A Flexible High Throughput Multi-ASIP Architecture for LDPC and Turbo Decoding

Purushotham MURUGAPPA, Rachid AL-KHAYAT, Amer BAGHDADI and Michel JEZEQUEL
E-mail: {Firstname.surname}@telecom-bretagne.eu
Electronics Department, Telecom Bretagne, Technople Brest Iroise 29238 Brest France

***Abstract*—In order to address the large variety of channel coding options specified in existing and future digital communication standards, there is an increasing need for flexible solutions. This paper presents a multi-core architecture which supports convolutional codes, binary/duo-binary turbo codes, and LDPC codes. The proposed architecture is based on Application Specific Instruction-set Processors (ASIP) and avoids the use of dedicated interleave/deinterleave address lookup memories. Each ASIP consists of two datapaths one optimized for turbo and the other for LDPC mode, while efficiently sharing memories and communication resources. The logic synthesis results yields an overall area of $2.6mm^2$ using $90nm$ technology. Payload throughputs of up to $312Mbps$ in LDPC mode and of $173Mbps$ in Turbo mode are possible at $520MHz$, fairing better than existing solutions.**

***Index Terms*—ASIP,LDPC,Turbo decoding.**

I. Introduction

The current and emerging digital communication standards target different sectors, namely, LTE and WiMAX covering metropolitan area for voice and data applications with limited video, the DVB series targeting video broadcasting for metropolitan area and Wi-Fi covering the very limited area range to support high bandwidth for gaming, networking, video and data applications. A select list of current standards and their throughput requirements are given in table.I. The user demands on the other hand, require these applications to be supported on a single portable device which calls for future wireless devices to be multi-standard. Numerous research groups have come up with different architectures that aim to solve these issues by providing enough reconfigurability to support multiple standards on a single device. A majority of these works target channel decoding, as it is computationally intensive and have complex non standardized signal processing arithmetic done on varying frame lengths and require high memory bandwidth. The types of channel coding to support usually are convolutional (CC), turbo or LDPC codes. The supported types in turbo are usually Single Binary and/ Double Binary Turbo Codes (SBTC and DBTC). The ASIC based flexible architecture presented in [2] can support multiple standards (HSDPA, WiMAX, Wi-Fi, DVB-H) with LDPC and turbo coding options. However, it occupies a large area of $0.9mm^2$ in $45nm$ technology ($3.6mm^2$ in $90nm$). In [1], a flexible architecture is presented that support Viterbi (for CC decoding), LDPC and turbo decoding (SBTC and DBTC). A high throughput of $257Mbps$ is achieved for LDPC mode while a limited throughput of $37.2Mbps$ in DBTC and $18.6Mbps$ in SBTC modes are achieved at $400MHz$. Yang et.al in [6] achieve $600Mbps$ in LDPC and $450Mbps$ in SBTC mode occupying an area of $3.2mm^2$ in 90nm technology. In spite of good throughput achieved, this architecture does not support DBTC. In this work, we present a novel architecture that achieves fair compromise in area and speed to support LDPC and turbo decoding for an array of standards (WiMAX, LTE, Wi-Fi, DVB-RCS) by sharing memories and network resources. The rest of the article is organized as follows: Section II discusses the decoding algorithms for turbo and LDPC used in the proposed architecture. Section III gives the system overview followed by functional description of the ASIP in turbo and LDPC modes. The synthesis results and comparisons w.r.t. the state of the art is given in section IV and finally the paper concludes with section V giving some future perspectives.

II. Decoding algorithms

A. Turbo decoding

The typical system diagram for the turbo decoding is shown in Fig.1a. It consists of two component decoders exchanging extrinsic information via an interleave (Π) and deinterleave (Π^{-1}) processes. The component decoder0 recieves Log-likelihood ratio Λ^k (1) for each bit k of a frame length N in the natural order while component decoder1 is initialized in interleaved order.

$$\Lambda^k = log\frac{Pr\{d^{k=0}|y^{0..N-1}\}}{Pr\{d^{k=1}|y^{0..N-1}\}} \tag{1}$$

For efficient hardware implementation Max-Log MAP algorithm is used, as described in [4]. For DBTC, the three Log-Likelihood Ratios(LLR) are defined by (2) where $i \in (01, 10, 11)$ of the k^{th} symbol , s' and s are the previous and current trellis state and d(s',s) is the decision respectively.

$$Z_k^{n.ext}(d(s',s)=i) = Z_k^{ext}(d(s',s)=i) - Z_k^{ext}(d(s',s)=00) \tag{2}$$

The extrinsic information defined by (3) is calculated from the aposteriori probability given by (4), wherein $\alpha_k(s)$ and $\beta_k(s)$ are the state metrics in forward (5) and backward recursion (6) respectively and $\gamma_k(s',s)$ are the branch metrics (7). The $\gamma_k^{sys}(s',s)$ and $\gamma_k^{par}(s',s)$ are the systematic and parity symbol LLRs. Finally, when the required number of iterations N_{iter}

are completed the hard decision is calculated as given by (9).

$$Z_k^{ext}(d(s',s)=i) = Z_k^{apos}(d(s',s)=i) - \gamma_k^{int}(s',s) \quad (3)$$

$$Z_k^{apos}(d(s',s)=i) = \max_{(s',s)/d(s',s)=i}(\alpha_{k-1}(s) + \gamma_k^{n.ext}(s',s) + \beta_k(s)), i \in \{00,01,10,11\} \quad (4)$$

$$\alpha_k(s) = max_{s',s}(\alpha_{k-1}(s) + \gamma_k(s',s)) \quad (5)$$

$$\beta_k(s) = max_{s',s}(\beta_{k+1}(s) + \gamma_{k+1}(s',s)) \quad (6)$$

$$\gamma_k(s',s) = \gamma_k^{int}(s',s) + \gamma_k^{n.ext}(s',s) \quad (7)$$

$$\gamma_k^{int}(s',s) = \gamma_k^{sys}(s',s) + \gamma_k^{par}(s',s) \quad (8)$$

$$Z_k^{Hard.dec} = sign(Z_k^{apos}) \quad (9)$$

For SBTC, the trellis length can be reduced by half through applying the one-level look-ahead recursion [6]. The modified α and β state metrics for this Radix-4 optimization are given by (10) and (11) where $\gamma_k(s'',s)$is the new branch metric for the combined two-bit symbol (u_{k-1}, u_k) connecting state s'' and s.

$$\alpha_k(s) = max_{s'',s}\{\alpha_{k-2}(s'') + \gamma_k(s'',s)\} \quad (10)$$

$$\beta_k(s) = max_{s'',s}\{\beta_{k+2}(s'') + \gamma_k(s'',s)\} \quad (11)$$

$$\gamma_k(s'',s) = \gamma_{k-1}(s'',s') + \gamma_k(s',s) \quad (12)$$

The extrinsic information for u_{k-1} and u_k are computed as:

$$Z_{k-1}^{n.ext} = max(Z_{10}^{ext}, Z_{11}^{ext}) - max(Z_{00}^{ext}, Z_{01}^{ext}) \quad (13)$$

$$Z_k^{n.ext} = max(Z_{01}^{ext}, Z_{11}^{ext}) - max(Z_{00}^{ext}, Z_{10}^{ext}) \quad (14)$$

Standard	Codes	Rates	States	Block size	Channel Throughput
IEEE-802.11 (Wi-Fi)	CC	1/2 - 3/4	64	1 -4095	6 - 54 Mbps
	CC	2/3	256	.. 1944	.. 450Mbps
	LDPC	1/2 - 5/6	-	.. 1944	.. 450Mbps
IEEE802.16 (WiMax)	CC	1/2 - 7/8	64	.. 2040	.. 54 Mbps
	DBTC	1/2 - 3/4	8	.. 648	.. 54 Mbps
	LDPC	1/2 - 3/4	-	.. 2304	≥100 Mbps
DVB-S2	LDPC	1/4 - 9/10	-	.. 64000	.. 255 Mbps
DVB-RCS	DBTC	1/3 - 6/7	-	.. 1728	.. 255 Mbps
3GPP-LTE	SBTC	1/3	-	.. 6144	.. 150 Mbps

TABLE I: Selection of standards and channel codes

B. LDPC decoding

The Fig.1c shows the typical format of the H_{base} specified to define LDPC check matrix. It consists of N_b block columns and M_b block rows wherein, each of the non-negative values $\Pi_{x,y}$ is replaced by a permutation matrix of size $Z \times Z$, where Z is the expansion factor specified by the standard. The negative values are replaced by zero square matrix of size Z. The LDPC check matrix is represented in a graphical form with a tanner graph (Fig.1b). Here CN^{xZ} $\forall$ $x = 1..M_b$ groups of check nodes of size Z connected to VN^{yZ} $\forall$ $y = 1..N_b$ groups of Variable nodes. Each VN^{yZ} of size Z is initialized with channel values Δ_n^{yZ}. The interconnecting links represent the ones in the expanded H_{base}. As with the Max-Log MAP algorithm, the 2-min algorithm is the hardware efficient implementation [1] of the belief propagation algorithm as described below. Let M(n) and N(m) be as defined below.

$$M(n) \quad = \{n | n \in (0..N_b * Z) \forall H_{nm} \neq 0\} \quad (15)$$

$$N(m) \quad = \{m | m \in (0..M_b * Z) \forall H_{nm} \neq 0\} \quad (16)$$

Every iteration i, consists of M sub-iterations corresponding to the M-check node groups in the H_{base}. Each sub-iteration consists of two phases:

- CN-update: all the VN nodes send extrinsic messages given by (17) to their corresponding check nodes.

$$\lambda_{nm}^i = \Delta_n^{i-1} - \Gamma_{mn}^{i-1}, \quad \Gamma_{mn}^0 = 0 \; if \; i = 0 \quad (17)$$

- VN-update: each check node in the group sends an update message given by (18) where, $sgn_m = \prod_{n \in N(m)} sgn(\lambda_{nm}^i)$and $min_{n'm} = \min_{n' \in N(m) \backslash n} \{|\lambda_{n'm}^i|\}$

$$\Gamma_{mn}^i = \alpha \times sgn_{nm} \times min_{n'm} \quad (18)$$

The overall updated VN is given by (19).

$$\Delta_n^i = \lambda_{nm}^i + sgn(\lambda_{nm}^i) \times \Gamma_{mn}^i \quad (19)$$

The above is repeated for all $CN^{xZ} \forall \;\; x = 1..M_b$ to complete one iteration. When the required number of iterations (N_{iter}) are completed or when all the parity check node conditions are satisfied hard decision $\Delta_n^{Hard.dec}$ is given as the sign of Δ_n^i.In this paper (17) and (18) are called as **"Running vector"** and (19) as **"Update cycle"**.

III. Multi-ASIP System architecture

The proposed **"UDec"** system architecture is shown in Fig.2. It consists of 8 ASIPs interconnected via a de-Bruijn network [3]. Within each component decoder the ASIPs are also connected by two 8-bit bus (named here to be $\alpha - \beta$ *bus*). Each ASIP can process three CNs in parallel, a total of 8 ASIPs are required to process the minimum Z=24 CNs(minimum paralellism level in LDPC). This number of ASIPs is just enough in turbo mode to work in 4×4 mode to achieve the targeted 150Mbps throughput. Each ASIP has 3

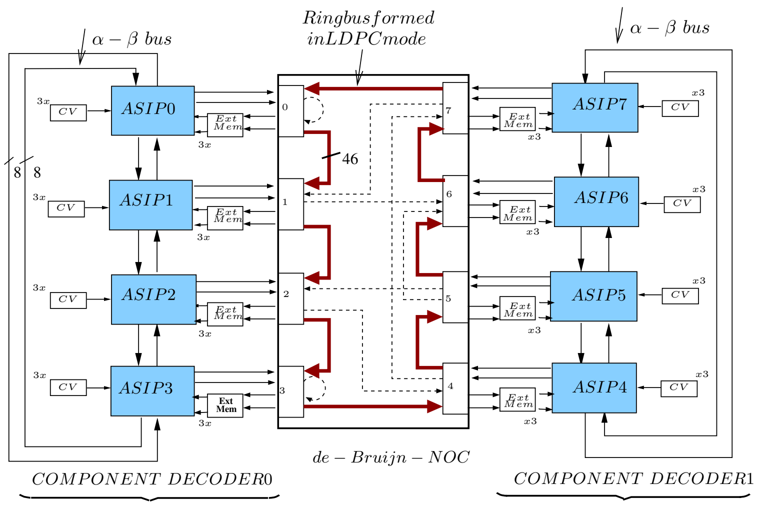
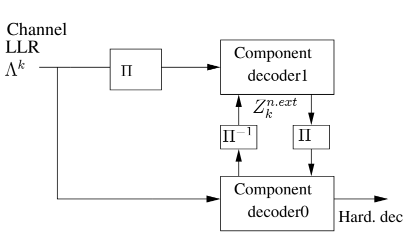
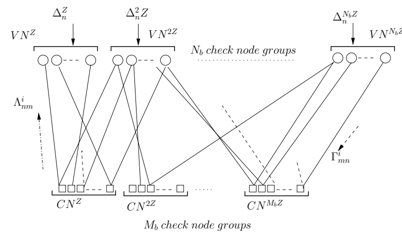


Fig. 2: UDec System Architecture

memory banks of size $24x256$ used to store the input channel LLR values (CV memories). There are also another 3 banks of size 30×256 used for storing extrinsic information. Each ASIP is further equipped with two 40×32 memories which implement buffers to store β in turbo mode and FIFO's to store λ_{nm}^i in LDPC mode (not shown in Fig.2).



(a) Turbo decoding system



(b) Tanner graph structure

$$\begin{array}{ccccc|c}
VN^Z & VN^{2Z} & VN^{yZ} & & VN^{N_bZ} & \\
\Pi_{0,0} & \Pi_{0,1} & \Pi_{0,y} & \cdots & \Pi_{0,N_b} & CN^Z \\
\Pi_{1,0} & \Pi_{1,1} & \Pi_{1,y} & \cdots & \Pi_{1,N_b} & CN^{2Z} \\
\vdots & \vdots & \vdots & & \vdots & \vdots \\
\Pi_{x,0} & \Pi_{x,1} & \Pi_{x,y} & \cdots & \Pi_{x,N_b} & CN^{xZ} \\
\vdots & \vdots & \vdots & & \vdots & \vdots \\
\Pi_{M_b,0} & \Pi_{M_b,1} & \Pi_{M_b,y} & \cdots & \Pi_{M_b,N_b} & CN^{M_bZ}
\end{array}$$

(c) H_{base}

Fig. 1: Turbo and LDPC system overview

A. *ASIP architecture: Turbo mode*

A shuffled decoding schedule [4] is adopted in this mode. Processing of large frames can be achieved by dividing the frame into N windows each with maximum size of 64 symbols. Each ASIP can have maximum of 12 windows. The Fig.3b shows the processing of the windows in Backward-Forward scheme, i.e. the ASIPs calculate the β values (backward recursion) first and then followed by α (forward recursion). State initializations $(\alpha\_int(w^i_{(n-1)}), \beta\_int(w^i_{(n-1)}))$ across ASIPs are done by message passing via the two 8-bit $\alpha - \beta bus$.

The Fig.3a shows the pipeline stages of the ASIP where the numbers indicate the equations (referred in section II-A) mapped to them. The input memories contain the channel LLRs Δ_n quantized to 6 bits each and normalized extrinsic values $\gamma^{n.ext}_{01}, \gamma^{n.ext}_{10}, \gamma^{n.ext}_{11}$ are quantized to 8 bits.

1) Assembly Code Example: An assembly code example of the ASIP in turbo mode is as shown in Fig.3c. First we initialize the ASIP mode (SBTC, DBTC), current iteration number ($iter = 0$), number of windows (N) per ASIP, length of windows (L) and the length of last window (L_{last}). The $REPEAT$ instruction controls the number of iterations ($ITER\_MAX = 6$). For the first iteration (i=0) the ASIPs start with zero as the initial state metric ($\alpha\_int(w^{i=0}_n) = \beta\_int(w^{i=0}_n) = 0$). If the executed window was the last window of the ASIP then the $EXCHANGE$ instructions is fetched to exchange 8 state metrics otherwise $ZOLB$ is fetched. The $ZOLB$ instruction controls the instructions @30-31 and @35-36 to execute L (or L_{last} in case of last window) number of times. The $DATA\_LEFT$ instruction executes the backward recursion calculating the β metric. The EX_BETA_ALPHA instruction saves the last calculated $\beta\_int(w_n{}^i)$ state metric and loads the $\alpha\_int(w^i_{(n-1)})$ metric from the previous window. The $DATA\_RIGHT$ instruction executes the forward recursion calculating α metric and $EXTCALC$ calculates the extrinsic information (3) and sends them over the de-Bruijn NOC. In case of SBTC mode, two extrinsic information is generated one for each input LLR ((13),(14)). The $EXCH\_WIN$ forwards the last $\alpha^i_{(n)}$ values as $\alpha\_int(w^i_{(n)})$, initializes state metric of the next window with $\beta_{w^i_{(n)}}$ of window n and increments the current window counter ($n = n + 1$).

2) Interleave/Deinterleave Address generation: The generated extrinsic information packets also carry the address header which determine the destination ASIP and the memory address at which the data is written. The interleaving/deinterleaving addresses required w.r.t. LTE standard QPP interleaving rule is as described below.

Let N be the number of **data couples** in each block at the encoder input.For $j = 0...N-1$, $I(j) = (F_1 * j + F_2 * j^2) modN$, where F_1 and F_2 are constants defined in the standard with j being the index of the natural order. These addresses can be recursively derived by:

$$I(j+1) = (I(j) + G(j))modN \tag{20}$$

$$G(j) = (G(j-1) + 2F_1)modN \tag{21}$$

The deinterleaved address pattern required by component decoder 1 can be generated recursively as described here. Let the deinterleaved address sequence be $D = [d_0, d_1, ..d_{N-1}]$. Taking a second order Modulo-N linear circular difference of the sequence D gives step size values as given by (22) and (23). The number of steps (Ns) depends on the block length and can be atmost 8 different values.

$$D' = [d_0 - d_{N-1}, d_1 - d_0, ..., d_{N-1} - d_{N-2}]modN \tag{22}$$

$$D'' = [D'_0 - D'_{N-1}, D'_1 - D'_0, ..., D'_{N-1} - D'_{N-2}]modN \tag{23}$$

The following pseudo code illustrates the deinterleave address generation process.

```
$for\ i = 1 : N$
    $d(i) = (d_{(i-1)} - D'_{(i-1)})modN;$
    $D'_i = (D'_{(i-1)} + D''((i-1)mod(Ns)))modN;$
$end$
```

Similar sequences can be generated for ARP interleaver, wherein the number of steps obtained is maximum of four.

B. *ASIP architecture: LDPC mode*

In the LDPC mode, the de-Bruijn NOC is reconfigured to form unidirectional interconnect of 46 bits wide as shown in the Fig.2. Each variable node message (λ^i_{nm}) and extrinsic check node message (Γ^i_{mn}) are quantized to 7 and 5 bits respectively. The entire input frame is partitioned into 8 sub-frames corresponding to 8 ASIPs and each sub-frame is divided further to 3 "fragments" of size equal to expansion factor Z. As shown in Fig.4a each channel value (CV) memory holds a fragment. Each location of CV memory holds two (λ^i_{nm}) messages. As the maximum value of $Z = 96$ (for WiMAX mode), the number of valid locations in CV memory is 48. Similarly there are 3 Extrinsic memory (Ext) banks per ASIP that store Γ^i_{mn} values. Since the VN degree is atmost 12 (for 802.11n standard) there can be at most 12 check node messages (Γ^i_{mn}) for each VN. These messages are stored in extrinsic memory in 4 groups each at an offset of 48 (refer Fig.4a). With this memory architecture an ASIP can process atmost 3 check nodes, each associated with 3 edges with 2 consecutive accesses to memories. Thus, with 8 ASIPs it is possible to process $8 \times 3 = 24$ check nodes edges in 2 clock cycles.

1) LDPC scheduling: A sub-iteration in the decoding process consists of Running vector and Update cycles carried over each group of Check nodes first on CN^Z, followed by $CN^{2Z}...CN^{M_bZ}$. An iteration is complete when all the groups have been processed. The Fig.4b shows the first two states in the LDPC decoding.

1) Running vector: at state=1, the ASIP0 process the check nodes ${CN^Z}_{1,2,3}$, while ASIP1 process the check nodes ${CN^Z}_{4,5,6}$ and so on. Each ASIP fetches the Δ_n and Γ_{mn} from CV and Ext memories respectively and also calculates λ^i_{nm}, quantizes them to 7 bits and stores them in the FIFO. It also generates three 14-bits "RVector" messages, each corresponding to a CN processed to be passed on to the adjacent ASIP via the de-Bruijn NOC. Each message contains:
   - The 2 least minimums (i.e. min0, min1) of $|\lambda^i_{nm}|$ each quantized to 4 bits.
   - The ASIP ID (3 bits)
   - Bank number (2 bits) that contains the min0.
   - $Asign = sgn_{nm}$ (1 bit) corresponding to the check node m under processing.

   At state=2, ASIP0 processes check nodes 22..24 while ASIP1 processes check nodes 1..3 and so on. The ASIPs do the same task as above except that the two least minimums found are of $|\lambda^i_{nm}|$ and (min0, min1) received from the previous ASIP. The new $Asign$ bit is the XOR of $Asign_{received}$ and the $Asign_{calculated}$. This process is repeated until all the variable nodes connected to check nodes 1..24 are covered, thus processing one complete block row of 24 check nodes in the expanded H_{base} matrix.
2) Update vector: when all the check nodes 1..24 are covered, RVector messages at the input of ASIP contain the min0, min1 of all the edges connected to the CNs under consideration (i.e. CN=1..3 for ASIP0, CN=4..6 for ASIP1, etc). These messages are now called the "UVector". The ASIPs calculate the update values Γ^i_{mn} according to (18) and save them to the extrinsic memories. Each ASIP calculates Δ^i_n which according to (19) is the sum of Γ^i_{mn} and the λ^i_{nm} (fetched from the FIFO) and updates the CV memories. The "UVector" is forwarded to the next ASIP.

If the number of check nodes remaining in the group is greater than 24, for example if the sub matrix size $Z = 96$, then the update cycles (UPDT) for $CN^Z_{1..24}$ can take place at the same time of running vector cycles (RV) of $CN^Z_{25..48}$. Thus "UVector" and current "RVector" can be passed over the NOC every alternate clock cycles.But if number of check nodes remaining is less than 24, say for $Z = 27$, then "UVector" is passed every alternate cycles while "RVector" is calculated for check nodes 25..27. During the first fetch of RunVecWithUpt(0,1) only ASIP0 executes (RV+UPDT) and ASIPs(1-7) execute only UPDT cycle. During the second fetch ASIP1 executes (RV+UPDT) while ASIPs(0,2-7) execute only UPDT cycle and so on.

2) CV and Ext memory address generation: The check nodes are processed based on the start address given by an internal LDPC address generator. If the number of ASIPs $N_A = 8$, the address pattern to be generated is given by the following pseudocode:

```
$for(p = 0; p < \lceil (Z/(3 * N_A)) \rceil; p++)\{$
 $for(state = 0; state < 8; state++)$
    $address = mod(\Pi_{x,y} + p * (3 * N_A) + 3 * (N_A - state), Z)$
$\}$
```

The Fig.4a shows the pipeline stages in the LDPC mode, with the numbers in the figure indicating the equations (from section II-B) mapped on to the hardware. The Fig.4c shows an example code for LDPC decoding for $Z = 27$. The ASIP is first initialized with processing size, sub-matrix size, three block columns of offset values from H_{base}. As the channel data is stored in couples, two clock cycles are needed to Read/Write the channel data associated with three check nodes. A read operation to CV and EXT memories is accomplished by RUNVEC and RUNVEC1 instructions. The write operation is accomplished by UPDATEVEC and UPDATEVEC1 instructions. RUNVEC1 also generates the "RVector" message to the next ASIP. Similarly, UPDATEVEC forwards the "UVector" to the next ASIP. RUNVECWITHUPDT reads the CV and EXT memories for check nodes 24..27, writes locations associated with check nodes 1..3 and then forwards this "UVector" packet to the next ASIP. RUNVECWITHUPDT1 does the second phase read from the CV and EXT memories and also calculates "RVector" message before forwarding it to the next ASIP.

IV. Synthesis Results

The ASIP was modeled in LISA language using CoWare's processor designer. The synthesis was done with $90nm$ CMOS technology that gave $0.155mm^2$ per ASIP with maximum clock frequency $F_{clk} = 520MHz$. The de-Bruijn network with 8 nodes has an area of $0.16mm^2$ as each router port is $42 + 4$ $bits\ priority = 46$. Thus the proposed UDec architecture with 8 ASIPs and interconnecting de-Bruijn network is $1.4mm^2$ with total memory area of 1.2 mm^2. The throughput estimate for LDPC mode is given by (24). The best throughput achieved is $312Mbps$ for WiMAX code rate $(Crate) = 5/6$, $Z = 96$, $M_b = 4$, $N_b = 24$ and $N_{iter} = 10$ iterations. The architecture has $N_A = 8$ ASIPs each processing $CN_A = 3$ check nodes per $Clk_{CN} = 2$ clocks.

$$\frac{Z * N_b * Crate * F_{clk}}{(\lceil (\frac{Z}{N_A * CN_A}) \rceil + 1) * Clk_{CN} * (\frac{N_b}{CN_A}) * M_b * N_{iter}} \qquad (24)$$

Similarly, equation (25) gives the throughput calculation for turbo mode. An average $N_{instr} = 4$ instructions are needed to give 1 symbol which is composed of $Bits_{sym} = 2$ bits (lines 29, 30, 35, 36 of the assembly code example given in Fig.3c). Considering $N_{iter} = 6$ iterations, the maximum throughput achieved is $173 Mbps$.

$$\frac{Bits_{sym} * F_{clk} * (N_A/2)}{N_{instr} * N_{iter}} \tag{25}$$

Table II compares the obtained results of UDec architecture with other related works. The achieved throughput is comparable to [1] in LDPC Wi-Fi mode while UDec achieves $75 Mbps$ more in LDPC WiMAX mode. In turbo mode, the UDec throughput is more than 5 times that achieved by [1] at cost of twice the occupied area (after technology normalization). [2] occupies 28% more area compared to UDec and does not achieve the throughput requirement of LTE. On the other hand, [6] achieves higher throughput in LDPC and SBTC modes at the cost of 20% more area compared to UDec and does not support DBTC.

	Core area mm^2	nm	Throughput in Mbps				F_{clk} in MHz
			LDPC WiMAX	LDPC WiFi	DBTC WiMAX, DVB-RCS	SBTC (LTE)	
UDec	2.6	90	312	263	173@6iter	173 @6iter	520
[6]	3.2	90	600	600	-	450 @6iter	500
[5]	-	90	70	70	54	14	-
[1]	0.62	65	27.7-237.8	34.5-257	18.6-37.2 @5iter	18.6 @5iter	400
[2]	0.9	45	70	100	70	18	150

TABLE II: Comparison with the state of the art

V. Conclusion

In this paper, we have presented a high throughput multi-ASIP channel decoder architecture supporting Wi-Fi, WiMAX, LTE and DVB-RCS in turbo and LDPC modes. Major gains in area were obtained by avoiding the need to have address lookup memories for turbo shuffled decoding schedule and efficient memory and communication resource sharing between turbo and LDPC modes. Future work targets to extend support to all 3GPP standards that use block interleavers and to explore low power decoding techniques.

VI. Acknowledgement

This work has been supported in part by the UDEC project. The authors would also like to thank Atif Raza Jafri for his helpful advice and inputs.

References

[1] M. Alles, T. Vogt, and N. Wehn. FlexiChaP: A reconfigurable ASIP for convolutional, turbo, and LDPC code decoding. In *2008 5th International Symposium on Turbo Codes and Related Topics*, pages 84 –89, sep. 2008.

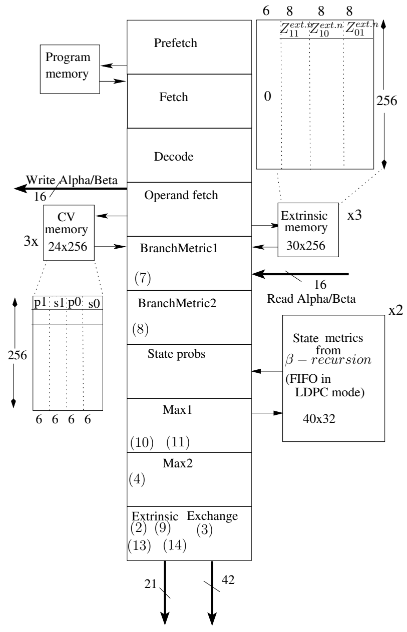
[2] M. R. Giuseppe Gentile and L. Fanucci. A Multi-Standard Flexible Turbo/LDPC Decoder via ASIC Design. In *2010 6th International Symposium on Turbo Codes and iterative information processing*, sep. 2010.

[3] H. Moussa, A. Baghdadi, and M. Jézéquel. Binary de Bruijn on-chip network for a flexible multiprocessor LDPC decoder. In *Proceedings of ACM/IEEE 45th Design Automation Conference*, pages 429–434, 2008.

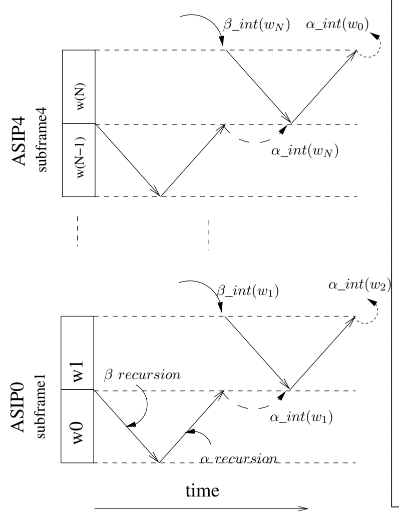
[4] O. Muller, A. Baghdadi, and M. Jezequel. From parallelism levels to a multi-asip architecture for turbo decoding. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17(1):92 –102, jan. 2009.

[5] A. Niktash, H. Parizi, A. Kamalizad, and N. Bagherzadeh. RECFEC: A Reconfigurable FEC Processor for Viterbi, Turbo, Reed-Solomon and LDPC Coding. In *Proceedings of IEEE Wireless Communications and Networking Conference*, pages 605 –610, mar. 2008.

[6] Y. Sun and J. Cavallaro. A Flexible LDPC/Turbo Decoder Architecture. *Journal of Signal Processing Systems*, pages 1–16, 2010.



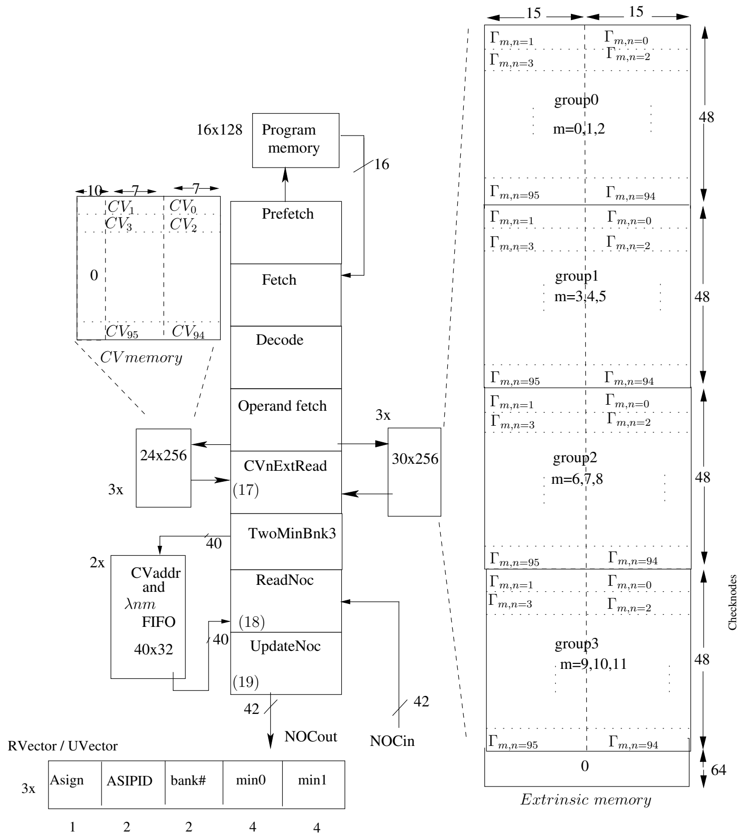
(a) Pipeline in Turbo Mode



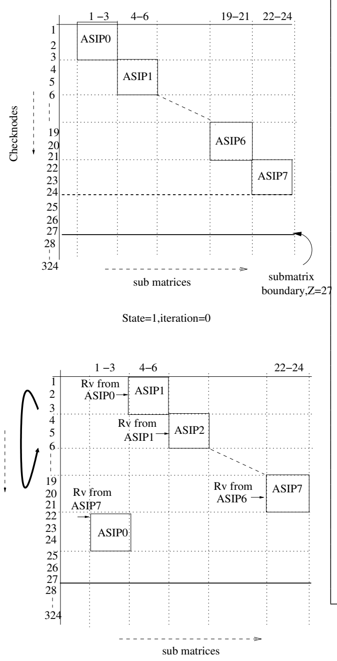
k	instruction
1	SET_CONF double
2	SET_WINDOW_ID 1
3	;setnum windows
4	SET_WINDOW_N 3
5	;1st n last window length
6	SET_SIZE 32,8
7	;repeat @11=41 if last window executed else
8	;repeat @28-41, for 6*WINDOW_N times
9	REPEAT until _LOOP 6 times
10	NOP
11	;exch alpha0-beta0 of state0
12	EXE_REC ALPHA_BETA0
13	;exch alpha-beta state1
14	EXE_REC ALPHA_BETA1
:	:
26	EXE_REC ALPHA_BETA7
27	;repeat 30-31, and 35-36 for CurrWindowLen times
28	ZOLB _RW1,_CW1,_LW1
29	NOP
30	DATA LEFT add m column2
31	_RW1: NOP
32	;save last beta load alpha_init
33	EX_BETA_ALPHA
34	_CW1: NOP
35	DATA RIGHT add m column2
36	_LW1: EXTCALC add i line2 EXT;gen ext
37	;save last alpha load beta_init if lastwindow else
38	;exch calculated alpha and beta
39	EXCH_WIN
40	NOP
41	_LOOP: NOP

(b) Windowing in Turbo mode

Fig. 3: Turbo pipeline and execution schedule



(a) Pipeline in LDPC Mode



k	instruction
1	*;8 ASIPs each processing 3 CN =24 at a time*
2	LDPCsize PSize,24
1	*;submatrix size*
3	LDPCZsize Zsize,27
1	*;rows,NumZerosTriplets*
4	LDPCAddrRegInit1 1,7
5	*;SubRows=floor(PSize/Zsize),*
6	*;num of ASIPs and RowRem=mod(Zsize,3)*
7	LDPCAddrRegInit2 1,8,1
5	*;set ASIP ID*
8	LDPCASIPid 0
9	*;writing the H matrix offset column 1*
10	LDPCAddrConfig1 0,17
11	LDPCAddrConfig1 1,3
:	:
22	*; column 2*
23	LDPCAddrConfig2 0,13
24	LDPCAddrConfig2 1,13
:	:
35	*; column 3*
36	LDPCAddrConfig3 0,8
37	LDPCAddrConfig3 1,8
:	:
48	Repeat until _ITER for 20 times
49	PUSH
50	Repeat until _LOOP0 for 8 times
51	*; load and initialize the address generator*
49	LDPCAddrGenInit
53	RunVec
54	RunVec1
55	_LOOP0: Repeat until _LOOP1 for 8 times
56	LDPCAddrGenInit
57	RunVecWithUpt
58	RunVecWithUpt1
59	_LOOP1: Repeat until _LOOP2 for 8 times
60	LDPCAddrGenInit
61	UpdateVec
62	UpdateVec1
63	_LOOP2: POP
51	;ceil(Zsize/2)=14
64	_ITER: Repeat until _DEC for 14 times
65	NOP
66	HardDecision
67	_DEC: NOP

(b) Scheduling in LDPC mode

Fig. 4: LDPC pipeline and execution schedule