Hardware-Efficient Node Processing Unit Architectures for Flexible LDPC Decoder Implementations

Peter Hailes, Lei Xu, Robert G. Maunder, Bashir M. Al-Hashimi and Lajos Hanzo
School of ECS, University of Southampton, SO17 1BJ, UK
Corresponding author: lh@ecs.soton.ac.uk

***Abstract*—In LDPC decoder implementations, the architecture of the Node Processing Units (NPUs) has a significant impact both on the hardware resource requirements and on the processing throughput. Additionally, some NPU architectures impose limitations on the decoder's support for intra- or inter-standard LDPC code flexibility at run-time. In this paper, we present a generalised algorithmic method of constructing NPUs that support run-time flexibility whilst maintaining a low hardware resource requirement and high maximum operating frequency. FPGA-based synthesis results demonstrate that the proposed architecture offers a significantly improved hardware efficiency, when compared to two commonly-employed alternatives.**

***Index Terms*—Digital communication, error correction codes, low-density parity check (LDPC) codes, stochastic computing, iterative decoding**

I. Introduction

Low-Density Parity Check (LDPC) codes [1] constitute a class of linear Forward Error Correction (FEC) codes that exhibit error correction performance very close to the Shannon limit, whilst supporting a high degree of parallel processing. As a result, LDPC codes are increasingly being adopted in modern communications standards, including IEEE 802.11n [2], IEEE 802.16e [3], and eMBB data in the forthcoming 3GPP 5G New Radio (NR) [4].

As described in [5], an LDPC code is defined by a sparse $M \times N$ *Parity-Check Matrix* (PCM) $\mathbf{H}$, which may also be depicted graphically by a bipartite factor graph. Here, edges connect *Check Nodes* (CNs) and *Variable Nodes* (VNs), as dictated by the positions of the non-zero elements of $\mathbf{H}$. The number of edges d_c connected to each CN and the number d_v connected to each VN is known as that node's *degree*. LDPC codes may be decoded by iteratively generating and exchanging messages between the VNs and CNs, via the edges of the factor graph. The specific operation of each node, the format of the messages, and the order in which they are exchanged, can vary between differing decoding algorithms and schedules.

In hardware LDPC decoder implementations, the functions of the nodes are performed by Node Processing Units (NPUs). The internal construction of an NPU may be thought of as a logical structure comprising one or more 2-input 1-output subnode blocks [6], [7]. These subnodes perform the specific calculations of the NPU, and so their design can vary between different LDPC decoding algorithms. In this paper, we consider the arrangement of subnodes within an NPU, in a manner that is independent of the internal subnode operation.

In a *fully-parallel* decoder architecture [8], each CN and VN is implemented by a separate hardware NPU. However, fully-parallel decoders are rarely practical due to the overwhelming routing complexity of the random interconnections between the nodes, and because they support only a single PCM, giving no flexibility over the block length or coding rate [9]. Instead, *partially-parallel* decoder architectures [8] employ a lower number of parallel NPUs, which are time-multiplexed between different VNs and CNs, using internal memories to store intermediate calculation results. This approach benefits from reduced routing complexity and the possibility to support multiple PCMs, whilst also offering a flexible trade-off between hardware resource usage and processing throughput. However, partially-parallel decoders require each Check Node Processing Unit (CNPU) and Variable Node Processing Unit (VNPU) to be able to perform the function of multiple different CNs and VNs, respectively. In irregular LDPC code decoders, or in flexible decoders that support multiple PCMs having different degree distributions, this requires the NPUs to have the ability to process varying numbers of inputs and outputs. This additional requirement for run-time flexibility can drastically complicate the design of an NPU, increasing its hardware resource requirements and/or critical path length.

Against this background, in this brief we present a hardware-efficient flexible NPU architecture, with general applicability to many different NPU types and LDPC decoding algorithms. In Section II, we provide a discussion of other NPU architectures, and their suitability for flexible LDPC decoder implementations. The proposed algorithm and the resultant architecture are then presented in Section III, followed by implementation characteristics and concluding remarks in Sections IV and V, respectively.

II. Flexible NPU architectures

NPUs each have a number I of inputs and outputs, which respectively accept and provide the messages that are exchanged via the connected edges of the factor graph. CNPUs

The financial support of the PhD studentship provided by Altera, California USA, the grants EP/J015520/1 and EP/L010550/1 provided by EPSRC, Swindon UK, the grant TS/L009390/1 provided by Innovate UK, Swindon UK, as well as the Advanced Fellow grant provided by the European Research Council is gratefully acknowledged. The research data for this paper is available at http://doi.org/10.5258/SOTON/D0311.

performing the function of degree-d_c CNs feature $I = d_c$ number of inputs and outputs, where the message calculated for a particular output depends on the messages received at all inputs other than the corresponding input. VNPUs for degree-d_v VNs function similarly, except that they have $I = d_v + 1$ inputs and outputs. Here, the extra input corresponds to the intrinsic channel information, while the extra output provides *a posteriori* information used to make a decision about the corresponding LDPC encoded bit. Again, each of the outputs depends on all inputs other than the corresponding input, with the exception of the *a posteriori* output which depends on all inputs, including the corresponding intrinsic input [5]. As mentioned in Section I, flexible NPUs must have the ability to represent multiple different VNs or CNs at run-time, varying the number of inputs and outputs used according to the degree of the current VN or CN being represented.

Some previously published *block-parallel* LDPC decoders [10], [11] have solved this problem by employing serial NPUs, comprising only a single subnode. Here, each node having i inputs is processed using $i - 1$ clock cycles. Whilst this may be very efficient in terms of hardware resource usage, it results in very low processing throughputs, which are not generally capable of meeting the requirements of modern communications standards. Alternatively, parallel flexible NPUs may be implemented having the maximum number of inputs and outputs, namely $I = D_C$ for CNPUs or $I = D_V + 1$ for VNPUs, where D_C and D_V refer to the maximum values of d_c and d_v within the set of supported PCMs, respectively. When used for processing nodes having lower degrees, the unused inputs may be effectively disabled by supplying them with a "null" value [12]–[14]. For the example of VNPUs processing messages in the form of binary fixed-point Logarithmic-Likelihood Ratios (LLRs), each subnode performs the addition of two LLRs. Here, providing any input with a 0-valued LLR will nullify any effect it may have on the proceeding calculations. However, a null value does not always exist, as is the case for the NPUs of stochastic LDPC decoders [6], [15], in which the messages exchanged between nodes are represented by bit-serial Bernoulli sequences [6]. Furthermore, the use of null values results in inefficient energy usage, since this implies that every subnode is activated during every NPU activation, regardless of whether it makes any contribution towards the decoding process. Instead, a more efficient solution would be to use multiplexers to flexibly replace the outputs of subnodes having unused inputs with values from earlier active inputs. Here, power- or clock-gating could be applied to any unused subnodes, effectively reducing their energy consumption.

Many previous LDPC decoder implementations have presented NPU architectures that are compatible with null inputs. Of these, the most efficient in terms of both hardware usage and propagation delay is to use a binary tree structure of subnodes to compute the combination of all I inputs, then perform the inverse subnode operation for each output in order to remove the contribution made by the corresponding input [16]. This architecture requires $N_{sub} = 2 \times I - 1$ subnodes and has a critical path of $D = \lceil \log_2(I) \rceil + 1$ subnode delays, but is only possible when the subnode operation is invertible. Alternative architectures must therefore be employed when the subnode operation is not invertible, as is the case with the $\min$ operation of CNPUs performing the Min-Sum Algorithm (MSA), for example [14]. One rudimentary option, henceforth referred to as a *multi-tree* architecture, is to use I parallel binary trees to calculate the combination of each of the I possible sets of $I - 1$ inputs [12]. This option provides the shortest possible propagation delay of $D = \lceil \log_2(I - 1) \rceil$ subnode delays. In a naïve implementation, the hardware requirement of this architecture is $N_{sub} = I \times (I-2)$ subnodes, although it is possible to reduce this by reusing early subnode outputs between trees, as described in [16]. As an alternative, the *forwards-backwards* architecture [17] maximally reuses subnode outputs in order to minimise the required number of subnodes to $N_{sub} = 3 \times I - 6$ [7]. However, the nature of the forwards-backwards algorithm results in a long propagation delay of $D = I - 2$ subnode delays, limiting the maximum operating frequency of practical decoder implementations.

A compromise between these two options is the *dual-tree* architecture of [6], [7], [9], however this has not previously been generalised to support an arbitrary and run-time flexible number I of inputs and outputs. In Section III we present a generalised algorithmic method for constructing dual-tree NPUs. We also propose an algorithmic method for placing multiplexers, in order to provide flexibility without the use of null inputs. This allows energy savings using clock- or power-gating, as well as flexibility when null inputs are not supported, such as in stochastic LDPC decoders. We also propose a method of reducing the number of multiplexers required, depending on the numbers of inputs that are required by a particular supported set of one or more LDPC PCMs. This results in an NPU architecture that supports flexibility for any number of inputs and outputs, regardless of the function performed by the internal subnodes, and without requiring excessive hardware resources or propagation delay.

III. The dual-tree NPU architecture

In this section, we propose the generalised algorithm of Fig. 1 for creating an NPU having the proposed flexible dual-tree structure. Fig. 2 then exemplifies an NPU having the proposed structure for the special case where the maximum number of inputs I is a power of 2, namely $I = 16$. Later, Fig. 3 will exemplify the general case, where I is not a power of 2.

In Fig. 2, each shaded square represents a single 2-input 1-output subnode, while the NPU inputs in_0 to in_{15} are depicted on the left, with corresponding outputs out_0 to out_{15} vertically flipped on the right. When the number of active inputs and outputs i is less than I, data is provided to only the top i inputs, in_0 to in_{i-1}, and the corresponding results are presented on the bottom i outputs, out_0 to out_{i-1}. Note that Fig. 2 represents a $D_V = 15$ VNPU, with channel input in_0 and decision output out_0. More specifically, unlike out_1 to out_{15}, out_0 is a function of all i inputs, which is achieved by providing the intermediate signal T_1^0 to the bottom subnode in the right-most stage. Alternatively, the structure in Fig. 2 could represent a CNPU with a maximum degree of $D_C = 16$,

Figure 1. Construction of flexible dual-tree NPU

procedure CONSTRUCT_NPU(I, $\mathbf{i}_{\text{sup}}$)
  $S \coloneqq \lceil \log_2(I) \rceil - 1$ ▷ *no. of stages per section*
  $T_0 \coloneqq$ inputs ▷ T_s^t = *signal* t *at stage* s
Summing section
  for $s = 1 : S$ **do**
    for $t = 0 : \lceil \frac{I}{2^s} \rceil$ **do**
      $L \coloneqq T_{s-1}^{2t}$
      $R \coloneqq T_{s-1}^{2t+1}$
      if R exists **then**
        $c \coloneqq \text{SUBNODE}(L, \quad R)$
        $r \coloneqq t \times 2^s$ ▷ *relevance*
        $b \coloneqq r + 2^{s-1}$ ▷ *bypass*
        if $(t \geq 1)$ **and**
         (any $i \in \mathbf{i}_{\text{sup}}$, $r < i \leq b$ exists) **then**
          $T_s^t \coloneqq \text{MUX}(b, \quad c \mid L)$
        else
          $T_s^t \coloneqq c$ ▷ *no mux after subnode*
        end if
      else
        $T_s^t \coloneqq L$ ▷ *no subnode*
      end if
    end for
  end for
Combining section
  for $s = S + 1 : 2S$ **do**
    for $t = 0 : \lceil \frac{I}{2^{2S-s}} \rceil$ **do**
      if $s > S + 1$ **then** $L \coloneqq T_{s-1}^{\lfloor \frac{t}{2} \rfloor}$
      else $L \coloneqq T_{s-1}^{1+\lfloor \frac{-t}{2} \rfloor}$ ▷ *1st stage flips inputs*
      $R \coloneqq T_{2S-s}^{t+(-1)^t}$
      if R exists **then**
        $c \coloneqq \text{SUBNODE}(L, \quad R)$
        if $t = 0$ **then** $r \coloneqq 2^{2S-s}$
        else $r \coloneqq t \times 2^{2S-s}$
        $b \coloneqq r + 2^{2S-s}$
        if $(t \bmod 2 = 0$ **or** $t = 1)$ **and**
         (any $i \in \mathbf{i}_{\text{sup}}$, $r < i \leq b$ exists) **then**
          if $t \leq 1$ **then** $T_s^t \coloneqq \text{MUX}(b, \quad c \mid R)$
          else $T_s^t \coloneqq \text{MUX}(b, \quad c \mid L)$
        else
          $T_s^t \coloneqq c$ ▷ *no mux after subnode*
        end if
      else
        $T_s^t \coloneqq L$ ▷ *no subnode*
      end if
    end for
  end for
  outputs $\coloneqq T_{2S}$
end procedure

but the second input to the bottom subnode in the right-most stage would be in_1 rather than T_1^0. This would result in out_0 becoming a function of only $i-1$ of the inputs, namely in_1 to in_{i-1}, as required.

The overall structure of the proposed architecture may be described as follows. The dual-tree topology is comprised of two main sections, *summing* and *combining*, each of which contains $S = \lceil \log_2(I) \rceil - 1$ stages. Let T_s^t represent the value of the t-th intermediate signal in the s-th stage, with $0 \leq s \leq 2S$. Furthermore, let N_s represent the total number

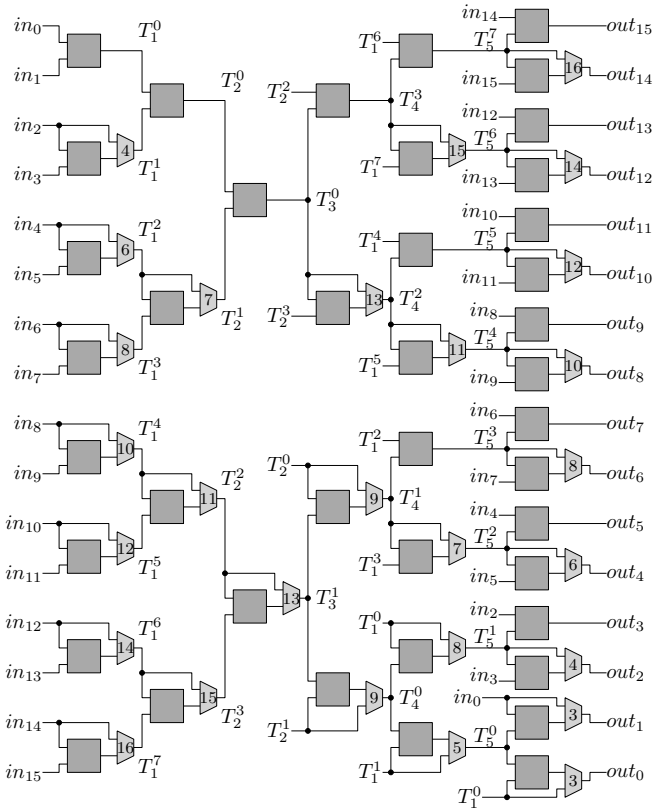

Figure 2. Dual-Tree VNPU with $I = 16$, exhibiting full flexibility for any number of inputs

of intermediate signals in stage s, with $0 \leq t < N_s$. Stage $s = 0$ comprises the set of NPU inputs, and so $N_0 = I$. The summing section functions similarly to a conventional binary tree, in which each intermediate signal T_s^t is calculated as the combination of two previous signals, namely T_{s-1}^{2t} and T_{s-1}^{2t+1}, giving $N_s = \lceil \frac{N_{s-1}}{2} \rceil$. This continues until only $N_S = 2$ signals remain, T_S^0 and T_S^1, each of which contains the combination of a different set of up to $2^{\lceil \log_2 I \rceil - 1}$ inputs.

Subsequently, in each combining section stage s, $S+1 \leq s \leq 2S$, each intermediate signal T_{s-1}^t is connected to two subnodes and combined with two earlier signals calculated during the summing section. More specifically, each combining section signal T_s^t is the combination of one signal from stage $s-1$ and one from stage $2S-s$. The number of intermediate signals N_s therefore doubles in each successive stage during the combining section, $N_s = 2N_{s-1}$. It can be readily seen that this structure facilitates an increasingly specific combination of signals, where each output contains the combination of $I-1$ inputs as required.

As mentioned previously, the flexibility of the proposed architecture is facilitated by multiplexers placed at the outputs of key subnodes throughout the NPU. More specifically, during the summing section, multiplexers are placed at each intermediate signal T_s^t where $t > 0$; during the combining section, they are placed at each signal where t is even, and additionally at signals where $t = 1$. These *bypass multiplexers* optionally allow the corresponding intermediate signal to replicate a value from an earlier connected stage, rather than using the

subnode output. This option is exercised when the subnode result would otherwise depend upon an unused NPU input. More specifically, each multiplexer has a *bypass threshold* b, and is used for bypassing the corresponding subnode when the number of active inputs and outputs satisfies $i \leq b$. The bypass threshold for each multiplexer is indicated in Fig. 2, and is calculated according to

$$b = \begin{cases} t \times 2^s + 2^{s-1}, & \textit{summing} \\ (t+1) \times 2^{2S-s}, & \textit{combining and } t \geq 1 \\ 2 \times (t+1) \times 2^{2S-s} & \textit{combining and } t = 0 \end{cases}. \quad (1)$$

In this way, the proposed NPU architecture offers full flexibility for any number of inputs $i \leq I$, by using only an additional $2 \times (I-2)$ multiplexers.

The algorithm of Fig. 1 can also extend the architecture of Fig. 2 to less regular cases having any number of inputs I, not only those that are a power of 2. In this generalised case, the total number of subnodes required by the proposed architecture is $N_{sub} = 3 \times I - 6$, which is identical to that of the forwards-backwards architecture [17]. However, the proposed architecture has a shorter propagation delay, whilst also facilitating the flexibility mentioned previously. More specifically, the propagation delay of the forwards-backwards architecture increases linearly with $D = I - 2$, whereas for the dual-tree architecture it is logarithmic, according to

$$D = \max(2,\ 2 \times \lfloor \log_2(I-1) \rfloor + \lfloor \frac{I-1}{2^{\lfloor \log_2(I-1) \rfloor - 1}} \rfloor - 3) \quad (2)$$

$$\approx 2 \times \lfloor \log_2(I-1) \rfloor. \quad (3)$$

Furthermore, motivated by the observation that full flexibility is not required for most applications, the algorithm of Fig. 1 can also optimise the architecture of Fig. 2 to only include the multiplexers that are actually required to support a reduced set of NPU degrees. For example, Fig. 3 depicts a VNPU which can provide the functionality of any variable node within the 12 PCMs of the IEEE 802.11n LDPC code [2]. More specifically, this $I = 13$ VNPU supports any number of inputs and outputs in the set $\mathbf{i}_{\text{sup}} \in \{3, 4, 5, 7, 8, 9, 12, 13\}$.

It may be observed that the NPU of Fig. 3 has a similar structure to that of Fig. 2, albeit with the removal of 9 subnodes across both summing and combining sections. Furthermore, this NPU supports a reduced level of run-time flexibility, demonstrated by the fact that $\mathbf{i}_{\text{sup}}$ contains only a subset of $\mathbf{i}_{\text{full}} \in \{0, 1, 2, ..., I\}$. Accordingly, Fig. 3 can omit the multiplexers at signals T_1^5, out_{10}, out_1, and out_0 in Fig. 2, in addition to those removed automatically alongside their corresponding subnodes. The set of multiplexers that can be removed in this way is identified using the *relevance* parameter r of Fig. 1. For each multiplexer, this parameter represents the minimum value of i for which the bypassed result provided by that multiplexer (i.e. when $i \leq b$) will be utilised in later stages. If $\mathbf{i}_{\text{sup}}$ does not include any values of i for which $r < i \leq b$, then the bypassed result will never be used, and hence the multiplexer is not required. In this case, the signal that would otherwise be provided by the multiplexer can be connected directly to the output of the corresponding subnode.

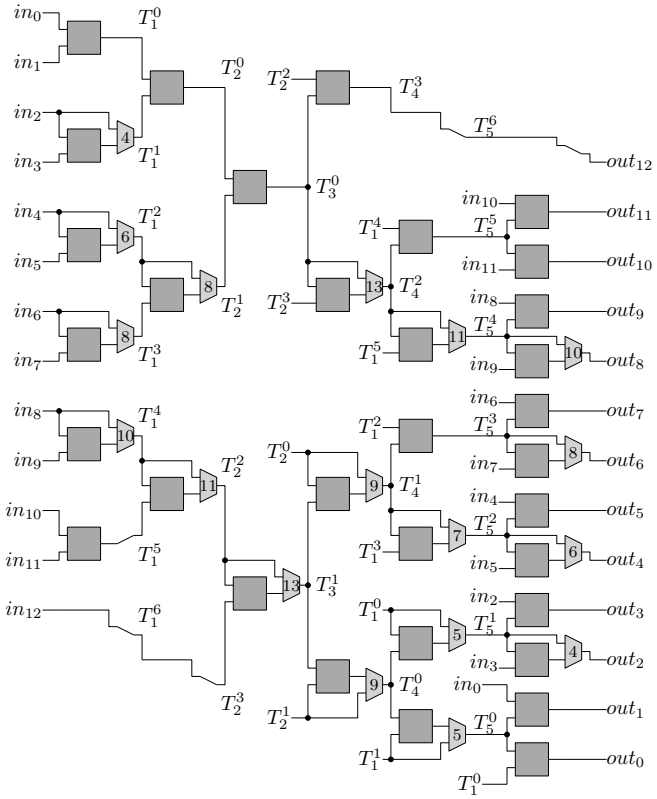

Figure 3. Dual-Tree VNPU with $I = 13$, exhibiting flexibility for any number of inputs required by IEEE 802.11n [2]

IV. Implementation results

The correct operation of the proposed dual-tree NPU architecture has been verified in isolation for a variety of different NPU types, in both bit-true C++ simulations and through HDL simulation in ModelSim. Furthermore, dual-tree NPUs have been integrated and validated in two full LDPC decoders, namely the flexible fixed-point architecture of [12], and a novel flexible stochastic architecture which will be the subject of a future publication.

As discussed in Section I, the proposed dual-tree architecture may be used to implement a variety of NPUs, as determined by the function of its 2-input 1-output subnodes. Accordingly, the specific implementation characteristics of NPUs having the proposed dual-tree architecture depend on the characteristics of these subnodes. In order to facilitate a comparison and discussion in this section, we consider NPUs having subnodes which calculate the $\min(x, y)$ of two 4-bit unsigned inputs x and y, as used within fixed-point CNPUs performing the MSA[1] [14]. Since this function is non-invertible, the single binary tree structure mentioned in Section I is not viable. Consequently, this motivates the use of one of the alternative general-purpose architectures discussed earlier, namely the proposed dual-tree, multi-tree [12] and

[1]Note that the full MSA calculation is $\text{sign}(\hat{x}) \times \text{sign}(\hat{y}) \times \min(|\hat{x}|, |\hat{y}|)$. However, the sign and modulus operations have significantly lower hardware requirements than the $\min$ computation, and may be calculated independently from it then combined at the end. Furthermore, this may be performed in a manner which is unaffected by the arrangement of subnodes discussed here. Accordingly, these aspects have been omitted from these results for clarity.

Table I
IMPLEMENTATION CHARACTERISTICS OF CNPUS HAVING THREE GENERAL-PURPOSE ARCHITECTURES, SUPPORTING IEEE 802.11N CNS

Architecture	ELBs	f_{max} (MHz)	f_{max} / ELBs
(Proposed) **Dual-tree**	408	137.85	0.338
FwdBwd	412	39.89	0.097
Multi-tree	884	134.01	0.152

forwards-backwards (FwdBwd) [17] architectures. For this specific CNPU, a further additional comparison may also be made with the architecture of [18] (denoted here as *Min2*), which computes the two minimum values of the full set of I inputs and assigns them to the required outputs [18], [19]. This is in contrast to the alternative general-purpose approaches, which calculate each output as the minimum of the corresponding set of $I-1$ inputs. Note that of these four architectures, only the proposed dual-tree architecture has inherent flexibility over the number of active inputs and outputs. However, in order to facilitate a fair comparison, the other architectures may be implemented to provide a "null" value of $+7$ on unused inputs to effectively turn those inputs off, as described in [12].

Fig. 4 presents the hardware requirements and maximum operating frequencies[2] (f_{max}) of CNPUs supporting a range of maximum numbers of inputs and outputs, I, where each node flexibly supports all numbers of inputs and outputs $i \leq I$, giving $\mathbf{i}_{\text{sup}} = \mathbf{i}_{\text{full}}$. More specifically, sample dual-tree, multi-tree and FwdBwd CNPUs have been generated and characterised, supporting $4 \leq I \leq 30$. Here, the hardware requirements and maximum operating frequencies are based on synthesis for an Altera Stratix IV EP4SGX530 FPGA, with hardware resource usage characterised using the Equivalent Logic Block (ELB) metric presented in [5]. Additionally, Fig. 4 also includes the characteristics of Min2 CNPUs for $I \in \{4, 8, 16, 32\}$, using results from [20]. Note that the original results in [20] do not consider the hardware required to provide null inputs, nor to assign the computed minimums to the corresponding outputs. Accordingly, the Min2 hardware characteristics presented in Fig. 4 include an additional cautious estimate of $6I$ ELBs to account for this. In addition to Fig. 4, Table I presents the measured characteristics of CNPUs adopting the three general-purpose architectures described above. These CNPUs have been designed to support any CN within IEEE 802.11n [2], having $I = D_C = 22$, and supporting each number of inputs and outputs in the set $\mathbf{i}_{\text{sup}} \in \{7, 8, 11, 14, 15, 19, 20, 21, 22\}$. Table I also presents the hardware efficiency of each CNPU, which is calculated as f_{max} / ELBs. Note that the Min2 architecture is only characterised in [20] for CNPUs having values of I which are a multiple of 2, and hence cannot be included in Table I.

[2]In order to provide a clear comparison of the characteristics of competing architectures, in this section we present the f_{max} of each NPU in isolation. Note however that the f_{max} of a complete LDPC decoder depends on the critical path length of the routing, NPUs, and memory accesses. When the LDPC decoder of [12] was adapted to use the proposed architecture, an f_{max} of 100.04 MHz was achieved, which resulted in a throughput of 550.5 Mbps for the case of the IEEE 802.11n LDPC code having rate $R = 0.75$ and frame length $N = 648$.

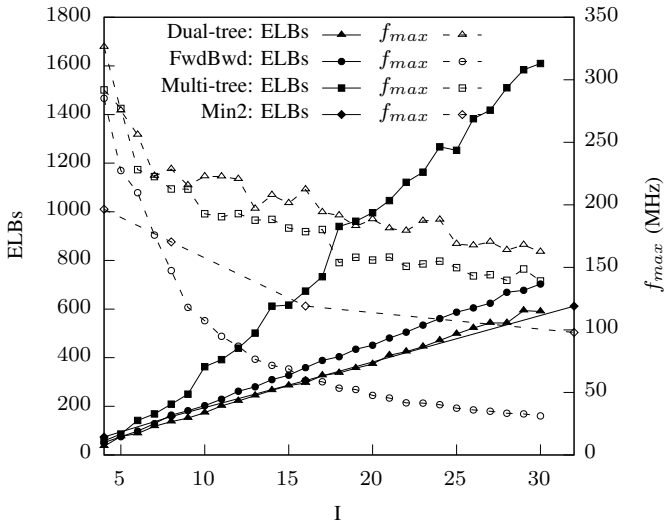

Figure 4. Comparison of fixed-point CNPUs constructed using Dual-Tree topology vs. three alternatives, for a range of input numbers I

As may be expected, the long critical path of the forwards-backwards architecture causes it to have a significantly lower f_{max} than the alternative architectures. However, Table I and Fig. 4 also indicate that its hardware resource requirement is slightly higher than that of the proposed dual-tree topology, despite requiring an equal number of subnodes. This may be explained by the distribution of the bypass multiplexers throughout the proposed dual-tree NPUs, allowing synthesis optimisations to combine their functions with unused inputs/outputs of the logic elements used by the subnodes. By contrast, the multiplexers that select between an input LLR and a null LLR in the forwards-backwards architecture cannot be similarly distributed and optimised.

The large number of subnodes required by the multi-tree architecture causes it to have a significantly higher hardware usage than the alternative architectures, as shown in Table I and Fig. 4. However, it can also be seen that the multi-tree architecture has a lower f_{max} than that of the proposed dual-tree architecture, despite having fewer subnode delays in its critical path. Again, this may be explained by the physical limitations of FPGA synthesis, in which a higher hardware resource usage creates longer routing paths, increasing the propagation delay.

Overall, it may be seen that the proposed dual-tree architecture offers a lower hardware resource requirement and a higher maximum operating frequency than both of the two general-purpose alternatives. This is demonstrated in its significantly higher hardware efficiency seen in Table I, which is 122% higher than that of the multi-tree architecture, and 248% higher than that of the forwards-backwards architecture. Additionally, by comparing the results in Table I and the points for $I = 22$ in Fig. 4, the efficacy of the bypass multiplexer reduction method discussed in Section III may be observed. More specifically, by supporting a set of inputs and outputs $\mathbf{i}_{\text{sup}}$ which is smaller than $\mathbf{i}_{\text{full}}$, the dual-tree architecture for $I = 22$ requires 17 fewer ELBs.

Finally, Fig. 4 also demonstrates that the hardware resource requirements of the dual-tree and Min2 architectures are

approximately equivalent, when utilised for flexible CNPUs. However, here the Min2 architecture suffers from a significantly longer critical path, due to the number of operations which must be performed after the global minimum of all I inputs has been found [18]. Consequently, it may be shown that the proposed dual-tree architecture has a 127% higher hardware efficiency than the Min2 architecture, when averaged across three sample cases for which data is available in [20], namely $I \in \{4, 8, 16\}$. Furthermore, the Min2 architecture is only applicable to fixed-point CNPUs performing the MSA, whilst the dual-tree architecture has general applicability to any CNPU or VNPU.

V. Conclusion

In this paper, we have presented an algorithmic generalisation for the construction of optimised dual-tree NPUs, which flexibly support any number of inputs and outputs. This general-purpose architecture minimises the number of 2-input 1-output subnodes required and achieves a short critical path, making it suitable for use in any NPU in which the subnode function is non-invertible. Additionally, the proposed architecture uses bypass multiplexers to achieve full flexibility over the supported number of inputs and outputs, facilitating its use in decoder architectures in which the number of NPU inputs and outputs may vary at run-time. We also present a method for reducing the hardware requirements of these bypass multiplexers, according to the specific set of supported numbers of inputs and outputs. We have demonstrated that the proposed architecture offers better than the best of both worlds compared to two alternative general-purpose architectures. More specifically, the proposed architecture improves on the hardware resource usage of the forward-backwards architecture and improves on the maximum supported clock frequency of the multi-tree architecture in FPGA implementations. In the case of an IEEE 802.11n LDPC decoder, the proposed architecture offers a 122% improved hardware efficiency compared to the best of these two benchmarkers. Additionally, in the case of run-time flexible fixed-point CNPUs performing the MSA, the proposed dual-tree architecture offers a significantly higher hardware efficiency than the alternative two-minimum approach.

References

[1] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, vol. IT-8, no. Jan., pp. 21–28, 1962.

[2] IEEE 802.11n-2009, "Standard for Information technology–Local and metropolitan area networks–Specific requirements-Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," 2009.

[3] IEEE 802.16-2004, "Standard for Local and Metropolitan Area Networks - Part 16: Air Interface for Fixed Broadband Wireless Access Systems," 2004.

[4] 3GPP, "R1-1611081 Final Report," in *3GPP TSG RAN WG1 Meeting #86bis*, Lisbon, Portugal, oct 2016, pp. 83–89.

[5] P. Hailes, L. Xu, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "A Survey of FPGA-Based LDPC Decoders," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 2, pp. 1098–1122, jan 2016.

[6] S. S. Tehrani, S. Mannor, and W. J. Gross, "Fully parallel stochastic LDPC decoders," *IEEE Trans. Signal Process.*, vol. 56, no. 11, pp. 5692–5703, 2008.

[7] Q. Lu, Z. Shen, C.-w. Sham, and F. C. M. Lau, "A Parallel-Routing Network for Reliability Inferences of Single-Parity-Check Decoder," in *Int. Conf. Advanced Technologies Communications*, Ho Chi Minh City, Vietnam, 2015, pp. 127–132.

[8] P. Schläfer, C. Weis, N. Wehn, and M. Alles, "Design space of flexible multigigabit LDPC decoders," *VLSI Design*, pp. 1–10, 2012.

[9] M. Awais and C. Condo, "Flexible LDPC decoder architectures," *VLSI Design*, pp. 1–16, 2012.

[10] C.-H. Liu, S.-W. Yen, C.-L. Chen, H.-C. Chang, C.-Y. Lee, Y.-S. Hsu, and S.-J. Jou, "An LDPC decoder chip based on self-routing network for IEEE 802.16e applications," *IEEE J. Solid-State Circuits*, vol. 43, no. 3, pp. 684–694, 2008.

[11] T.-C. Kuo and A. N. Willson, "A flexible decoder IC for WiMAX QC-LDPC codes," in *IEEE Custom Integrated Circuits Conf.* San Jose, CA, USA: IEEE, sep 2008, pp. 527–530.

[12] P. Hailes, L. Xu, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "A Flexible FPGA-Based Quasi-Cyclic LDPC Decoder," *IEEE Access*, vol. 5, pp. 20 965–20 984, 2017.

[13] Q. Lu, C.-w. Sham, and F. C. M. Lau, "Rapid Prototyping of Multi-Mode QC-LDPC Decoder for 802.11n/ac Standard," in *Proc. Asia and South Pacific Design Automation Conference*, Macau, China, 2016, pp. 19–20.

[14] S. Kumawat, R. Shrestha, N. Daga, and R. Paily, "High-throughput LDPC-decoder architecture using efficient comparison techniques & dynamic multi-frame processing schedule," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 62, no. 5, pp. 1421–1430, 2015.

[15] I. Perez-Andrade, S. Zhong, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "Stochastic Computing Improves the Timing-Error Tolerance and Latency of Turbo Decoders: Design Guidelines and Tradeoffs," *IEEE Access*, vol. 4, pp. 1008–1038, 2016.

[16] X. Zuo, "Fully Parallel Implentation of Timing-Error-Tolerant LDPC Decoders," Ph.D. dissertation, University of Southampton, 2016.

[17] X.-Y. Hu, E. Eleftheriou, D.-M. Arnold, and A. Dholakia, "Efficient implementations of the sum-product algorithm for decoding LDPC codes," in *IEEE Global Telecommun. Conf.*, vol. 2, San Antonio, TX, USA, nov 2001, pp. 1–6.

[18] C.-L. Wey, M.-D. Shieh, and S.-Y. Lin, "Algorithms of finding the first two minimum values and their hardware implementation," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 55, no. 11, pp. 3430–3437, 2008.

[19] X. Zhu and G. He, "An Area Time-Efficient Structure to Find the Approximate First Two Minima for Min-Sum-Based LDPC Decoders," *IEEE Trans. Circuits Syst. II, Express Briefs*, vol. 63, no. 8, pp. 793–797, 2016.

[20] F. Gutierrez, G. Corral-Briones, D. Morero, T. Goette, and F. Ramos, "FPGA implementation of the parity check node for min-sum LDPC decoders," in *8th Southern Conf. Programmable Logic*, Bento Goncalves, Brazil, 2012, pp. 1–6.