, if you end up needing to implement multiple algorithms for the same functionality with many parameters as the case is in CE, then the resulting HW would be large, configurable, and resembling a processor in terms of layout, size and power consumption, defeating the purpose of implementing them in HW to begin with. Updating HW is much harder than updating SW, so you may decide to avoid HW implementations when the algorithm that should do the job is changing over time, or the algorithm for the job is not fixed. CE is a big territory, where most vendors keep secrets on how they do CE. There are many algorithms suitable for CE [1], [2], [3], [4], [5], [6], [7], [8]. These vary vastly in complexity and *channel estimate* ($\hat{\boldsymbol{H}}$) quality for particular channel conditions. Interestingly, a more compute intensive algorithm does not produce a better quality $\hat{\boldsymbol{H}}$ in all channel conditions [2], nor does a better quality $\hat{\boldsymbol{H}}$ facilitate good quality transmission in all channel conditions [9]. Therefore, a good CE scheme uses a multitude of algorithms, to deliver a high *quality of service* (QoS) to the user. In addition, the new communication standard 3GPP 5th Generation New Radio (5G NR) requires a plethora of dynamic SW defined transmission configurations [10, Sec. 7] that strain the need for flexibility. With the above in mind, it is highly beneficial to implement the CE *vector processing* kernel in SW.

The fourth question, after deciding for a SW implementation: what resources are required to meet the throughput and deadline[2] requirements of targeted use cases? This article tackles this question in depth. Section II defines use cases and their processing requirements based on the latest 5G NR specifications. From these we draw conclusions about future industry trend, particularly on the binding kernel deadlines and how traditional algorithms optimised for high throughput, although necessary in many cases, are not optimal in high-end cases due to the risk of overprovisioning. The core contribution of this article is identifying such a case among processing steps within CE, as well as dealing with the challenge of efficiently mapping such a kernel onto a *vector digital signal processor* (vDSP) with a *very long instruction word* (VLIW) and SIMD architecture. For a further discussion on VLIW and SIMD vDSP engines in mobile communications and multi-standard support we would like to point the reader towards [11]. Alternatively, the CE scheme can also be performed in SW on a *graphics processing unit* (GPU), but in this article we show that a single core vDSP is sufficient to get the CE job done efficiently without losing determinism in scheduling and memory accesses on the one hand, and a lower power consumption and footprint compared to a standard GPU on the other hand. Section IV introduces the processing stages within CE along with MATLAB simulations and mathematical formalism to support it. Section V covers the idea of latency and throughput trading, algorithmic optimisations, pseudo code, vectorisation and other implementation aspects considered. Next, Section VI demonstrates the impact of throughput and latency trading in SW, with supporting cycle counts and latency measurements[3] of the implemented algorithm variants. Last, in Section VII we validate the presented numbers against theoretical bounds and show that the results are indeed representative. The conclusion is in Section VIII.

## II. PROBLEM DEFINITION

As we are writing this article the development of modems and 3GPP standard specification for 5G NR is in full swing. The giants Huawei's HiSilicon, Qualcomm and Samsung are reporting 7.5 $Gb/s$ [12], 7.5 $Gb/s$ [13] and 7.3 $Gb/s$ [14] as *downlink* peak data rates, respectively. However with the data rates ever increasing we see a new challenge. The contributions of this article are twofold: First, we believe that with consistently growing data rates, a new creeping threat is going to be latency, to an extent that the desired rate will be less of a problem than stringent deadlines. Second, we provide the reader with a methodology to overcome numerous challenges when implementing these high-end cases on machines with two levels of parallelism, namely a SIMD vDSP with a VLIW architecture, such that the vDSP is fully utilised.

## III. LATENCY, THROUGHPUT, AND TRENDS IN 5G NR

As we show below (see Fig. 1) the *state-of-the-art* (SotA) data rates are impressive and on track to reach the full potential of active 5G NR specifications. In this section we introduce to the reader active and planned standard specification and how the constraints put on by the standard impact latency and throughput requirements of the CE kernel, as well as force a fresh way of thinking into their implementation.

1. In computer science terminology: a self-contained code or operation sequence that fully capture the functionality of an algorithm.

2. Simplified: throughout this article we abbreviate the term execution deadline with deadline and when we say that the latency exceeds the deadline we are referring to the execution time being longer than that which the deadline permits.

3. The latency within a processing step is calculated by measuring the number of cycles it takes data to propagate from the input of the kernel till the last data item of the associated OFDM symbol is processed and then dividing that number with the allocated clock frequency.

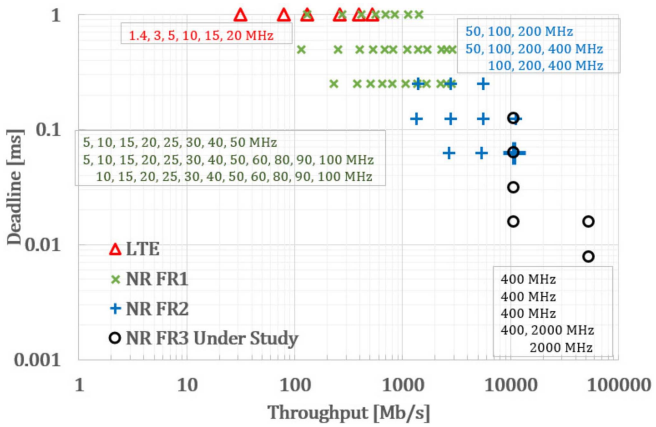


**FIGURE 1.** Calculated Throughput and Deadlines of the Channel Estimation Kernel Assuming 16-bit real + 16-bit imaginary Data Precision for all 3GPP Specified and Under Study Use Cases per Carrier Component Bandwidth Configuration.

### A. 3GPP SPECIFICATION: DATA RATES AND LATENCY

The active 3GPP 5G NR specifications cover two frequency ranges: *frequency range 1* (FR1) ($0.41 GHz - 7.125\ GHz$) [15] and *frequency range 2* (FR2) ($24.25\ GHz - 52.6\ GHz$) [16]. 5G NR in FR1 overlaps with *3GPP* $4^{th}$ *Generation Long Term Evolution* (4G LTE) and other legacy standards. From specifications [15], [16] the duration of 14 *Orthogonal Frequency Division Multiplexing* (OFDM) symbols or *transmission time interval* (TTI)[4] and therefore associated use case deadlines of FR2 go down 16×, whilst the throughput per *carrier component*[5,6,7] per ms goes up 11× compared to the high-end of 4G LTE. Additionally, as per conclusions of the $86^{th}$ 3GPP *radio access network technical specification group* meeting [17], 3GPP is studying a new frequency range *frequency range 3* (FR3) ($52.6\ GHz - 71\ GHz$) for use in future 3GPP *Release 17* (R17) of 5G NR. The initial report [18] of the mentioned study envisions even higher throughput and shorter TTI and therefore shorter deadlines in R17 FR3 compared to FR2 ($24.25 GHz - 52.6\ GHz$). Hence, the need to balance throughput and latency aligns with 3GPP specification development on the way towards 6G.

From the TTI duration we can calculate the latency budget. As stated in [19], [20] the deadline for *digital baseband physical layer* is constrained by the *hybrid automatic repeat request* (HARQ) *media access control layer* (MAC-L) procedure, which is 3 TTIs long. As a rule of thumb we assume up to $^1/_3$ of the 3 TTI budget to be associated with CE, whilst the other $^2/_3$ are reserved for other processing steps (see Section IV-B), e.g., synchronisation, waveform demodulation, decoding, etc.[8] Since we are interested to showcase intricacies of implementing channel estimation the next step is to determine the amount of data that needs to be processed for the mentioned use cases under the TTI imposed deadline. Per RB (see note[7]), the kernel needs to output $\hat{\boldsymbol{H}}$[9] data items, for every RE, hence to get the processing load per TTI we need to multiply the number of bits per RE, number of REs per RB, number of RBs per *carrier component*, and number of supported *carrier component*s. In Fig. 1 we show the calculated throughput and associated deadlines per *carrier component* with 16-bit real + 16-bit imaginary precision for REs. We can observe the deadline halving by two with every row in Fig. 1, reaching as low as $62.5\ \mu s$ in the FR2 corner. This contraction of the deadline is due to the inverse relationship between the duration of OFDM symbols and their *subcarrier frequency spacing*, which increases by two with every row in Fig. 1. Specifications allow two mechanisms for extending the total number of *carrier component*s: *carrier aggregation* (CA) up to $4 \times 400\ MHz$ [21] and *multiple-input, multiple-output* (MIMO) up to 8 spatial data layers[10] [10]. *Carrier aggregation* utilises the excess available frequency spectrum, whilst MIMO provides additional *carrier component*s on a separate link.

Reaching 6G we cannot rule out the simultaneous use of both *carrier aggregation* and MIMO in high-end use cases, therefore we include both in our further consideration. As of writing this article the FR3 study item use cases are not yet part of the active specifications, so in Fig. 1 we select the highlighted high-end FR2 use case for demonstration of an example use case where deadlines are endangered when using the throughput efficient algorithm. This way the examined case is both standard compliant and foreshadowing of the challenges to come in future 3GPP specification instalments. In Table 1 we list other use cases for comparison with the selected high-end FR2. These use cases are used to illustrate the decline of excess unused (free) latency budget of CE in Fig. 2.

4. Simplified: in this article we use the term TTI to refer to a data packet consisting of 14 OFDM symbols. In a more advanced understanding there are exceptions to this rule and a TTI can be longer or shorter than 14 OFDM symbols with several data packets (in the sense of a transport block code words) multiplexed within a TTI.

5. 3GPP terminology for a communication channel identified by allocated *bandwidth* (BW) and numerology, depending on the device class it can support a different number of *carrier component*s. *Carrier components* consist of several *resource blocks* (RBs).

6. Numerology is a 3GPP term for OFDM symbol's *subcarrier frequency spacing*.

7. A *resource block* (RB) is a unit in 3GPP terminology representing a grid section onto which *quadrature amplitude modulation* (QAM) symbols are mapped; 1 data item of that grid is called a *resource element* (RE), $1\ RB = 12$ subcarriers$\times 14$ OFDM symbols$=168$ RE, with some exceptions to the rule.

8. No reference for this rule of thumb, the actual timing budget distribution is vendor specific and kept private. We use this value as a first order approximation of the upper bound, so that we can be sure if the latency problem exists for the bounded value, it will certainly exist for more conservative deadline estimates too. Hence, further use of deadline is referring to this bound.

9. Important to note that $\hat{\boldsymbol{H}}$ does not represent user data that has been transferred via the communication system, but rather the change that the pilots have undergone during communication with a goal of reverting the change in the equalisation process.

10. Simplified: a link, not to be confused with massive MIMO which is a method of generating a link via constructive and deconstructive interference of radio waves from multiple antennas.

**TABLE 1.** Compared use cases.

| Use Case | CA | MIMO | Throughput | Deadline |
|---|---|---|---|---|
| | $[CCs]$ | $[layers]$ | $[Gb/s]$ | $[\mu s]$ |
| High-end LTE | 1 | 1 | 0.525 | 1000 |
| Mid FR1 | 1 | 1 | 1.396 | 500 |
| Mid FR2 | 4 | 1 | 10.83 | 125 |
| High-end FR2 | 4 | 8 | 336 | 62.5 |

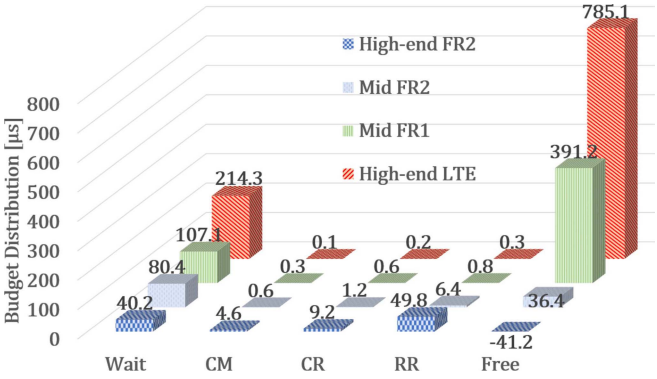


**FIGURE 2.** Latency Distribution in CE Assuming a 1 GHz Budget.

### B. LATENCY BUDGET DISTRIBUTION IN CHANNEL ESTIMATION

CE in 4G LTE and in 5G NR works by transmitting predefined data items on predefined RE locations within RBs, called *reference signal* (RS) in 3GPP terminology or pilots in the broader communications community. These predefined RE are coded with 2-bit *quadrature phase shift keying* (QPSK) in 5G NR[11] on the *transmitter* (Tx) side [10, Sec. 7.4.1.5], but measured with a higher resolution, e.g., 32-bit on the *receiver* (Rx) side, to precisely identify changes that the pilots within RBs have been subject to whilst in the channel between Tx and Rx. For non-pilot REs the $\hat{\boldsymbol{H}}$ is generated from the observed changes to the pilot REs by a series of interpolations or extrapolations [2], [3], [22], [23]. The latency budget of one TTI duration is divided among several processing steps and buffer wait times within the CE kernel. For an example algorithm sequence (see Section IV-C) the latency budget can be written as

$$t_{budget} = t_{wait} + t_{CM} + t_{CR} + t_{RR} + t_{free}. \quad (1)$$

$t_{budget}$ is the total size of the latency budget in seconds and it is fixed to the TTI duration for that use case. $t_{wait}$ depends on the pilot layout and the interpolation algorithm, e.g., the modem may have to wait for a couple of future pilots carrying OFDM symbols of the RBs before processing the current $\hat{\boldsymbol{H}}$. Depending on which interpolation algorithm the designer wants to use, e.g., use more future pilots, this wait time can increase. $t_{budget}$ and $t_{wait}$ represent the structural latency of the system. $t_{CM}$, $t_{CR}$, $t_{RR}$ are the time spent on CE steps *channel measurement* (CM), *column reconstruction* (CR) and *row reconstruction* (RR). Time spent on processing depends on the algorithms in these steps, as well as the HW and the SW used to implement them. $t_{CM}$, $t_{CR}$, $t_{RR}$ represent the implementation latency. $t_{free}$ is the leftover *time till deadline*.

Assuming an allocated frequency budget of 1 GHz for the vDSP CE procedure and an optimised implementation (see Section VII) of the traditional high throughput algorithms for CE, in Fig. 2 we show the latency budget distribution among CE processing steps of Table 1 use cases.

From Fig. 2 we can observe two things: first, $t_{free}$ gets smaller and eventually becomes negative in the high-end FR2 case; and second, the order of magnitude difference between the pilot dependent $t_{wait}$ and the implementation specific $t_{CM}$, $t_{CR}$, $t_{RR}$ is decreasing as you move from 4G LTE towards advanced 5G NR FR2 use cases. Negative $t_{free}$ means that the deadline is broken and cannot be achieved for the given implementation without allocating more HW resources, optimising the SW more or changing the algorithms. As we note in Section VII the code is well optimised reaching the asymptotic bound for the vDSP, so optimising the SW further is off the table. Scaling up the HW can be costly, so best avoid if possible. This leaves investigating the algorithms as the preferred option. Further, the second observation of the above Fig. 2, indicates that the structural wait time $t_{wait}$ due to pilot layout and OFDM symbol duration does not dominate the budget of the high-end FR2, rather *row reconstruction*, which depends on the implementation solely, is the biggest spender in the latency budget and reducing it could save the budget deficit. Therefore, the best spot to start the investigation is the algorithm of *row reconstruction* processing step.

### C. SPECIFICATION TRAJECTORY TOWARDS 6G

As we saw in Fig. 1 the specifications are evolving on a path from *low rate - high latency* towards *high rate - low latency* use cases. Rather than a revolutionary jump from 4G LTE to 5G NR New Radio (NR), we see that the industry and its specification body are taking incremental evolutionary steps in terms of rate and latency. Based on the observation we can predict a similar gradual change on the trajectory into 6G, with new use cases for handsets appearing in the bottom right *high rate - low latency* corner of the throughput-deadline graphs.

Another intuitive way of viewing why the latency constraint of the channel estimation kernel is becoming stricter as we move towards newer standards is considering the change of the channel coherence time.[12] With new frequency ranges (see Section III-A) and operation in the upper mmWave spectrum, the channel coherence time is becoming ever shorter [24]. The channel measurement is valid for a shorter interval compared to lower GHz operation of older standards and therefore the measurement and its subsequent processing steps within channel estimation have to be done

11. There can be between 4 and 24 pilots per RB.

12. A period during which the channel instance can be considered highly correlated. In practice this would mean that another channel measurement is not needed in this period.

faster, i.e., within a shorter deadline. Similarly, we can expect throughput requirements to go up in parallel, higher subcarrier spacing and therefore shorter OFDM symbols on the one hand and higher overall number of subcarriers per OFDM symbol through spatial layers and *carrier aggregation* on the other hand, increase the overall number of data items that need to be processed in a given time period.

## IV. CHANNEL ESTIMATION

This section describes the 5G NR CE in formal mathematical language, followed by a MATLAB simulation. We use the simulation to identify the suitable algorithm for implementation on the vDSP, based on the channel quality requirement for the FR2 high-end use case and estimation error of the candidate algorithms. Mathematical analysis complements the simulation and helps us understand the origin of the estimation error.

### A. CHANNEL MODEL

We use measurement based channel models [25, Sec. 7.7] for the 0.5 - 100 GHz frequency range and assume a separate channel for every Tx - Rx antenna pair. Channel's Doppler spread per multipath component parameter is set to $10\,Hz$ and the delay spread of multipath components parameter is set to $300\,ns$, to reflect observations from [26, Fig. 3]. Namely, the delay spread of multipath components in mmWave channels is well observable, whilst Doppler spreading of individual multipath components is rather small and clustered. In addition, the effects of thermal noise from the devices' circuitry and low power spurious spectral emissions are modelled as an *additive white Gaussian noise* (AWGN) complex number sequence $w(n)$ with zero mean and variance $\sigma_w^2$. One such channel is assumed constant in both time and frequency over one RE of the 5G NR RB (due to scalable subcarrier spacing and symbol samples' duration of the 5G waveform). Channels are considered time and frequency variant between two different REs. Assuming all REs of all RBs in transmission form a *resource grid* (RG) of size $N \times M$ REs, the relationship between the Tx and Rx RG can be expressed in matrix form via *Hadamard-Schur* product[13] and addition as:

$$\boldsymbol{Y} = \boldsymbol{H}_0 \circ \boldsymbol{X} + \boldsymbol{W}, \tag{2}$$

where $\boldsymbol{Y} \in \mathbb{C}^{N\times M}$ is the Rx RG, $\boldsymbol{X} \in \mathbb{C}^{N\times M}$ is the Tx RG, $\boldsymbol{H}_0 \in \mathbb{C}^{N\times M}$ is the *channel frequency response* in the RG form, and $\boldsymbol{W} \in \mathbb{C}^{N\times M}$ is the *independent identically distributed* (i.i.d.) AWGN sequence $w$ with zero mean and variance $\sigma_w^2$. If we were to depict REs (data items) with $k \in \{1, 2, \ldots, N\}$ representing the row, i.e., subcarrier index and $l \in \{1, 2, \ldots, M\}$ representing the column, i.e., OFDM symbol index, (2) becomes:

$$\boldsymbol{Y}(k, l) = \boldsymbol{H}_0(k, l) \cdot \boldsymbol{X}(k, l) + \boldsymbol{W}(k, l). \tag{3}$$

As mentioned in Section III-A for channel measurement purposes the standard imbues the Tx RG of every antenna port with a pseudo random reference signal on certain $l$ and $k$ indices coded with QPSK. In 5G NR there can be anywhere between 4 and 24 pilots per RB in a variety of patterns [10, Tabs. 7.4.1.1.2-1/5]. The channel [25, Sec. 7.7] model under high-end FR2 conditions exhibits moderate time and frequency selectivity[14] with a dominant *line-of-sight* (LOS) cluster in the *channel impulse response* ($\boldsymbol{h}_0$), which matches other observations from literature of mmWave channels [24], [27], [28], [29], [30]. For our MATLAB simulations we have selected a pattern[15] that can perform well in moderate selectivity, however in practice the pilot layout is dynamically assigned by the base station based on a vendor specific control algorithm. The dynamic allocation schedules different pilot layouts among different antenna ports, limiting interference and helping identify the Tx - Rx antenna pairs on the Rx side. Within the RG we denote the row pilot index with $k_p \in \{k_{p1}, k_{p2}, \ldots, k_{pK}\}$ and the column pilot index with $l_p \in \{l_{p1}, l_{p2}, \ldots, l_{pL}\}$. Boldface $\boldsymbol{k}_p$ and $\boldsymbol{l}_p$ are vectors of all row and column pilot indices, respectively. $\boldsymbol{k}_p$ and $\boldsymbol{l}_p$ as well as pilot values are taken to be known and shared between the Tx and Rx devices. Additionally, like in 4G LTE, pilot locations between consecutive columns $l_{pi}$ and $l_{p(i+1)}$ can be configured to alternate between different subsets of $\boldsymbol{k}_p$ such that:

$$\boldsymbol{k}_p = \bigcup_s \boldsymbol{k}_{p_s}, \tag{4}$$

where $\boldsymbol{k}_{p_s}$ are all possible subsets of $\boldsymbol{k}_p$ of size $S \leq K$.

The duration of $\boldsymbol{h}_0$ is met with a cyclic guard interval called *cyclic prefix*, which scales with $\boldsymbol{h}_0$ regardless of many different modes of operation in 5G NR devices. Furthermore its duration in samples can be used as the upper bound of sparsity when estimating $\boldsymbol{h}_0$ [6], [7].

### B. DOWNLINK RECEIVER

Figure 3 shows the *digital baseband physical layer downlink* Rx system, made up of a series of intertwined processing steps that can be divided into six blocks: synchronisation, waveform demodulation, channel equalisation, resource demapping, decoding and the highlighted CE. As mentioned in Section III-A the latency budget is constrained by the HARQ procedure and leaves the CE with one TTI duration deadline to work with.

### C. ESTIMATION STEPS

CE of a channel described in Section IV-A can be performed in multiple stages as shown in Fig. 4. The procedure is not standardised by 3GPP, however the community [2], [3], [22], [23] points towards an organisation in two parts: estimation at pilot and non-pilot RG indices, whilst keeping the algorithms within those two parts vendor specific. CE is done after synchronisation and waveform demodulation, and conversion

13. Corresponding to MATLAB's ".*" operation.

14. Simplified: how fast do the channel values change across the time and frequency axis.

15. Illustrated in Fig. 7 bottom left, see [10, Tab. 7.4.1.1.2-4] for a list of possible patterns.

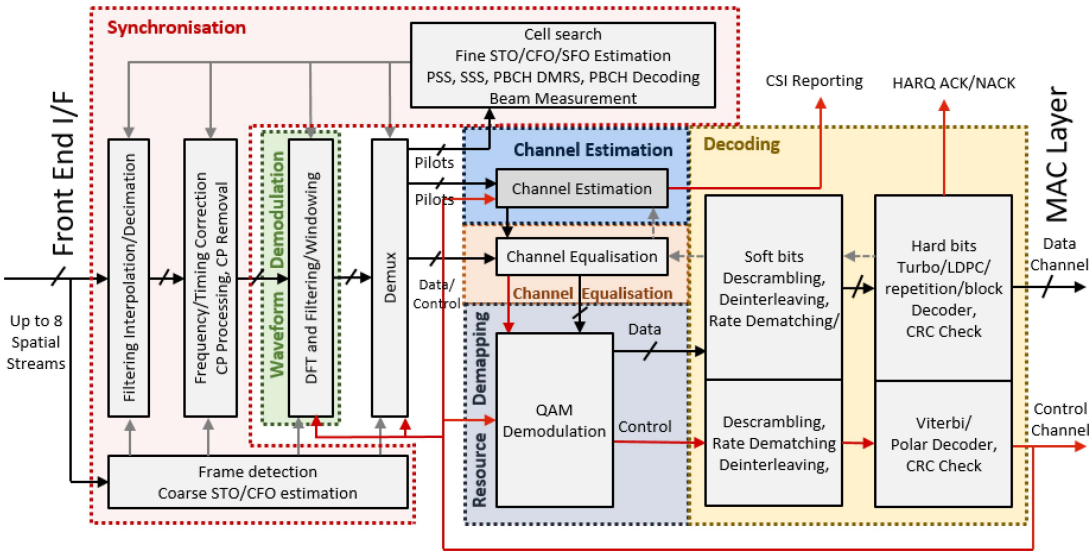


**FIGURE 3.** 3GPP Downlink Receiver DBB PHY System Diagram.

of OFDM symbols from time to the frequency domain via *discrete Fourier transform* (DFT), hence we are estimating $\boldsymbol{H}_0$ and not $\boldsymbol{h}_0$, see Fig. 3.

*Channel measurement* is the first step of CE which compares the values of the sent pseudo random pilot sequence with the received values of the pilot sequence at pilot locations $\boldsymbol{k}_{p_s}$ and $\boldsymbol{l}_p$ of the RG. Observing the change in pilot values between Tx and Rx gives us an indication of the channel for the observed Tx - Rx antenna pair. There are two main approaches to this stage: *least squares* or Linear Minimum Mean Square Error[16] [1]. The former is more optimal in the computational sense, since it has a substantially lower computational complexity and the output can be further filtered in the next stage to improve quality in the error reduction sense [23]. The *least squares* measurement can be described as:

$$\boldsymbol{H}_m = \boldsymbol{Y}(\boldsymbol{k}_{p_s}, \boldsymbol{l}_p) \circ \overline{\boldsymbol{X}(\boldsymbol{k}_{p_s}, \boldsymbol{l}_p)} \oslash |\boldsymbol{X}(\boldsymbol{k}_{p_s}, \boldsymbol{l}_p)|^{\circ 2}, \tag{5}$$

where $\boldsymbol{H}_m \in \mathbb{C}^{S \times L}$ is the channel measurement, $\bar{\cdot}$ is the complex conjugate in the *Hadamard-Schur* sense, $\oslash$ is the *Hadamard-Schur* division, $|\cdot|$ is the complex modulus in the *Hadamard-Schur* sense, and $(\cdot)^{\circ 2}$ is the square in the *Hadamard-Schur* sense. Evaluating (5) we get:

$$\begin{aligned} \boldsymbol{H}_m &= \boldsymbol{H}_0(\boldsymbol{k}_{p_s}, \boldsymbol{l}_p) + \boldsymbol{W}(\boldsymbol{k}_{p_s}, \boldsymbol{l}_p) \oslash \boldsymbol{X}(\boldsymbol{k}_{p_s}, \boldsymbol{l}_p) \\ &= \boldsymbol{H}_0(\boldsymbol{k}_{p_s}, \boldsymbol{l}_p) + \boldsymbol{V}, \end{aligned} \tag{6}$$

where $\boldsymbol{V} \in \mathbb{C}^{S \times L}$ is an i.i.d. complex AWGN sequence matrix with zero mean and variance $\sigma_v^2$. $\sigma_v^2$ is defined as:

$$\sigma_v^2 = \sigma_w^2 \left| \frac{1}{p} \right|^2, \tag{7}$$

where $|p|^2$ is the power per QPSK coded pilot tone. We can also define the error matrix $\boldsymbol{e}$ for the channel estimate at pilot locations as:

$$\boldsymbol{e} = \boldsymbol{H}_0(\boldsymbol{k}_{p_s}, \boldsymbol{l}_p) - \boldsymbol{H}_m = -\boldsymbol{V} = \boldsymbol{e}_v. \tag{8}$$

Next stage is optional and involves low pass filtering of $\boldsymbol{H}_m$, across rows and columns to reduce $\boldsymbol{e}_v$, formally written as:

$$\boldsymbol{H}_{m,f} = \boldsymbol{F}_S \cdot \boldsymbol{H}_m \cdot \boldsymbol{F}_L, \tag{9}$$

where $\boldsymbol{H}_{m,f} \in \mathbb{C}^{S \times L}$ is the 2D filtered $\boldsymbol{H}_m$, and $\boldsymbol{F}_S \in \mathbb{C}^{S \times S}$ and $\boldsymbol{F}_L \in \mathbb{C}^{L \times L}$ are the filtering matrices. Selecting proper filter coefficients of the noise suppressing filters is a key step, since it will introduce a new error $\boldsymbol{e}_{H_0,f}$, which represents the loss of information about the *channel frequency response* ($\boldsymbol{H}_0$), whilst reducing $\boldsymbol{e}_v$. We represent the error of $\boldsymbol{H}_{m,f}$ compared to the true $\boldsymbol{H}_0$ value as:

$$\boldsymbol{e}_f = \boldsymbol{H}_0(\boldsymbol{k}_{p_s}, \boldsymbol{l}_p) - \boldsymbol{H}_{m,f} = \boldsymbol{e}_{H_0,f} + \boldsymbol{e}_{v,f}, \tag{10}$$

where $\boldsymbol{e}_{H_0,f}$ and $\boldsymbol{e}_{v,f}$ are the newly introduced information loss error[17] and reduced AWGN error, respectively. These two error components propagate and vary throughout the rest of the CE stages. For the purpose of this work we do not perform filtering, i.e., matrices $\boldsymbol{F}_S$ and $\boldsymbol{F}_L$ are identity matrices, since the high-end use cases are scheduled by the base station if the reported $1/\sigma_v^2$ is high, i.e., good channel conditions and therefore $\boldsymbol{e}_v$ is small and we can avoid introducing $\boldsymbol{e}_{H_0,f}$. With this stage done, we have completed the estimation of the channel at pilot indices and can start the estimation at non-pilot indices.

*Symmetry reconstruction* is an optional step, which has the purpose to simplify following processing steps. If the pilot layout is alternating between different subsets of $\boldsymbol{k}_p$ as per (4), then $\boldsymbol{H}_{m,f}(k_s, l_p)$ for a fixed $k_s$ maps to several subcarrier indices $k$ depending on the OFDM symbol

16. In literature Linear Minimum Mean Square Error can be also found under Winner filtering.

17. Sometimes referred to as the channel model mismatch.

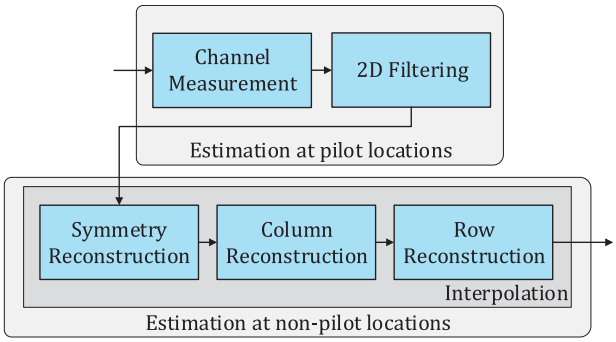


**FIGURE 4.** Deconstructing Channel Estimation into Stages.

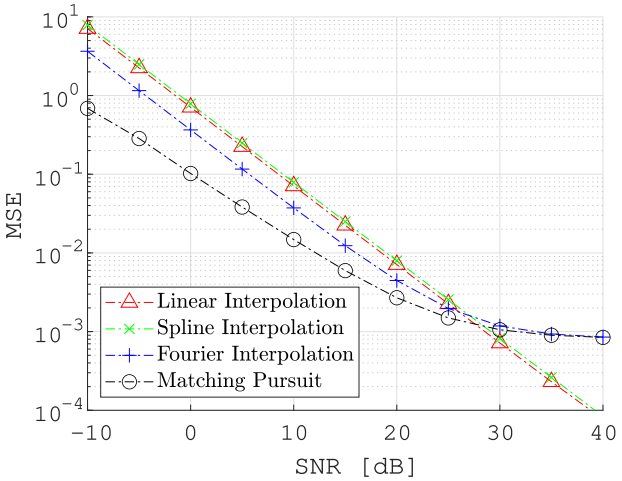


**FIGURE 5.** Column Reconstruction Stage Algorithms: MSE Comparison for [25, Sec. 7.7] TDL-C, 300 ns Delay Spread and 10 Hz Doppler Shift.

$l_p$. This means that the processing down the line would need to be $l_p$ specific, which can limit possible algorithmic and parallelism choices. *Symmetry reconstruction* avoids this limiting inconvenience by interpolating additional elements into the $\boldsymbol{H}_{m,f}$ matrix. In the newly created matrix $\boldsymbol{H}_{m,f,t} \in \mathbb{C}^{K\times L}$ every row maps to one and only one subcarrier index. Since the interpolation is column specific, the resulting $\boldsymbol{H}_{m,f,t}$ is formally expressed as a concatenation of individually interpolated columns:

$$\begin{aligned}\boldsymbol{H}_{m,f,t} &= \left[\boldsymbol{H}_{m,f,t_1}, \boldsymbol{H}_{m,f,t_2}, \ldots, \boldsymbol{H}_{m,f,t_L}\right],\\ \boldsymbol{H}_{m,f,t_l} &= \boldsymbol{T}_l \cdot \boldsymbol{H}_{m,f_l},\end{aligned} \tag{11}$$

where $\boldsymbol{T}_l \in \mathbb{C}^{K\times S}$ is the interpolation matrix for the $l^{th}$ column of $\boldsymbol{H}_{m,f}$ resulting in the $l^{th}$ column of $\boldsymbol{H}_{m,f,t}$. Note that vendor specific algorithms for this stage can be more or less sophisticated and that the weights in the interpolation matrix $\boldsymbol{T}_l$ can also be a function of $\boldsymbol{H}_{m,f}$ values surrounding the $l^{th}$ column, e.g., up to $l-2$ and $l+1$ as is the case in [23]. For the vDSP implementation we have selected a symmetric 5G NR pilot layout where $\boldsymbol{k}_{p_s} = \boldsymbol{k}_p$, i.e., $S = K$ and therefore the symmetrisation step is omitted. In our MATLAB simulations of CE for the channel in [25, Sec. 7.7] classic methods like *linear interpolation* (linear) and *spline piecewise cubic Hermite interpolation* (pchip) between neighbouring columns of $\boldsymbol{H}_{m,f}$ show about $3\times$ improvement in quality of the final $\hat{\boldsymbol{H}}$ for non-symmetric pilot patterns compared to not using this stage. When not using this stage, $\boldsymbol{T}_l$ is defined as a set of $K \times K$ identity matrices.

*Column reconstruction* is a stage which interpolates columns of $\boldsymbol{H}_{m,f,t}$, such that the size of columns after interpolation matches the column size of the RG, i.e., $K \to N$:

$$\boldsymbol{H}_{m,f,t,c} = \boldsymbol{C} \cdot \boldsymbol{H}_{m,f,t}, \tag{12}$$

where $\boldsymbol{C} \in \mathbb{C}^{N\times K}$ is the column interpolation matrix, and $\boldsymbol{H}_{m,f,t,c} \in \mathbb{C}^{N\times L}$ is the result. The error at this stage can be represented as:

$$\boldsymbol{e}_{m,f,t,c} = \boldsymbol{H}_0\left(:, \boldsymbol{l}_p\right) - \boldsymbol{H}_{m,f,t,c} = \boldsymbol{e}_{H_0,f,y,c} + \boldsymbol{e}_{v,f,t,c}, \tag{13}$$

where $\boldsymbol{H}_0\left(:, \boldsymbol{l}_p\right)$ represents all row entries of $\boldsymbol{l}_p$ columns. Here we note that $\boldsymbol{H}_0$ has an underlying structure that can be exploited when choosing an interpolation algorithm for this stage with the aim to generate a higher quality channel estimate. Namely, columns of $\boldsymbol{H}_{m,f,t}$ can be interpolated with one of the Pursuit algorithms [31] or Fourier interpolation [1], due to the relation between $\boldsymbol{H}_0$ and $\boldsymbol{h}_0$ and known bound of sparsity for the $\boldsymbol{h}_0$. In our MATLAB simulation we compared 4 interpolation methods: linear, pchip, Fourier and Matching Pursuit, as shown in Fig. 5. In Fig. 5 the x-axis is $1/\sigma_v^2$ or Signal-to-Noise-Ratio (SNR) in dB and the y-axis is the Mean Square Error (MSE) of (13). $\sigma_v^2$ is the noise variance of $\boldsymbol{V}$ from (7). In high SNR conditions in which we expect the FR2 high-end to be scheduled we see that linear outperforms other methods. Interestingly, Fourier and Pursuit interpolations show good quality in low SNR conditions, where their filtering properties reduce the impact of $\boldsymbol{e}_{v,f,t,c}$. However those same filtering properties increase $\boldsymbol{e}_{H_0,f,y,c}$ which dominates the high SNR conditions, where filtering destroys part of the channel information. Due to observations and use-case requirements we proceed with linear implementation as the *column reconstruction* step.

The final step in $\hat{\boldsymbol{H}}$ is the *row reconstruction* (RR), for which we define the row interpolation matrix $\boldsymbol{R} \in \mathbb{C}^{L\times M}$ and mathematically express the interpolation as:

$$\hat{\boldsymbol{H}} = \boldsymbol{H}_{m,f,t,c,r} = \boldsymbol{H}_{m,f,t,c} \cdot \boldsymbol{R}, \tag{14}$$

where $\boldsymbol{H}_{m,f,t,c,r} \in \mathbb{C}^{N\times M}$ is the interpolated matrix and the final channel estimate $\hat{\boldsymbol{H}}$ matching the size of the RG. We define the error as:

$$\boldsymbol{e} = \boldsymbol{e}_{m,f,t,c,r} = \boldsymbol{H}_0 - \hat{\boldsymbol{H}} = \boldsymbol{e}_{H_0,f,y,c,r} + \boldsymbol{e}_{v,f,t,c,r}, \tag{15}$$

where $\boldsymbol{e}_{H_0,f,y,c,r}$ and $\boldsymbol{e}_{v,f,t,c,r}$ are the propagated errors from (10). Based on the MATLAB simulation Fig. 6 shows measured (15) as MSE on the y-axis and $1/\sigma_v^2$ as SNR in dB on the x-axis. We can see a slightly higher (about $1.5\times$) overall MSE of linear and pchip interpolation in Fig. 6 compared to their MSE in Fig. 5 due to an extra interpolation step, where small changes of $\boldsymbol{H}_0$ between pilot carrying OFDM

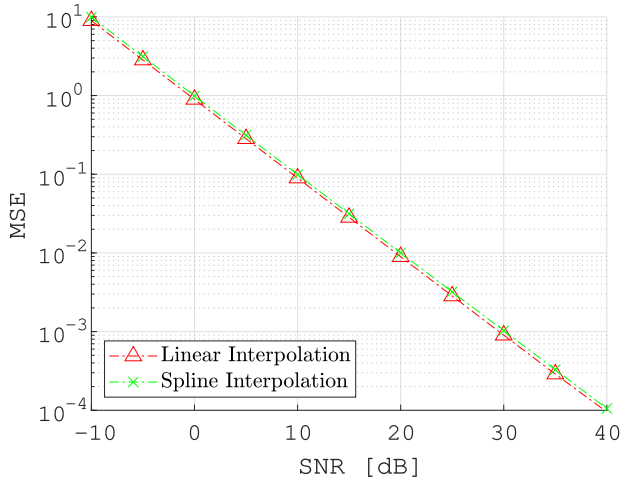


**FIGURE 6.** Row Reconstruction Stage Algorithms: MSE Comparison for [25, Sec. 7.7] TDL-C, 300 ns Delay Spread and 10 Hz Doppler Shift.

symbols are not captured linked to the information loss error $\boldsymbol{e}_{H_0,f,y,c,r}$. An interesting observation is also that linear and pchip interpolation perform similarly over the whole SNR range. This can be attributed to both methods being polynomial in nature, i.e., of first and third degree, respectively. In terms of quality linear slightly outperforms the computationally more complex pchip. We owe this fact due to a properly sampled channel with a good pilot configuration selected for the simulated channel on the one hand, and to pchip having a higher degree interpolation polynomial than the actual data would require on the other.

To sum up, for high-end use cases the algorithm choice in each processing step is favourable in terms of computational complexity: filtering after *channel measurement* can be omitted and linear is sufficient to get a good quality $\hat{\boldsymbol{H}}$ in both *column reconstruction* and *row reconstruction* steps. In Fig. 7 we show a simplified processing graph with selected algorithms of key steps and an illustration of how these processing steps populate $\hat{\boldsymbol{H}}$ of a single RB. You may notice that there are no pilots at the edges of the RG and these values have to be either extrapolated[18] or interpolated with pilots from the previous or future TTIs provided they exist.

If we follow the steps described in this section we will surely estimate the channel, it is just a matter of how good that estimate will be. The question "Where is the threshold which says this much MSE is good enough for the system?" has no easy answer and requires fine tuning based on channel conditions and noise, quality of other kernels in the system, modulation code rate schemes and acceptable codeword bit error rates based on applications. Reference [9] examines the impact of channel estimate MSE on the performance of a simple system and from it we can take that in low SNR MSE value is less important whilst in high SNR MSE value is more important. Hence, if we would imagine a "good enough" line on the MSE vs SNR graph, it would be falling

18. We use linear extrapolation as an extension of linear interpolation.

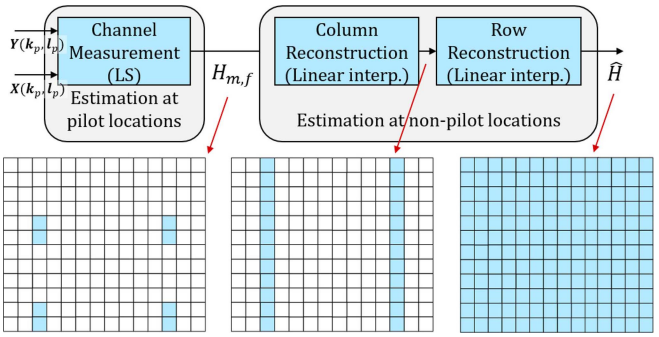


**FIGURE 7.** Channel Estimate Generation and Output of Key Steps Illustrated on a Resource Block.

with rising SNR, but it's gradient and offset would change based on environmental and system parameters.

## V. IMPLEMENTATION ASPECTS

### A. LATENCY AND THROUGHPUT TRADING

Latency and throughput trading is a known and widely used concept in computer science. In stream/data flow processing [32] and computing in general context switching of tasks/kernels has an overhead cost. Therefore waiting for a quantity of data to reach a threshold (batch) before processing that data is incentivised. This waiting period introduces some latency to increase throughput. Even further back in 1959, IBM 7030 Stretch used pipelining for the first time with a 4-stage pipeline, therefore trading some latency, logic, and memory for 100× throughput gain [33]. More recently however, with new applications and the end of Moore's law, the trend has been going the other way around, giving up some throughput for lower latency. This is not as easy as just reverting the changes since the throughput requirements have been increasing as well. Examples are big data in finance [34], data queuing in data centres [35] and packet routing on a network-on-chip [36]. The concept is widely used and with this work we explore its application to high-end CE.

### B. ALGORITHM OPTIMISATION

Since the interest of this article is the RR step, let us develop the throughput and latency efficient versions of linear interpolation from the starting equation:

$$\hat{\boldsymbol{H}}(k,l) = \hat{\boldsymbol{H}}\big(k,l_{pi}\big) + \frac{\hat{\boldsymbol{H}}\big(k,l_{p(i+1)}\big) - \hat{\boldsymbol{H}}\big(k,l_{pi}\big)}{l_{p(i+1)} - l_{pi}} \cdot \big(l - l_{pi}\big), \tag{16}$$

where $\hat{\boldsymbol{H}}\big(k,l_{pi}\big)$ is $\boldsymbol{H}_{m,f,y,c}$ of (12), row index $k \in \{1,2,\ldots,N\}$, and column index $l \in \{l_{pi}+1, l_{pi}+2,\ldots,l_{p(i+1)}-1\}$. We see that there are many operations per data item, i.e., a *division* (*div*), 3× *subtraction* (*sub*), a *multiplication* (*mpy*), an *addition* (*add*) and at least 2× *fixed point type cast* (*cast*).

First, let us bind (16) to a local reference frame of a single block between two known columns such that $l_{pi} = 0$ is the

time index of the left known column and $l_{p(i+1)} = R_{ps}$ is the time index of the right known column, where $R_{ps}$ is the row pilot spacing for that block. This removes two *subs* from the second term of (16). Consequently, the column index changes to $l \in 1, 2, \ldots, R_{ps} - 1$. Next, we notice that $R_{ps}$ is small in general, in our demo case $R_{ps} = 9$, so there are not many data items to process along the time axis and we could use a preloaded *look-up table* (LUT) and potentially remove some operations from (16) and as a byproduct also save some memory bandwidth. Division and multiplication are commutative, so we can reorder the operations and get:

$$\begin{aligned}\hat{\boldsymbol{H}}(k,l) &= \hat{\boldsymbol{H}}(k,0) + \left(\hat{\boldsymbol{H}}\left(k,R_{ps}\right) - \hat{\boldsymbol{H}}(k,0)\right) \cdot \frac{l}{R_{ps}} \\ &= \hat{\boldsymbol{H}}(k,0) + \left(\hat{\boldsymbol{H}}\left(k,R_{ps}\right) - \hat{\boldsymbol{H}}(k,0)\right) \cdot \boldsymbol{x}(l), \quad (17)\end{aligned}$$

where $\boldsymbol{x}$ is the LUT. This removes *div* and replaces it with a LUT. This leaves us with a *sub*, a *mpy*, an *add* and a *cast* per data item. Next step is to define (17) recursively as:

$$\hat{\boldsymbol{H}}(k,l) = \hat{\boldsymbol{H}}(k,l-1) + \left(\hat{\boldsymbol{H}}\left(k,R_{ps}\right) - \hat{\boldsymbol{H}}(k,0)\right) \cdot \dot{\boldsymbol{x}}(l), \quad (18)$$

where $\dot{\boldsymbol{x}}$ is a new LUT. Equations (17) and (18) have same number of operations per data item, but (18) is a stepping stone to the final equation. If we were to unwind (18) all the way, we could express it as:

$$\hat{\boldsymbol{H}}(k,l) = \hat{\boldsymbol{H}}(k,0) + \sum_{i=1}^{l}\left(\hat{\boldsymbol{H}}\left(k,R_{ps}\right) - \hat{\boldsymbol{H}}(k,0)\right) \cdot \dot{\boldsymbol{x}}(i), \quad (19)$$

which opens up the possibility to use the *multiply-accumulate* (*mac*) operation efficiently. Even though it is also possible to replace *mpy* and *add* of (17) with a *mac* operation, it would require reloading/resetting of the *accumulator* (ACC) register for every $k$ and $l$, which does not reduce the overall number of operations needed.

Equation (19) is an arithmetic progression with a common difference for a fixed $k$. This situation is ideal for *mac* operations since the ACC register only needs to be set once per $k$ and due to the recursion all the other updates to the ACC register happen via *mac*. It may seem that for every $l$ the sum has to be recalculated, but due to the recursion all terms but the last, i.e., all but $i = l$ are already computed in the previous $l - 1$ steps and that value is available in the ACC. Notice that the subtraction $\hat{\boldsymbol{H}}\left(k,R_{ps}\right) - \hat{\boldsymbol{H}}(k,0)$ needs to be computed only once per $k$. In (19) we have a *mac* and a *cast* for $l \neq 1$ and an extra *cast* and *sub* when $l = 1$ to preload the ACC register and a regular data register with $\hat{\boldsymbol{H}}(k,0)$ and $\hat{\boldsymbol{H}}\left(k,R_{ps}\right) - \hat{\boldsymbol{H}}(k,0)$, respectively. This means that per data item (19) has 4 operations when $l = 1$ and 2 in other cases; compared to (17) which has 4 operations per data item regardless of the indices. Further, because of the common difference between neighbouring terms in the sum and a fixed $R_{ps}$ for the whole segment of the RG, the size of $\dot{\boldsymbol{x}}$ can be reduced to a single entry:

$$\dot{\boldsymbol{x}}(1) = \cdots = \dot{\boldsymbol{x}}(i) = \cdots = \dot{\boldsymbol{x}}\left(R_{ps} - 1\right) = \dot{x} = \frac{1}{R_{ps}}. \quad (20)$$

The next question is do we process column-by-column[19] or row-by-row[20]? To minimise latency a column-by-column approach would be preferable since the channel estimate for older OFDM symbols in the block would be available as soon as their respective column is processed. The drawback of using (19) is that the accumulation happens along the row, so the accumulation value of individual rows needs to be preserved until the very last element in the row is processed. Column-by-column processing would require us to preserve $K$ number of ACC states.[21] The usual register file holds just a handful of ACC registers, making it highly impractical for (19) to be processed column-by-column without driving up the HW cost[22] or lowering throughput,[23] both of which we are trying to avoid. Equation (17) does not have this limitation, and can be processed column-by-column to minimise latency. If latency is not an issue we can continue using (19) to optimise the total operation count per data item, and if latency is an issue we need to use (17) to meet the deadline and avoid overprovisioning our devices.

We call (19) the high throughput variant, and (17) the low latency variant. Equation (19) requires row-by-row processing and therefore latency can be an issue despite its low operation count per data item. Adversely, (17) has lower latency due to its column-by-column processing even though it has more operations per data item. When there is only one column to process (17) and (19) have identical operational complexity and latency. It is worth noting that the throughput gain or latency reduction of one algorithm variant over the other scales linearly with the number of columns to process and number of data items per column. These two equations showcase the throughput-latency trade-off that exists when implementing linear interpolation.

### C. PSEUDO CODE AND LATENCY SCALING

The high throughput pseudo code can be seen in Alg. 1. The low latency pseudo code can be seen in Alg. 2. Algorithm 2 can be expanded with an additional parameterised loop to scale and trade-off latency with memory accesses and throughput on a fine scale by processing additional data items along the time axis, as seen in Alg. 3. SW switch $T$ determines the number of extra data items processed along the time axis and if set to non-zero value, the code block would reduce the number of memory accesses[24] and *sub*s $T\times$, and increase the latency by the same factor. If fine tuning is not needed or the least latency is required, the go-to solution would remain Alg. 2, due to a lower loop control overhead. We show measurements of Alg. 1 and Alg. 2 in Section VI.

19. Loop over $k$ is the nested loop, loop over $l$ is the outer loop.
20. Loop over $l$ is the nested loop, loop over $k$ is the outer loop.
21. Up to a total of 8.4k ACC register states for FR2 high-end.
22. Via additional ACC registers or memory interfaces.
23. ACC register contents can be spilt to memory, but the processor stalls induced by the congestion of the load/store unit and additional *cast* operations needed would make the algorithmic gains on throughput pointless.
24. Loads from memory in the particular case.

**Algorithm 1:** RR Block High Throughput Variant (19)

**input:** Known columns $\hat{\boldsymbol{H}}(:,0)$, $\hat{\boldsymbol{H}}(:,R_{ps})$ and LUT $\dot{x}$
**output:** Interpolated block $\hat{\boldsymbol{H}}(:,:)$
// Data registers $v_1, v_2$
// ACC register $w_1$

**for** $k \leftarrow 0$ **to** $N-1$ **do**
  $v_1 = sub(\hat{\boldsymbol{H}}(k,R_{ps}),\hat{\boldsymbol{H}}(k,0))$;
  $w_1 = cast(\hat{\boldsymbol{H}}(k,0))$;
  **for** $l \leftarrow 1$ **to** $R_{ps}-1$ **do**
    $w_1 = mac(w_1, v_1, \dot{x})$;
    $v_2 = cast(w_1)$;
    $\hat{\boldsymbol{H}}(k,l) = v_2$;
  **end**
**end**

**Algorithm 2:** RR Block Low Latency Variant (17)

**input:** Known columns $\hat{\boldsymbol{H}}(:,0)$, $\hat{\boldsymbol{H}}(:,R_{ps})$ and LUT $\boldsymbol{x}$
**output:** Interpolated block $\hat{\boldsymbol{H}}(:,:)$
// Data registers $v_1, v_2$
// ACC register $w_1$

**for** $l \leftarrow 1$ **to** $R_{ps}-1$ **do**
  **for** $k \leftarrow 0$ **to** $N-1$ **do**
    $v_1 = sub(\hat{\boldsymbol{H}}(k,R_{ps}),\hat{\boldsymbol{H}}(k,0))$;
    $w_1 = mpy(v_1, \boldsymbol{x}(l))$;
    $v_2 = cast(w_1)$;
    $v_2 = add(v_2, \hat{\boldsymbol{H}}(k,0))$;
    $\hat{\boldsymbol{H}}(k,l) = v_2$;
  **end**
**end**

**Algorithm 3:** RR Block (17) Parameterised

**input:** Known columns $\hat{\boldsymbol{H}}(:,0)$, $\hat{\boldsymbol{H}}(:,R_{ps})$ and LUT $\boldsymbol{x}$
**output:** Interpolated block $\hat{\boldsymbol{H}}(:,:)$
// Data registers $v_1, v_2$
// ACC register $w_1$

**for** $l \leftarrow 1$ **to** $R_{ps}-1$ **do**
  **for** $k \leftarrow 0$ **to** $N-1$ **do**
    $v_1 = sub(\hat{\boldsymbol{H}}(k,R_{ps}),\hat{\boldsymbol{H}}(k,0))$;
    **for** $t \leftarrow 0$ **to** $T$ **do**
      $w_1 = mpy(v_1, \boldsymbol{x}(l+t))$;
      $v_2 = cast(w_1)$;
      $v_2 = add(v_2, \hat{\boldsymbol{H}}(k,0))$;
      $\hat{\boldsymbol{H}}(k,l+t) = v_2$;
    **end**
  **end**
  l += $T$;
**end**

### D. IMPLEMENTATION PLATFORM

The implementation platform of choice is a programmable 512-bit SIMD style vDSP with a VLIW architecture, designed using the ASIP Designer tool suite [37]. The tools come equipped with a C-compiler and debugging/profiling environment to provide accurate cycle measurements. The vDSP is designed to support complex fixed point precision arithmetic with 16-complex-bit (16-bit real, 16-bit imaginary) resolution per data item and 40-complex-bit ACC, meaning the vDSP can operate on 16×16-complex-bit data items per issued instruction. On a general vDSP like the one we used in our implementation, data types, i.e., resolution per data item, are configurable as well, however our decision to use 16-complex-bit resolution is based on the previous study [20] which concludes that 16-complex-bit resolution is sufficient to satisfy the 3GPP *error vector magnitude* requirements for transferred data in another *physical layer* kernel. The device supports two levels of parallelism: SIMD style data level parallelism for *vector processing* (VP) and multiple VLIW issue slots that provide instruction level parallelism. VLIW instruction parallelism enables performing several vector operations in parallel, using specific issue slots to access memory and others to perform scalar or vector operations. Each VLIW issue slot has a specific set of *functional units* associated with it, and effectively enables completion of kernel processing in fewer cycles compared to a device with a single issue slot. More on the VLIW configuration in Section VII.

### E. VECTOR PROCESSING

As mentioned earlier, a vectorisable algorithm can be written as a loop with the data processed per loop iteration being independent of the data in other loop iterations. The RR processing step operates on data structured in a matrix pattern of REs with two axes (time and frequency), which is favourable in terms of VP since it allows another loop to be nested in the existing loop structure as we have seen in the pseudo codes of Section V-C. The more loops the designer has to work with the more flexible the implementation gets.

The biggest trick is to decide and pick the correct loop to vectorise and in case the loops are mutually independent which nesting order of loops to use. This will impact your data reading and storing pattern and number of vector operations needed, which in turn has an effect on the latency and throughput of the kernel, and to an extent power consumption through number of vector operations and number of memory accesses [19], [20]. In Section V-C we have seen the effect of loop ordering on latency, throughput, and memory accesses. SIMD style vectorisation exaggerates this effect by working with bigger data batches per instruction.

In RR the two main options for vectorisation are vectorising along the time or the frequency axis. In Fig. 8 we show the vectorisation options for the RR stage, with RE for interpolation pattern shaded. Since the data is incoming per OFDM symbol column-wise in any transmission, processing a data vector across the time axis would require to wait for several OFDM symbols before fully loading the vector for processing. This makes the time axis unfavourable for long vector machines since it would introduce excess latency that is already critical. For example, a 16 data item vector, like the one we use, would load data from two different TTIs, which breaks the combined CE latency budget. Coarsely, a short vector machine would not have these latency limitations as

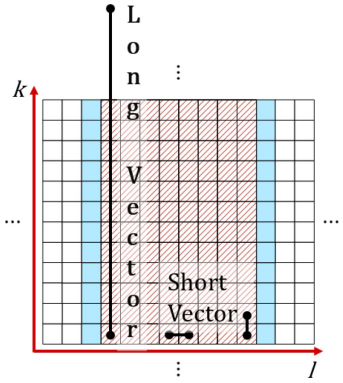


**FIGURE 8.** Options for Vectorising the Row Reconstruction Stage. Illustrated on a Resource Block: Short and Long Vector Machines.

**TABLE 2.** Normalised processing load and delay.

| Implementation | Processing Load | | Delay |
|---|---|---|---|
| | $\left[\frac{cycles}{RB \cdot OFDM}\right]$ | $\left[\frac{cycles}{data\ vector}\right]$ | $\left[\frac{cycles}{RB \cdot OFDM}\right]$ |
| Algorithm 1 (19) | 1.52 | 1.9 | $1.52(R_{ps}-1)$ |
| Algorithm 2 (17) | 2.04 | 2.55 | 2.04 |

long as the number of data items loaded per vector is smaller than the $R_{ps}$ spacing between known columns, giving it more flexibility, at a cost of lower throughput per SIMD operation compared to long vector machines. This flexibility can be used to potentially explore a different algorithm variant, but since we are also constrained by the use-case throughput requirement, we opt to use the long vector machine and vectorise along the frequency axis.

## VI. RESULTS

In this section we show the measurements of the implemented high throughput and low latency algorithm variants.

### A. CYCLE MEASUREMENTS

Table 1 holds the normalised processing load and delay in cycles per 12 data points or RB OFDM symbol processed.[25] We define processing load as the number of cycles needed to process a set of data items. We define delay as the number of cycles it takes a set of data items to pass through the kernel. It follows that scaling these numbers with the amount of RBs in a transmission gives us the number of cycles needed and corresponding delay per OFDM symbol of the particular use case. Scaling further with clock frequency gives us their processing time and delay towards the total size of the latency budget. For the purpose of measuring implementation efficiency we also show the processing load in cycles per 16 data points or vector length worth of data.

We see that Alg. 1 per OFDM symbol basis takes fewer cycles to interpolate compared to Alg. 2, primarily due to one less operation in the inner loop, however latency of Alg. 1 scales with the number of symbols to process. Alg. 2 is not so cycle efficient, but its delay is fixed and smaller than that of

25. One column of a RB, i.e., 12 subcarriers $\times$ 1 OFDM symbol.

**TABLE 3.** High throughput and low latency high-end FR2 vDSP implementation.

| Use Case | Algorithm 1 | | Algorithm 2 | |
|---|---|---|---|---|
| | proc.load | delay | proc.load | delay |
| | $[cycles]$ | $[cycles]$ | $[cycles]$ | $[cycles]$ |
| High-end FR2 | $49.8k$ | $49.8k$ | $66.8k$ | $8.4k$ |

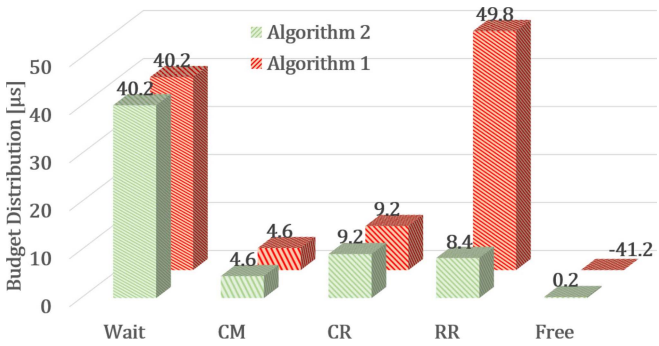


**FIGURE 9.** CE Latency Distribution Assuming an Allocated 1 GHz for the FR2 High-End.

Alg. 1 as long as there are at least 2 columns to interpolate, which is always the case as per active specifications.

When we take into the account the use case data loads, we get the values shown in Table 3. Through implementing a different variant of the same algorithm we have manged to reduce the latency by a noteworthy 83%, irrespective of the allocated frequency.

Finally, putting these numbers into perspective of the whole CE kernel, as shown in Fig. 9, we can see that the targeted FR2 high-end use case becomes possible also for the FR2 high-end on the same machine configuration, without the need for additional HW resources.

To summarise, there exists a trade-off between throughput and latency, which can be used to the designer's advantage. It is favourable to process across the frequency axis as fast as possible to reduce latency of CE for an individual OFDM symbol, likewise it is favourable to process across the time axis as fast as possible to increase throughput, when latency is not critical.

With the end of Moore's law, rising data rates and shorter deadlines of 6G and beyond, we can expect that these algorithmic optimisations become ever more impactful on communication systems as a whole.

## VII. VALIDATION

Often the question when optimising a piece of SW is when is the code "good enough" and you should stop optimising it. Sometimes, comparison with other implementations, e.g., implementations of *channel estimation* for FR2 high-end, are impossible since these are developed within a closed setting of R&D departments of private companies. Other times the work is pioneering or is within a niche and hence there is a literature gap. In all these cases a good measure of efficiency is measuring how well is your processor utilised. For a given vDSP, the SW implementation is well optimised if you are close in cycle counts to one of the various bounds of the

vDSP. For example, the vDSP has a memory bound defined by the maximum number of memory accesses that it can perform per cycle. An algorithm implementation is memory bound if it fully utilises this memory access capacity of the vDSP. Another vDSP bound would be the scheduling bound for functional units on different VLIW issue slots. This bound counts how many cycles you would need if you could have a perfect schedule on the disassembly level and it can be estimated by scheduling the assembly level code "by hand" for a few loop iterations. There are other bounds too, but they are practically hard to calculate, like the compiler bound - which requires you to know your compiler in depth. Assuming that the correct vectorisation axis and loop order were selected and there is no further optimisation possible on the algorithm level, one could consider calculating the memory interface bound and/or the VLIW scheduling bound.

Our vDSP has a load unit and a load-store unit available at different VLIW issue slots, meaning we can do a load + store or load + load per cycle. The algorithms need 2 loads and 1 store per output data item. If we were bounded by the memory interface *functional units* only we would need at least 3 cycles to process 2 output data vectors, i.e., we could expect 1.5 cycles between two vector stores. Similarly, if we were to try and schedule the algorithm on the appropriate VLIW slots by hand we would get around 1.8 cycles between two vector stores, a number lower than the memory bound due to the finite availability of computational *functional units*. If we compare now these values with Table 2, we can see that we are approaching 95% of the VLIW bound for Alg. 1 code and 71% of the VLIW bound for Alg. 2, which we deem sufficiently optimised.

## VIII. CONCLUSION

Both the SIMD parallelism and instruction-level parallelism of a VLIW-style vDSP can be effectively utilised for CE. As we have seen, there is usually some trade-off involved when implementing an algorithm. If you optimise to have the highest throughput, i.e., kernel complete in least amount of processor clock ticks, it may turn out that this solution has a large number of memory reads/writes or that it breaks the associated kernel deadline, and if you want to reduce latency you would possibly pay it with more operations. That said, if done right, the vDSP is a powerful platform to provide the required flexibility with high-performance implementations of algorithms that allow data parallelism to be exploited.

## ACKNOWLEDGMENT

The authors would like to thank Dr. Yankin Tanurhan and the Synopsys team for their guidance, support, sponsorship and initiation of the Industry - University cooperation within the "Efficient Implementation of 5G Baseband Kernels on a Vector Processor" project.

## REFERENCES

[1] J.-J. van de Beek, O. Edfors, M. Sandell, S. K. Wilson, and P. O. Borjesson, "On channel estimation in OFDM systems," in *Proc. IEEE 45th Veh. Technol. Conf. Countdown Wireless 21st Century*, vol. 2, 1995, pp. 815–819.

[2] S. Coleri, M. Ergen, A. Puri, and A. Bahai, "Channel estimation techniques based on pilot arrangement in OFDM systems," *IEEE Trans. Broadcast.*, vol. 48, no. 3, pp. 223–229, Sep. 2002.

[3] Y. Shen and E. C. Martínez. (2006). *Freescale Semiconductor Application Note AN 3059 Rev.0,1/2006 Channel Estimation in OFDM Systems*. [Online]. Available: https://pdfs.semanticscholar.org/f3fa/327521f2dcbeb87d81a17b8bcc216d108cbb.pdf

[4] C. Studer, A. Burg, and H. Bolcskei, "Soft-output sphere decoding: Algorithms and VLSI implementation," *IEEE J. Sel. Areas Commun.*, vol. 26, no. 2, pp. 290–300, Feb. 2008.

[5] P. Maechler, P. Greisen, B. Sporrer, S. Steiner, N. Felber, and A. Burg, "Implementation of greedy algorithms for LTE sparse channel estimation," in *Proc. Conf. Rec. 44th Asilomar Conf. Signals Syst. Comput.*, Nov. 2010, pp. 400–405.

[6] R. Ferdian, Y. Hou, and M. Okada, "A low-complexity hardware implementation of compressed sensing-based channel estimation for ISDB-T system," *IEEE Trans. Broadcast.*, vol. 63, no. 1, pp. 92–102, Mar. 2017.

[7] F. Gomez-Cuba and A. J. Goldsmith, "Sparse mmWave OFDM channel estimation using compressed sensing," in *Proc. IEEE Int. Conf. Communications (ICC)*, 2019, pp. 1–7.

[8] M. Simko, D. Wu, C. Mehlfuehrer, J. Eilert, and D. Liu, "Implementation aspects of channel estimation for 3GPP LTE terminals," in *Proc. 17th Eur. Wireless Sustain. Wireless Technol.*, 2011, pp. 1–5.

[9] X. Tang, M.-S. Alouini, and A. J. Goldsmith, "Effect of channel estimation error on M-QAM BER performance in Rayleigh fading," *IEEE Trans. Commun.*, vol. 47, no. 12, pp. 1856–1864, Dec. 1999.

[10] *NR; Physical Channels and Modulation V15.6.0*, 3GPP Standard TS 38.211, 2020. [Online]. Available: https://bit.ly/2xO4Qrt

[11] K. van Berkel, F. Heinle, P. P. E. Meuwissen, K. Moerman, and M. Weiss, "Vector processing as an enabler for software-defined radio in handheld devices," *EURASIP J. Adv. Signal Process.*, vol. 2005, no. 16, pp. 2613–2625, Sep. 2005. [Online]. Available: https://doi.org/10.1155/ASP.2005.2613

[12] HiSilicon. (Jul. 2020). *Balong, Official Product Page*. [Online]. Available: http://www.hisilicon.com/en/Products/ProductList/Balong

[13] Qualcomm. (Jul. 2020). *Snapdragon X60, Official Product Page*. [Online]. Available: https://www.qualcomm.com/products/snapdragon-x60-5g-modem

[14] Samsung. (Jul. 2020). *Exynos 5123, Official Product Page*. [Online]. Available: https://www.samsung.com/semiconductor/minisite/exynos/products/modemrf/exynos-modem-5123/

[15] *NR; User Equipment (UE) Radio Transmission and Reception Range 1 Standalone V16.3.0*, 3GPP Standard TS 38.101-1, Apr. 2020. [Online]. Available: https://bit.ly/2LUtz5K

[16] *NR; User Equipment (UE) Radio Transmission and Reception Range 2 Standalone V16.3.1*, 3GPP Standard TS 38.101-2, Apr. 2020. [Online]. Available: https://bit.ly/3gnnevt

[17] *Study on Supporting NR From 52.6 GHz to 71 GHz*, 3GPP Standard RP-193259, Dec. 2019. [Online]. Available: https://bit.ly/2BAysxV

[18] *Study on Supporting NR From 52.6 GHz to 71 GHz V0.0.1*, 3GPP Standard TR 38.808, Jun. 2020. [Online]. Available: https://bit.ly/3itcwFO

[19] S. A. Damjancevic, E. Matus, D. Utyansky, P. van der Wolf, and G. P. Fettweis, "From challenges to hardware requirements for wireless communications reaching 6G," in *Multi-Processor System-on-Chip 2*, L. Andrade and F. Rousseau, Eds. London, U.K.: ISTE Ltd., 2020, ch. 1.

[20] S. A. Damjancevic, E. Matus, D. Utyansky, P. van der Wolf, and G. Fettweis, "Towards GFDM for handsets—Efficient and scalable implementation on a vector DSP," in *Proc. IEEE 90th Veh. Technol. Conf. (VTC-Fall)*, 2019, pp. 1–7.

[21] *NR; User Equipment (UE) Radio Transmission and Reception Range 2 Standalone V15.4.0*, 3GPP Standard TS 38.101-2, Jan. 2019. [Online]. Available: https://bit.ly/2SfjQIc

[22] F. Weng, C. Yin, and T. Luo, "Channel estimation for the downlink of 3GPP-LTE systems," in *Proc. 2nd IEEE Int. Conf. Netw. Infrastruct. Digit. Content*, 2010, pp. 1042–1046.

[23] H. Lime, "Cross-subframe channel estimation for low-complexity devices in LTE," M.S. thesis, KTH Royal Inst. Technol., Sch. Electr. Eng., Stockholm, Sweden, 2017.

[24] T. Bogale, X. Wang, and L. Le, "Chapter 9—mmWave communication enabling techniques for 5G wireless systems: A link level perspective," in *mmWave Massive MIMO*, S. Mumtaz, J. Rodriguez, and L. Dai, Eds. San Diego, CA, USA: Academic, 2017, pp. 195–225. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128044186000091
[25] *Study on Channel Model for Frequencies From 0.5 to 100 GHz V15.0.0*, 3GPP Standard TR 38.901, Jun. 2018. [Online]. Available: https://bit.ly/323kf4g
[26] C. Gustafson, K. Haneda, S. Wyne, and F. Tufvesson, "On mm-Wave multipath clustering and channel modeling," *IEEE Trans. Antennas Propag.*, vol. 62, no. 3, pp. 1445–1455, Mar. 2014.
[27] X. Song, S. Haghighatshoar, and G. Caire, "Efficient beam alignment for millimeter wave single-carrier systems with hybrid MIMO transceivers," *IEEE Trans. Wireless Commun.*, vol. 18, no. 3, pp. 1518–1533, Mar. 2019.
[28] M. R. Akdeniz *et al.*, "Millimeter wave channel modeling and cellular capacity evaluation," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 6, pp. 1164–1179, Jun. 2014.
[29] P. Schniter and A. Sayeed, "Channel estimation and precoder design for millimeter-wave communications: The sparse way," in *Proc. 48th Asilomar Conf. Signals Syst. Comput.*, 2014, pp. 273–277.
[30] T. S. Rappaport *et al.*, "Millimeter wave mobile communications for 5G cellular: It will work!" *IEEE Access*, vol. 1, pp. 335–349, 2013.
[31] M. F. Duarte and Y. C. Eldar, "Structured compressed sensing: From theory to applications," *IEEE Trans. Signal Process.*, vol. 59, no. 9, pp. 4053–4085, Sep. 2011.
[32] J. Carreira and J. Li, *Optimizing Latency and Throughput Trade-Offs in a Stream Processing System*, Adv. Topics Comput. Syst. Course, Univ. California, Berkeley, CA, USA, 2014. [Online]. Available: https://people.eecs.berkeley.edu/kubitron/courses/cs262a-F14/projects/reports/project11_report_ver3.pdf
[33] I. Pantazi-Mytarelli, "The history and use of pipelining computer architecture: MIPS pipelining implementation," in *Proc. IEEE Long Island Syst. Appl. Technol. Conf. (LISAT)*, 2013, pp. 1–7.
[34] X. Tian, R. Han, L. Wang, G. Lu, and J. Zhan, "Latency critical big data computing in finance," *J. Finan. Data Sci.*, vol. 1, no. 1, pp. 33–41, 2015.
[35] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, "Less is more: Trading a little bandwidth for ultra-low latency in the data center," in *Proc. 9th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2012, pp. 253–266.
[36] E. Nilsson, "Exploring trade-offs between latency and throughput in the nostrum network on chip," Ph.D. dissertation, KTH Royal Inst. Technol., Sch. Inf. Commun. Technol., Stockholm, Sweden, 2006.
[37] Synopsys, Inc. *ASIP Designer Website*. Accessed: Aug. 9, 2020. [Online]. Available: https://www.synopsys.com/asip

**STEFAN A. DAMJANCEVIC** (Graduate Student Member, IEEE) received the Dipl.-Ing. degree in electrical engineering form the University of Novi Sad and the M.Sc. degree in electrical engineering form TU Dresden. He is currently pursuing the Ph.D. degree under the supervision of Prof. G. P. Fettweis. He is a Junior Scientist with the Signal Processing Hardware Group, Vodafone Chair Mobile Communications Systems, TU Dresden. He is an alumni of the German Academic Exchange Service and the Gustav Stresemann Institute. His research is on the implementation aspects of 5G and beyond physical layer algorithms and modem architectures.

**EMIL MATUS** received the M.S. and Ph.D. degrees in electrical engineering from TU Kosice, Slovakia, where he focused on the wavelet transform and image compression research. He has been a Senior Scientist with Vodafone Chair for Mobile Communications Systems, TU Dresden since 2003, where he is leading Signal Processing Hardware Research Group. His current research interests include wireless physical layer algorithms and communications signal processing hardware architectures. His group introduced several generations of the signal-processing MPSoC architecture called "Tomahawk," which received the DAC/ISSCC Student Design Award in 2009, the CoolSilicon Cool Award in 2014, and the HIPEAC Award in 2017.

**DMITRY UTYANSKY** received the M.S. degree in computer science from State Electrotechnical University "LETI," Saint Petersburg, Russia, in 1994. He is a Senior Staff Software Engineer with Synopsys Corporation, St. Petersburg, where he is engaged in digital signal processing algorithms exploration and design and benchmarking of Synopsys DSP architectures. He was subsequently involved in various research and development projects in DSP, audio and image processing, and communications. His current interests include algorithms for linear algebra and its applications in RADARs and AI, and utilizing vector processors.

**PIETER VAN DER WOLF** received the M.Sc. and Ph.D. degrees in electrical engineering from the Delft University of Technology. He is a Principal Product Architect with Synopsys. He was an Associate Professor with the Delft University of Technology before joining Philips Research in 1996. In 2006, he joined NXP Semiconductors when it was spun out of Philips Electronics. In 2009, he joined Virage Logic, which was subsequently acquired by Synopsys. He has worked on a broad range of topics, including (multi-)processor architectures and system design methodologies, with a focus on DSP applications.

**GERHARD P. FETTWEIS** (Fellow, IEEE) received the Ph.D. degree under supervision of Prof. H. Meyr's from RWTH Aachen in 1990. He has been a Vodafone Chair Professor with TU Dresden since 1994, and has been heads the Barkhausen Institute since 2018. After one year with IBM Research, San Jose, CA, USA, he moved to TCSI Inc., Berkeley, CA, USA. He coordinates the 5G Lab Germany, and has coordinated two German Science Foundation (DFG) centers at TU Dresden, namely cfaed and HAEC. In Dresden, his team has spun-out seventeen start-ups, and setup funded projects in volume of close to EUR 1/2 billion. In 2019, he was elected into the DFG Senate. His research focusses on wireless transmission and chip design for wireless/IoT platforms, with 20 companies from Asia/Europe/U.S. sponsoring his research. He also serves on the board of National Instruments Corp, and advises other companies. He is a Co-Chair the IEEE 5G/Future Networks Initiative, and has helped organizing IEEE conferences, most notably as a TPC Chair of ICC 2009 and TTM 2012, and as a General Chair of VTC Spring 2013 and DATE 2014. He is a member of the German Academy of Sciences (Leopoldina), the German Academy of Engineering (acatech), and received multiple IEEE recognitions as well has the VDE ring of honor.