VLSI Architectures for Communications and Signal Processing

Kiran Gunnam

IEEE SSCS Distinguished Lecturer

Director of Engineering, Violin Memory

Outline

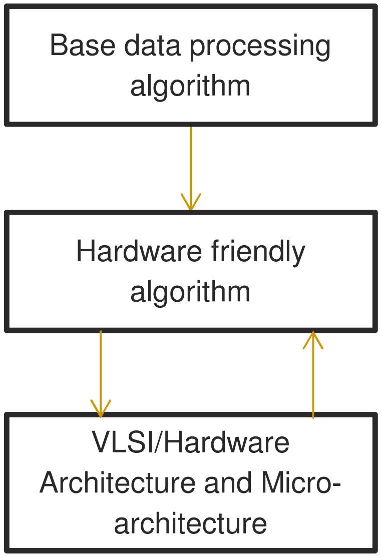
Part I

- Basics
- Pipelining and Parallel Processing
- Folding, Unfolding, Retiming, Systolic Architecture Design

Part II

- LDPC Decoder
- Turbo Equalization, Local-Global Interleaver and Queuing
- Error Floor Mitigation (Brief)
- T-EMS (Brief)

VLSI Architectures for Communications and Signal Processing


Base data processing algorithm
Hardware friendly algorithm
VLSI/Hardware Architecture and Micro-architecture

A systematic design technique is needed to transform the communication and signal processing algorithms to practical VLSI architecture.

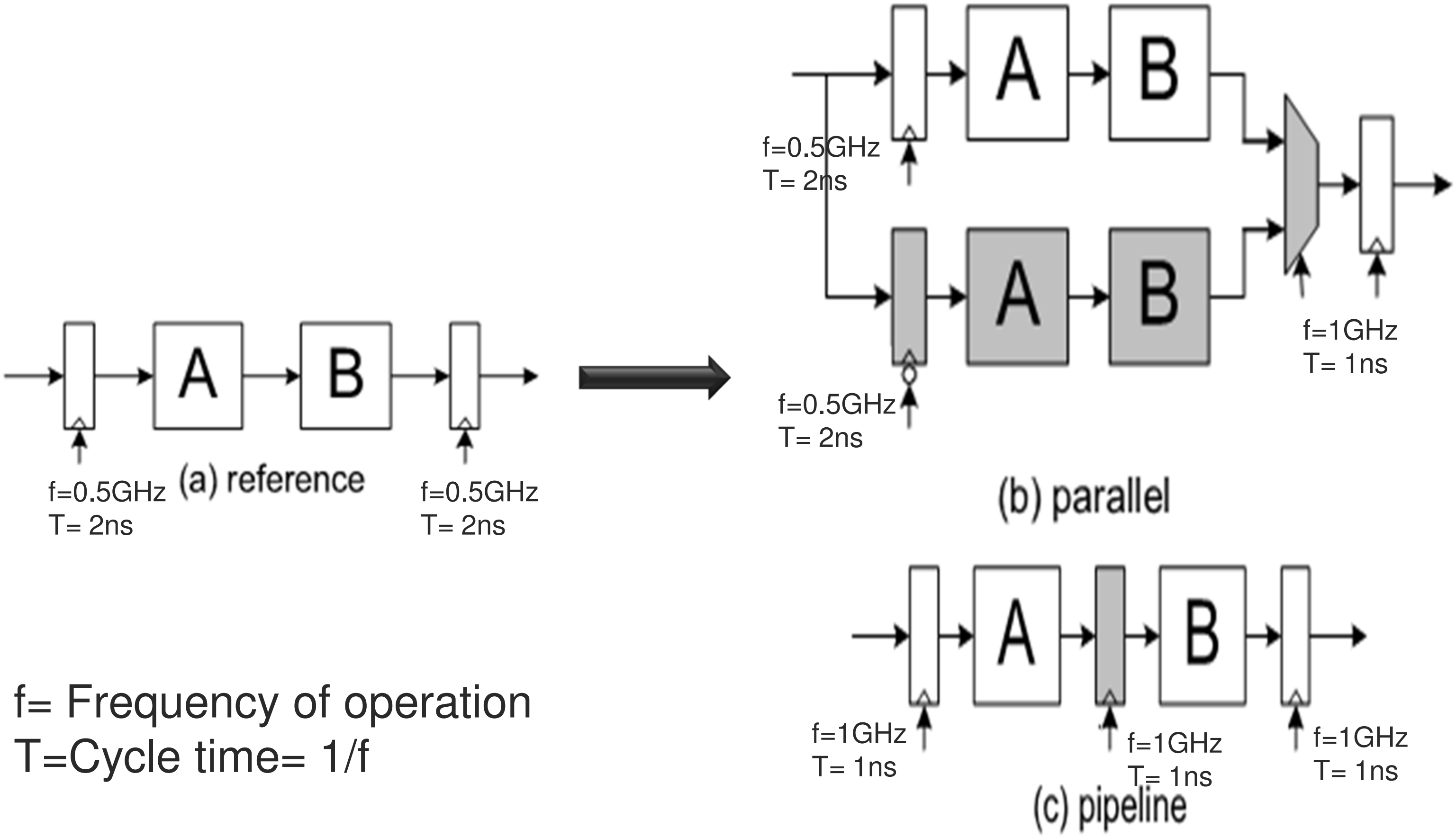
– Performance of the base algorithm has to be achieved using the new hardware friendly algorithm

– Area, power, speed constraints govern the choice and the design of hardware architecture.

– Time to design is increasingly becoming important factor: Configurable and run-time programmable architectures

– More often, the design of hardware friendly algorithm and corresponding hardware architecture involves an iterative process.

Basic Ideas



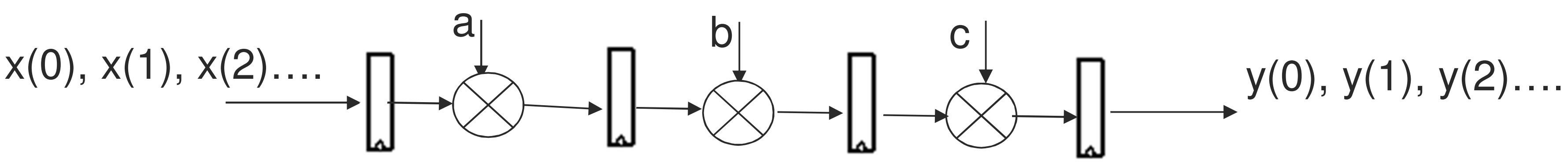
f= Frequency of operation
T=Cycle time= 1/f

Basic micro-architectural techniques: reference architecture (a), and its parallel (b) and pipelined (c) equivalents. Area overhead is indicated by shaded blocks. A and B are the operations performed on the input.

Bora et. al, “Power and Area Efficient VLSI Architectures for Communication Signal Processing”, ICC 2006

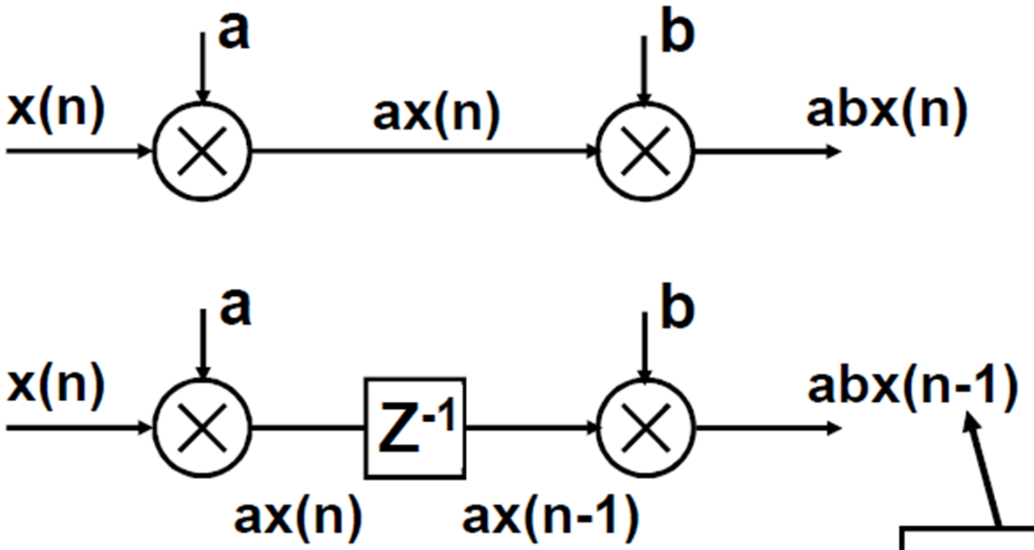
Pipelining

– Splits the logic path by introducing pipeline registers
– leads to a reduction of the critical path but introduce latency
– Either increases the clock speed (or sampling speed) or reduces the power consumption at same speed in a DSP system (due to reduced Vdd)



Clock	Operation A	Operation B	Operation C	Output
1	a x(1)	-	-	-
2	a x(2)	ab x(1)	-	-
3	a x(3)	ab x(2)	abc x(1)	-
4	-	ab x(3)	abc x(2)	y(1)
5	-	-	abc x(3)	y(2)
6	-	-	-	y(3)

Pipelining



Introduced Latency by 1cc

Critical path cut in half

- double clock speed

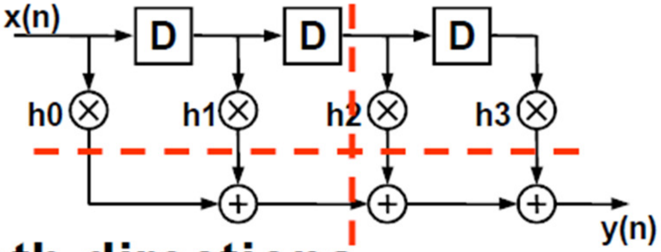
or

- lower power consumption due to reduced V_{DD}

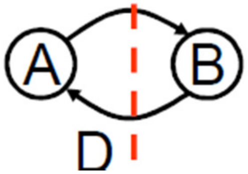
Definitions

Cutset: A set of edges that if removed, or cut, results in two disjoint graphs.

Feedforward Cutset: if data is moved in forward direction on all cutsets.

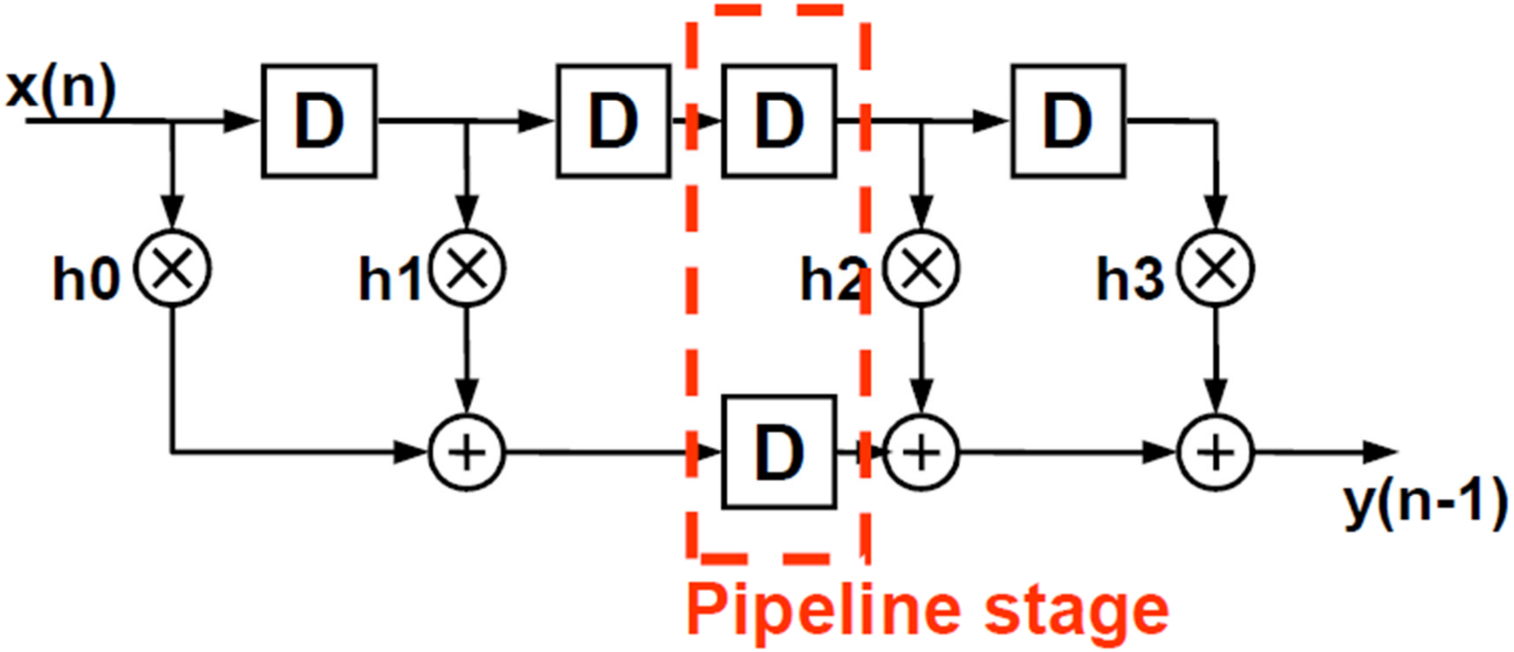


Feedback Cutset: data in both directions.



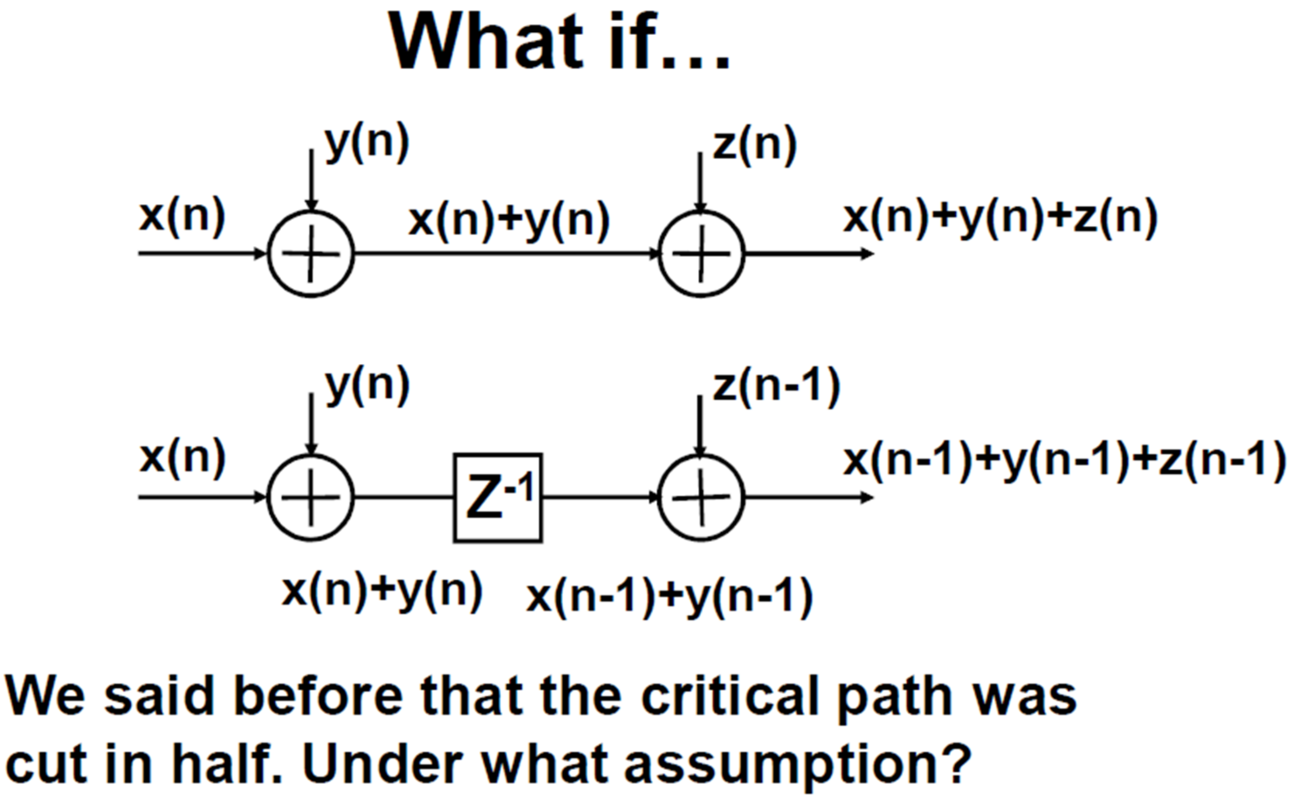
Pipelining

Pipelining: Placing delays at feedforward cutsets.



Pipelining will not affect functionality but introduce latency!

Pipelining-Critical Path, (quiz 2)

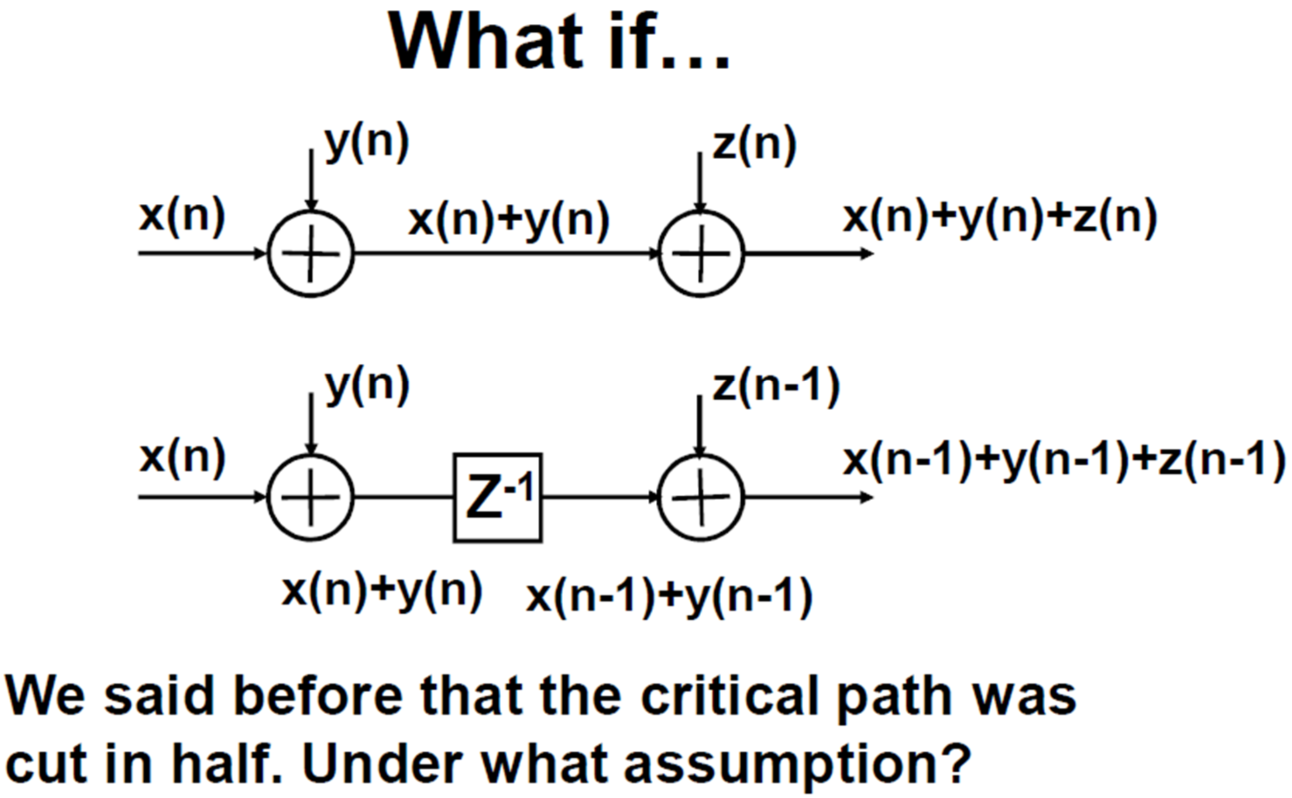


What if…

We said before that the critical path was cut in half. Under what assumption?

Any assumption on adders?

Pipelining-Critical Path, (quiz 2 solution)



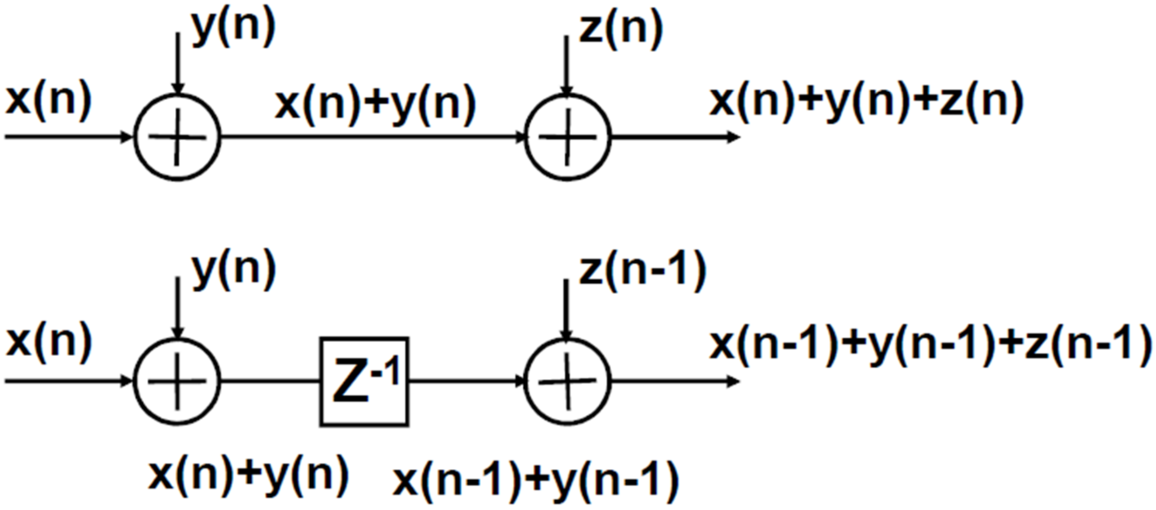
What if…

We said before that the critical path was cut in half. Under what assumption?

If the adders are carry look ahead adders , our assumption holds correct

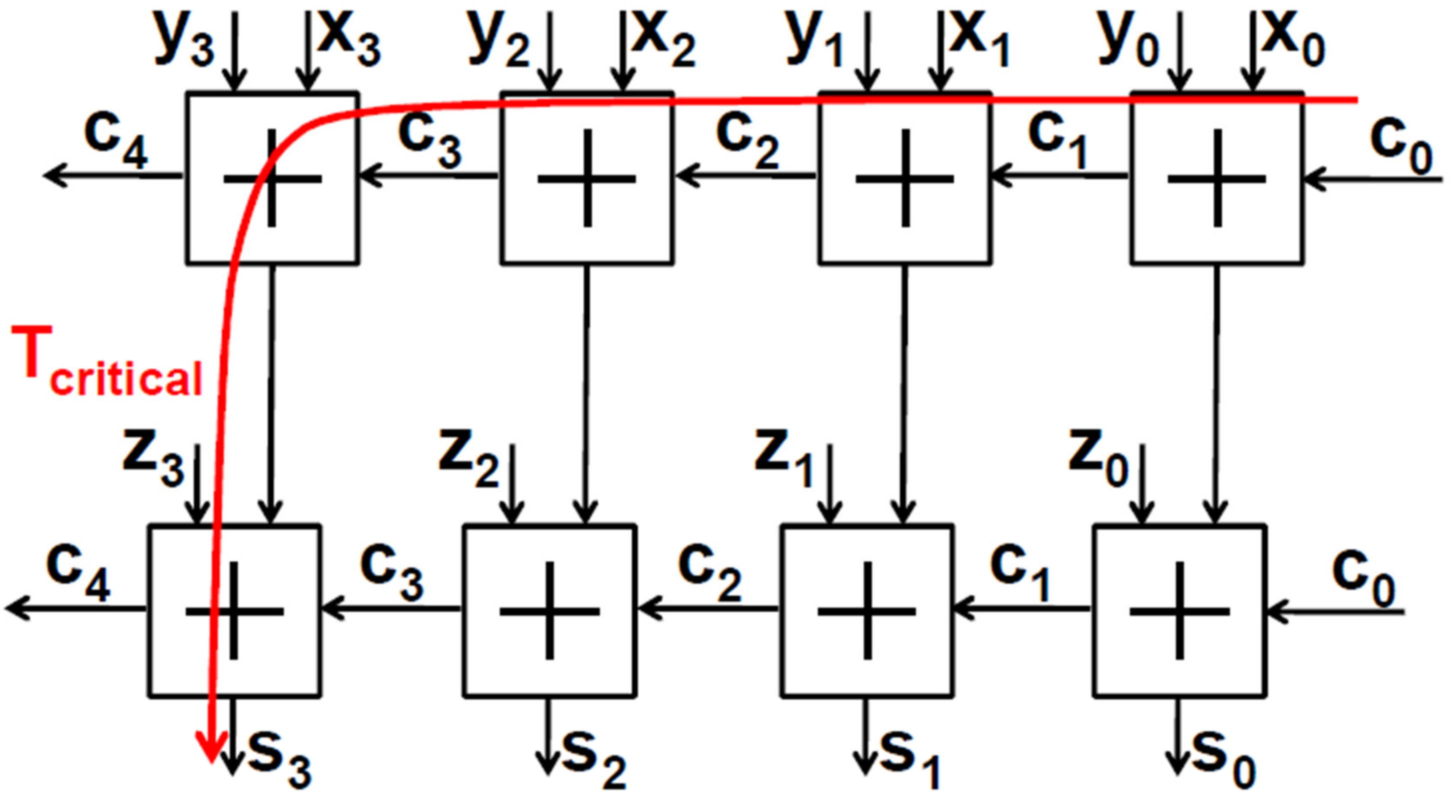
Pipelining-Critical Path, (quiz 3)

What if…



What if adders are rippler carry adders?
What is the savings in the critical path assuming a 4-bit sample for x(n),y(n) and z(n) ?

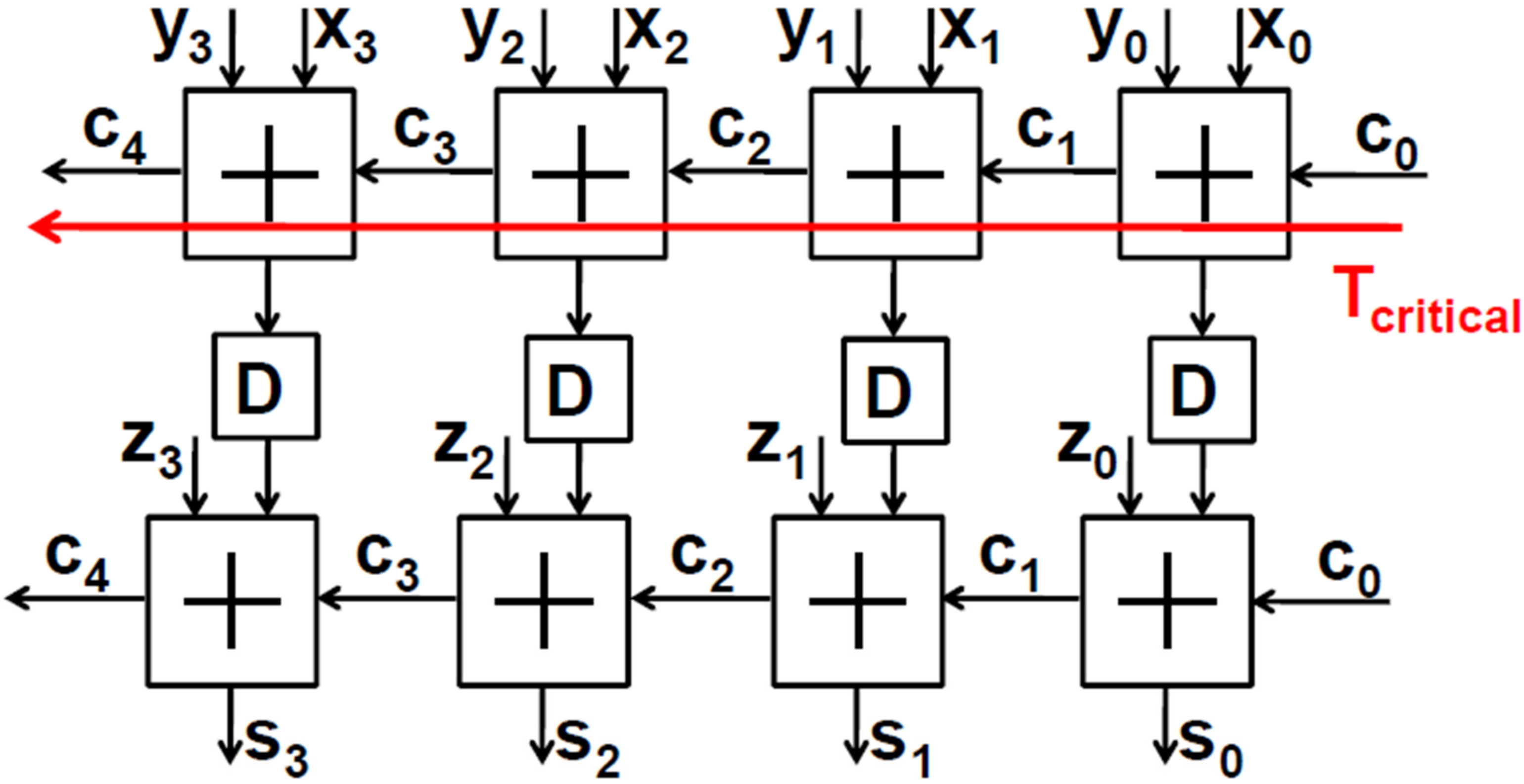
Quiz 3: Solution



What is the critical path now?

$$4 \times c_{delay} + s_{delay} \approx 5 \times c_{delay}$$

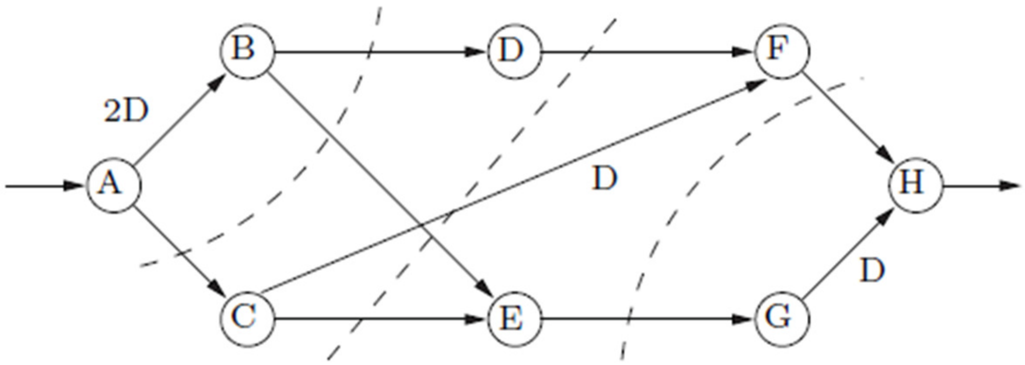
Example: Pipelining when ripple-carry adders



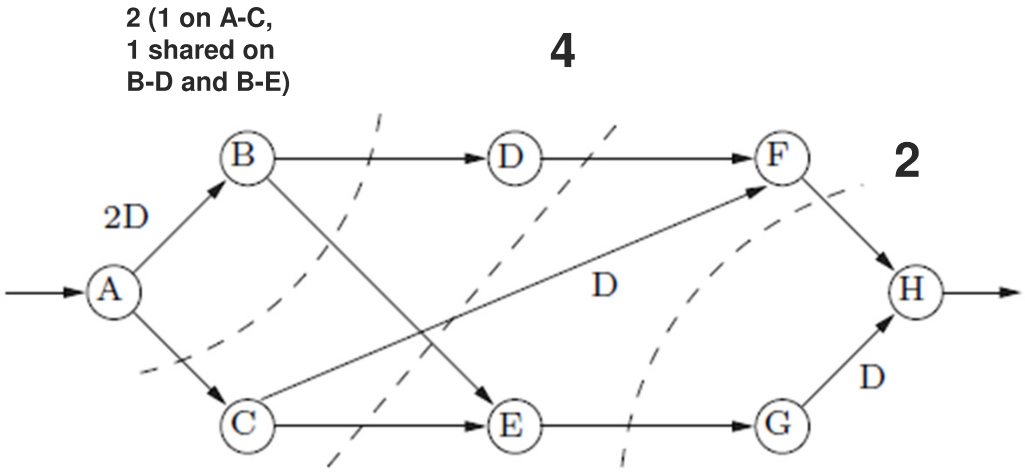
What is the critical path now?

4 x c_{delay}

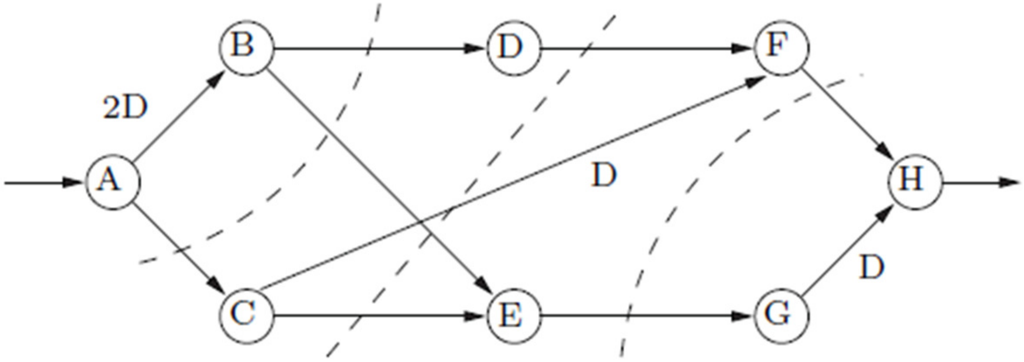
Quiz 4: Identify the number of registers
Needed for each of feed forward cutset (dotted line)



Quiz 4 Solution: Identify the number of registers Needed for each of feed forward cutset (dotted line)



Quiz 5: Compute the sample rate for different scenarios. Assume all the functions have same execution time T.



Without any feed forward cutset pipelining: fs=1/4T

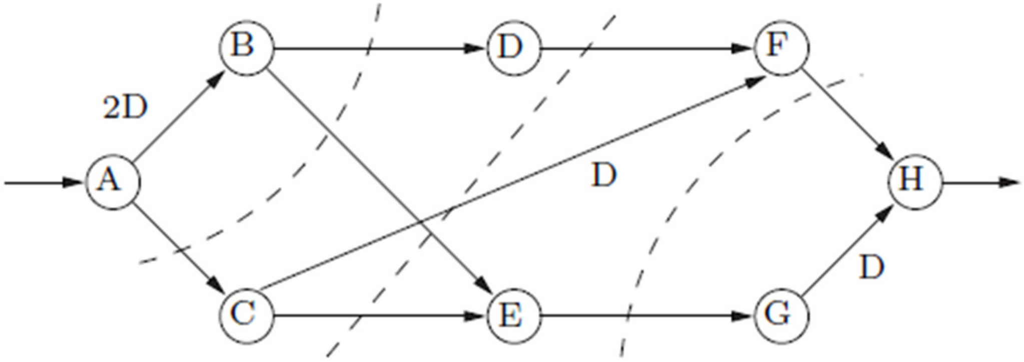
With all the cutsets pipelining:

With the right feed forward cutset pipelining:

With the left feed forward cutset pipelining:

With the central cutset pipelining:

Quiz 5: Compute the sample rate for different scenarios. Assume all the functions have same execution time T.



Without any feed forward cutset pipelining: B-D-F-H and A-C-E-G. fs=1/4T

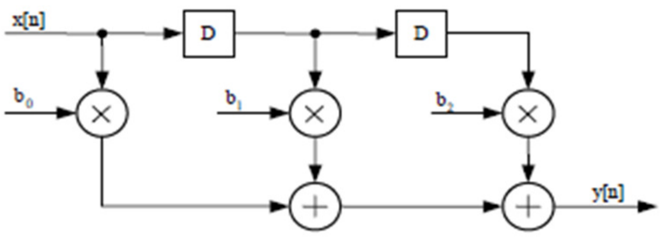
With all the cutsets pipelining: fs=1/4T

With the right feed forward cutset pipelining: D-F-H and C-E-G. fs=1/3T

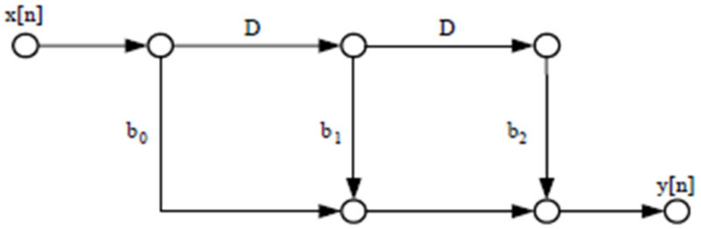
With the left feed forward cutset pipelining: B-D-F and A-C-E. fs=1/3T

With the central cutset pipelining: B-D, F-H , A-C and E-G, A. fs=1/2T

Block Diagram and SFG



Block diagram



SFG

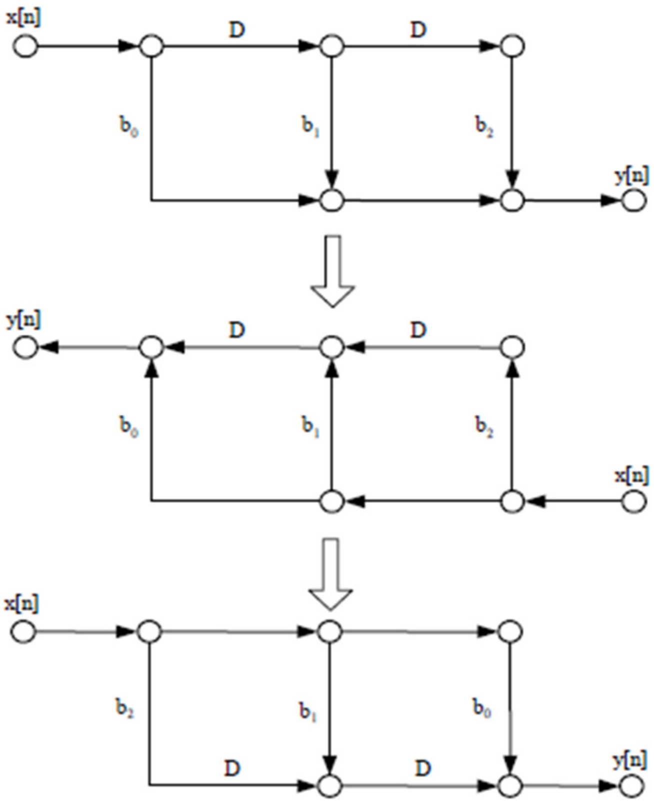
$$y[n] = \sum_{k=0}^{M-1} b_k \cdot x[n-k]$$

$$t_c = (M-1) \cdot t_{add} + t_{mult}$$

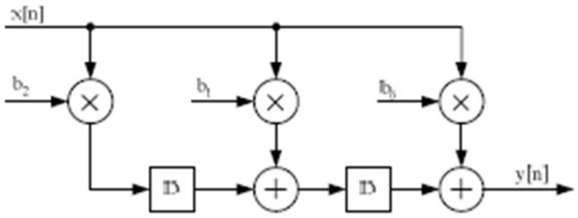
$$t_c = \lceil \log_2 (M) \rceil \cdot t_{add} + t_{mult}$$

When we use tree adders

SFG Transformation



- Reverse the direction of all edges
- Exchange the input and output nodes
- Resulting SFG maintains the same system functionality
- Critical path of FIR filter is now independent of number of taps



$$t_c = t_{add} + t_{mult}$$

Block Processing/Vectorized Parallel Processing

- One form of vectorized parallel processing of DSP algorithms. (Not the parallel processing in most general sense)
- Block vector: [x(3k) x(3k+1) x(3k+2)]
- Clock cycle: can be 3 times longer
- Original (FIR filter):

$$y(n) = a \cdot x(n) + b \cdot x(n-1) \\ + c \cdot x(n-2)$$

Courtesy: Yu Hen Hu

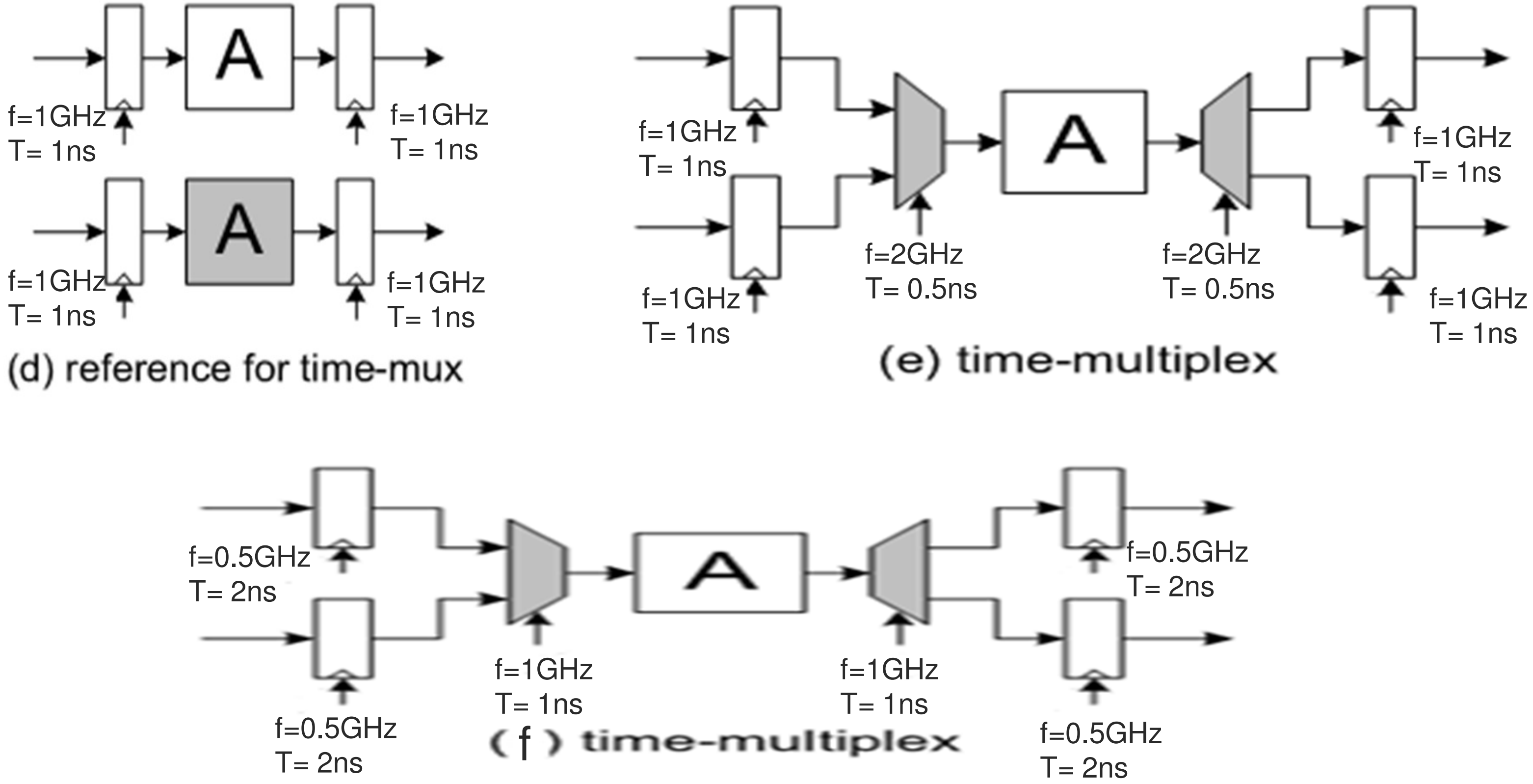
- Rewrite 3 equations at a time:

$$\begin{bmatrix} y(3k) \\ y(3k+1) \\ y(3k+2) \end{bmatrix} = a \begin{bmatrix} x(3k) \\ x(3k+1) \\ x(3k+2) \end{bmatrix} + b \begin{bmatrix} x(3k-1) \\ x(3k) \\ x(3k+1) \end{bmatrix} + c \begin{bmatrix} x(3k-2) \\ x(3k-1) \\ x(3k) \end{bmatrix}$$

- Define block vector $\mathbf{x}(k) = \begin{bmatrix} x(3k) \\ x(3k+1) \\ x(3k+2) \end{bmatrix}$
- Block formulation:

$$\mathbf{y}(k) = \begin{bmatrix} a & 0 & 0 \\ b & a & 0 \\ c & b & a \end{bmatrix} \mathbf{x}(k) + \begin{bmatrix} 0 & c & b \\ 0 & 0 & c \\ 0 & 0 & 0 \end{bmatrix} \mathbf{x}(k-1)$$

Time- MUX



(d) reference for time-mux

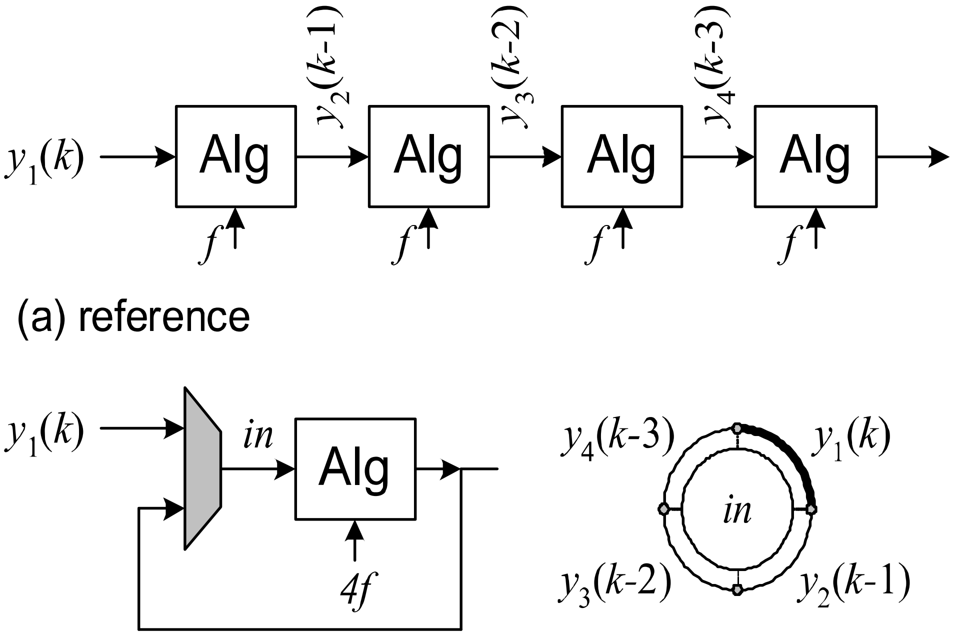
(e) time-multiplex

(f) time-multiplex

Reference architecture (d) for time-multiplexing (e) and the time multiplexer design with same throughput as reference (f). Area overhead is indicated by shaded blocks. A is the operation performed on the input.

Bora et. al, “Power and Area Efficient VLSI Architectures for Communication Signal Processing”, ICC 2006

Folding



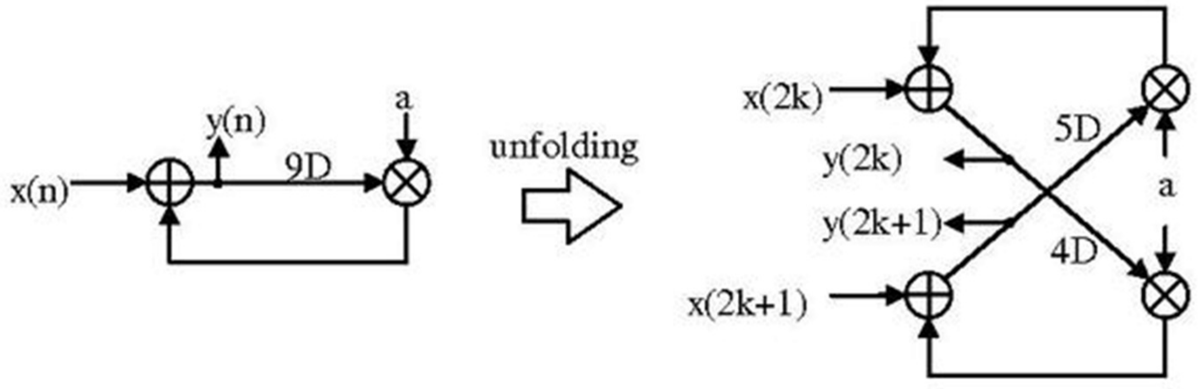
(a) reference

(b) folding

Concept of folding: (a) time-serial computation, (b) operation folding.
Block ***Alg*** performs some algorithmic operation. For example Alg may be x(n)=x(n-1) * x(n).

Bora et. al, “Power and Area Efficient VLSI Architectures for Communication Signal Processing”, ICC 2006

Unfolding



- Transform the Data Flow Graph (DFG) of 1 input and 1 output into DFG that receives 2 inputs and produce 2 outputs at each time.

Original Expression:

$y(n) = ay(n-9) + x(n)$ $\Longrightarrow$

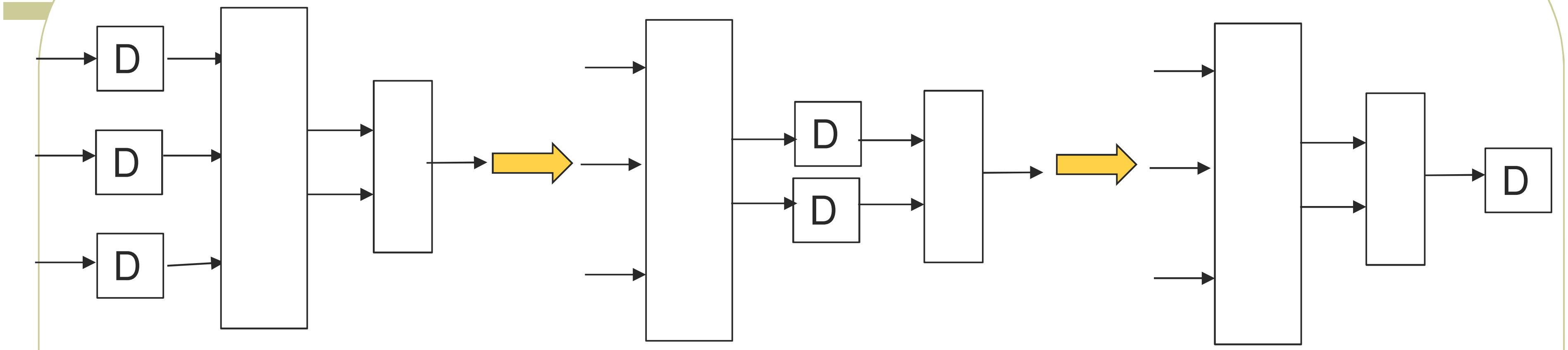
Converted DFG:

$$
\begin{aligned}
y(2k) &= ay(2k-9) + x(2k) \\
&= ay((2k+1)-\mathbf{10}) + x(2k) \\
y(2k+1) &= ay(2k+1-9) + x(2k+1) \\
&= ay(2k-\mathbf{8}) + x(2k+1)
\end{aligned}
$$

- Delay on each path gets decreased by the **unfolding factor 2** (Because at each time there are 2 inputs and 2 outputs).
- So the delays 10 and 8 gets decreased to 5 and 4. It is easy to see as on each edge of DFG, we are considering alternate sample times.

Courtesy: Yu Hen Hu

Retiming – Moving Delays



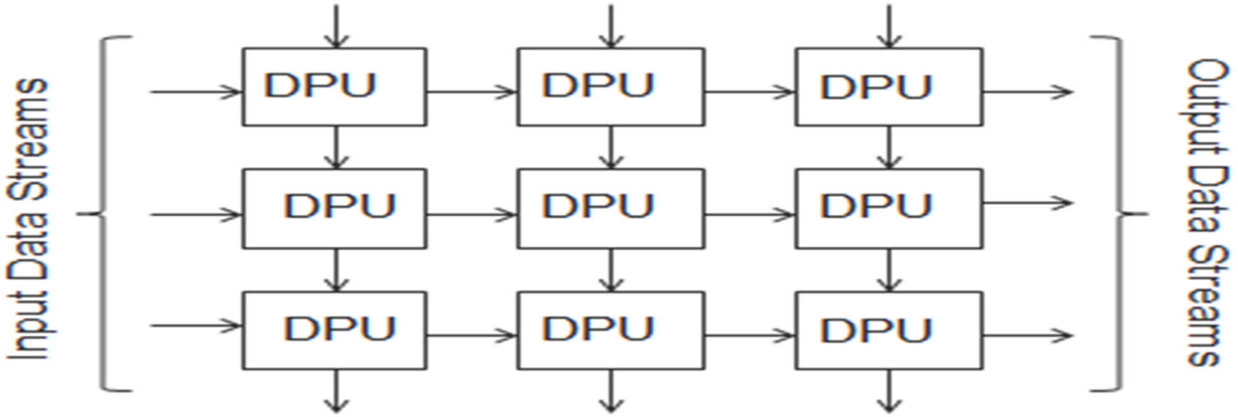
Delays can be moved from ALL inputs to ALL outputs.

Reduce Critical path

- Faster
- Reduced Power consumption

Reduced number of registers

Systolic Architectures

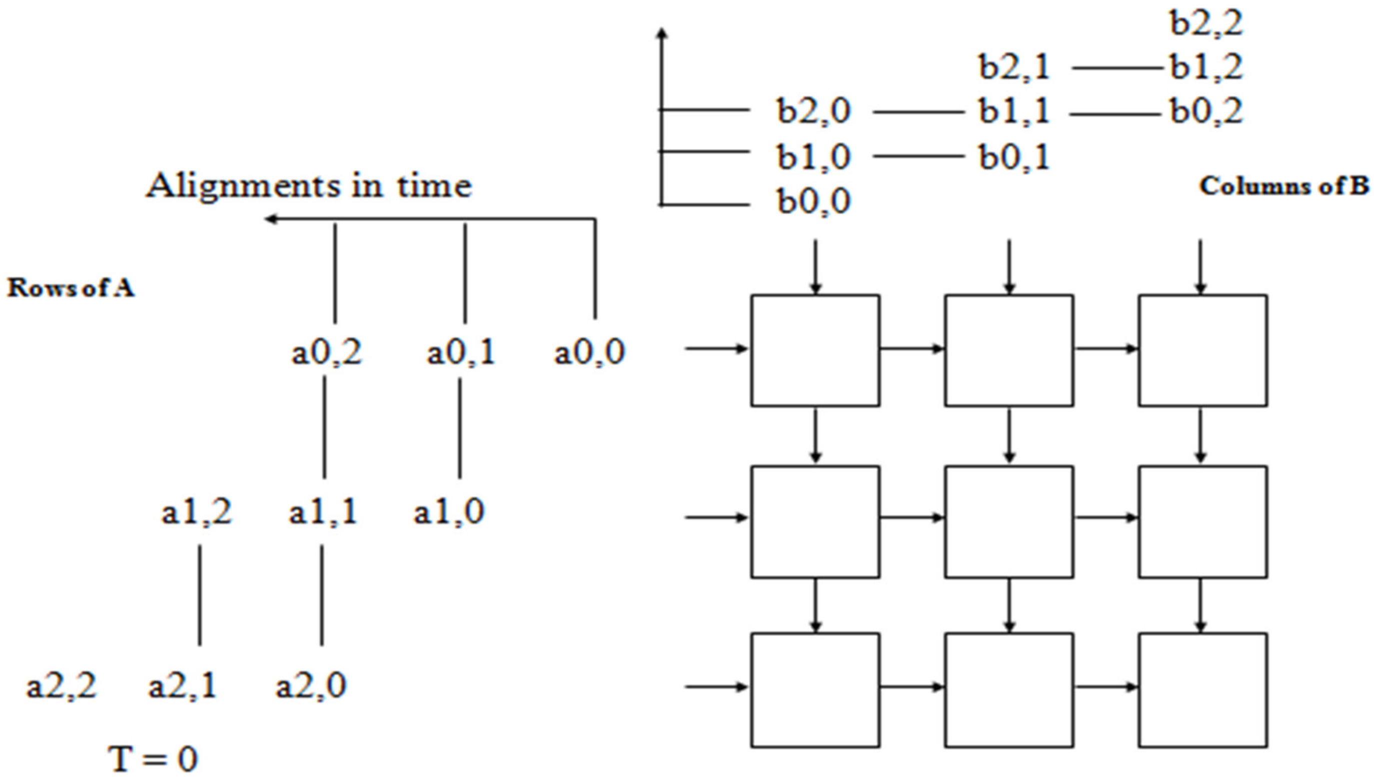


- Matrix-like rows of Data Processing Units (DPU) called cells.
- Transport Triggered.
- Consider Matrix multiplication C=A*B.

$$A = \begin{bmatrix} a0,0 & a0,1 & a0,2 \\ a1,0 & a1,1 & a1,2 \\ a2,0 & a2,1 & a2,2 \end{bmatrix} \qquad B = \begin{bmatrix} b0,0 & b0,1 & b0,2 \\ b1,0 & b1,1 & b1,2 \\ b2,0 & b2,1 & b2,2 \end{bmatrix}$$

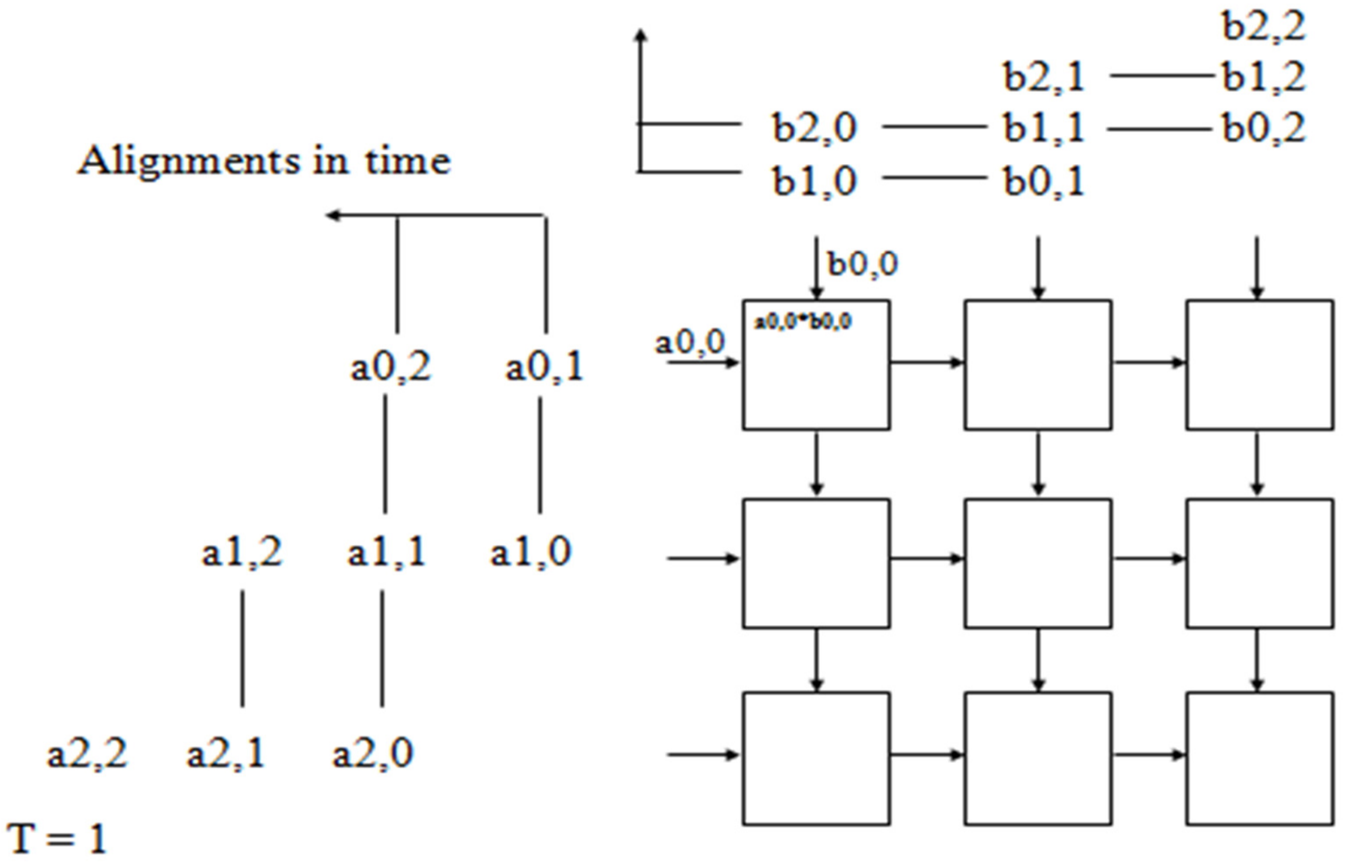
- A is fed in a column at a time from the left hand side of the array and passes from left to right.
- B is fed in a row at a time from the top of the array and is passed down the array.

Systolic Architectures (Contd.)

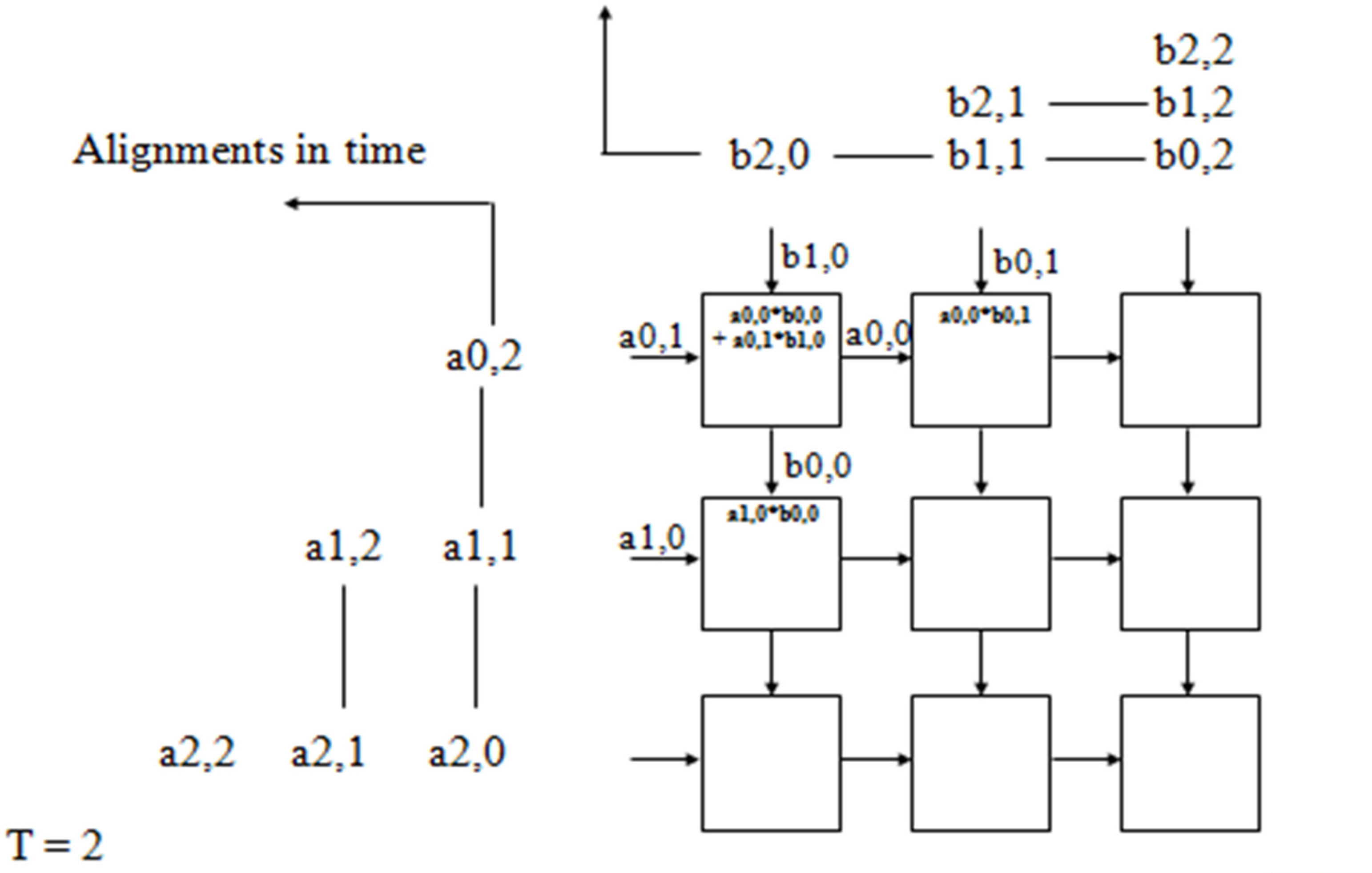


- Dummy values are passed in until each processor has seen one whole row and one whole column.
- The result of the multiplication is stored in the array and can now be output of a row or a column at a time, flowing down or across the array.

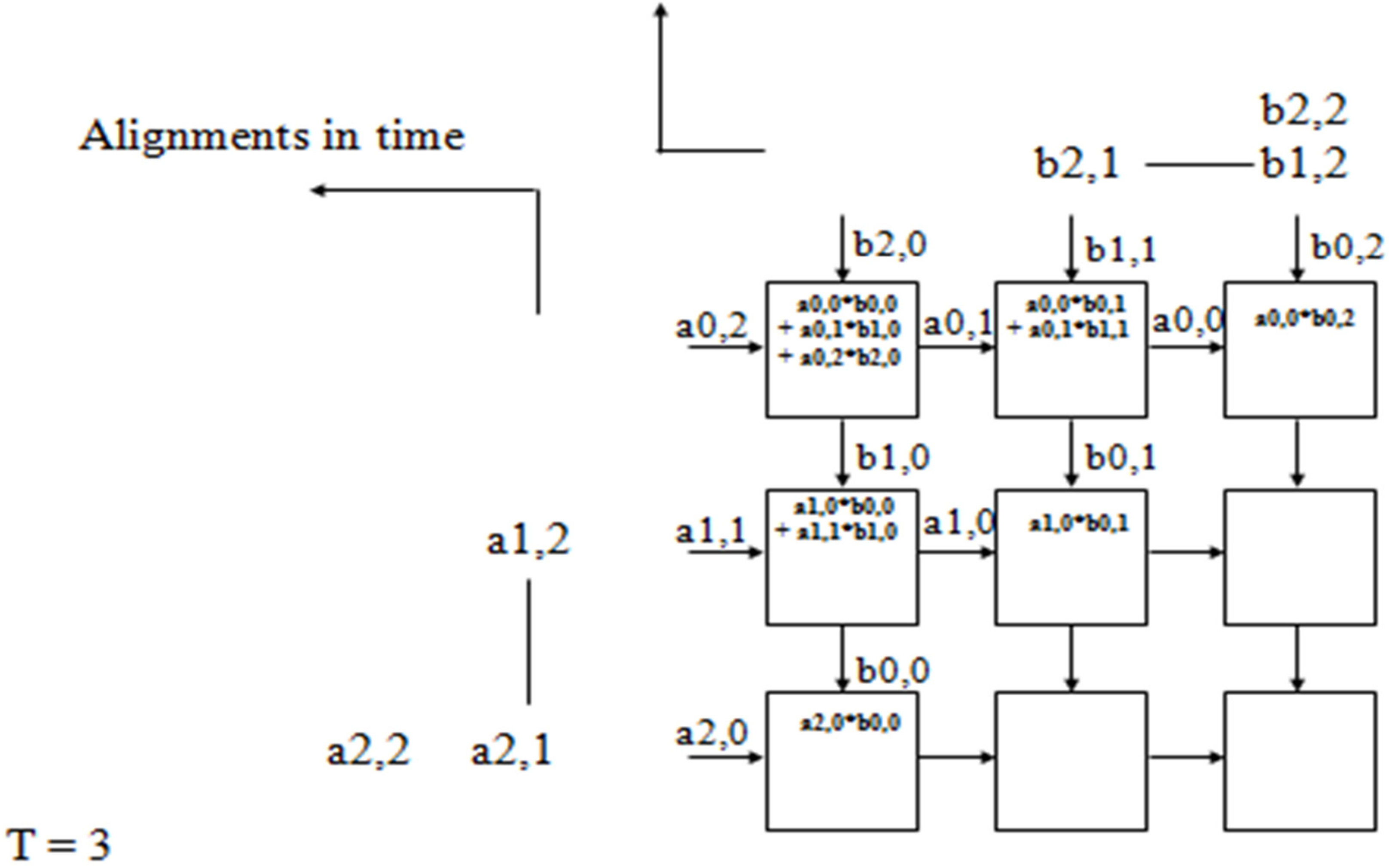
Systolic Architectures (Contd.)



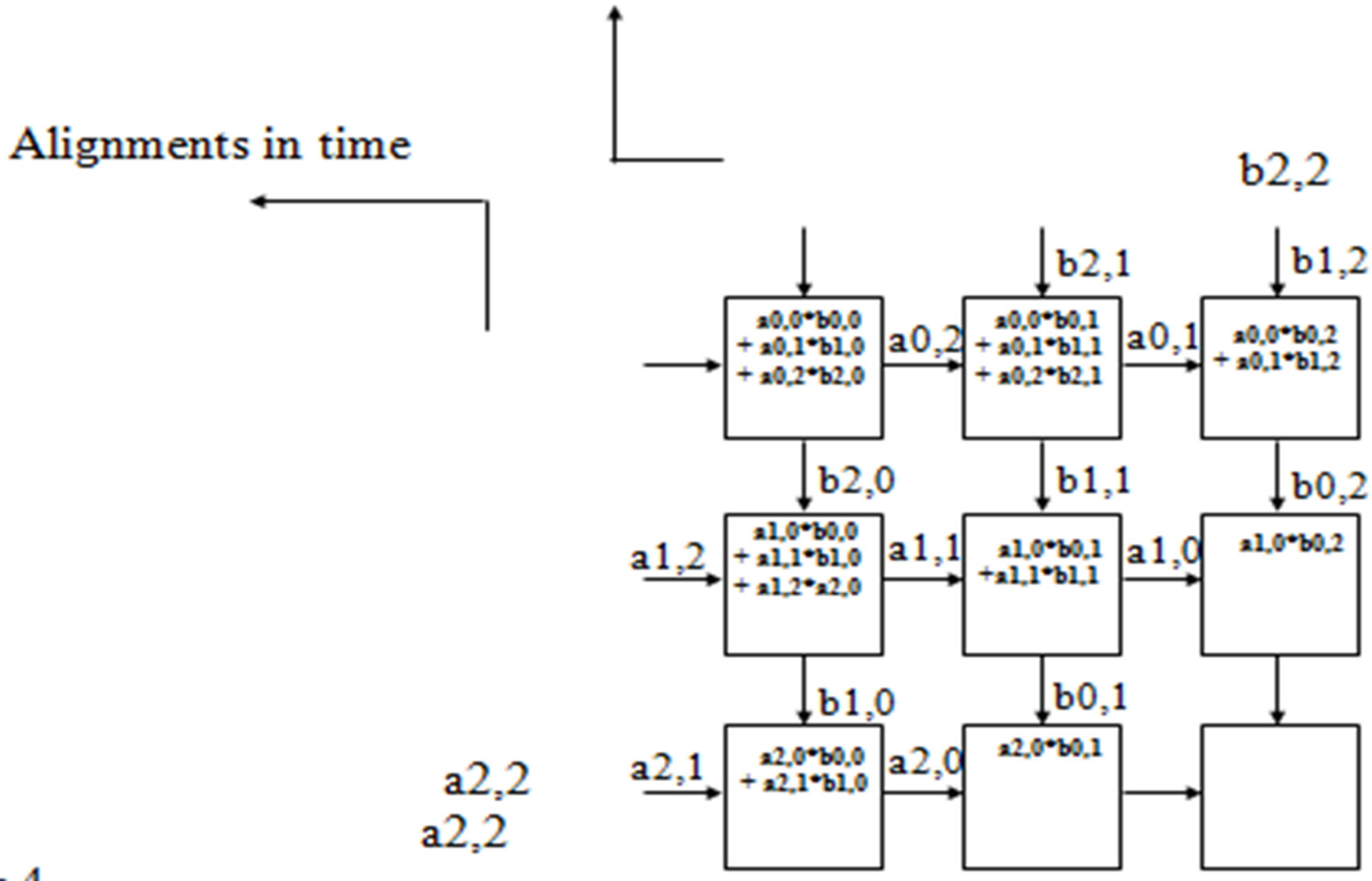
Systolic Architectures (Contd.)



Systolic Architectures (Contd.)



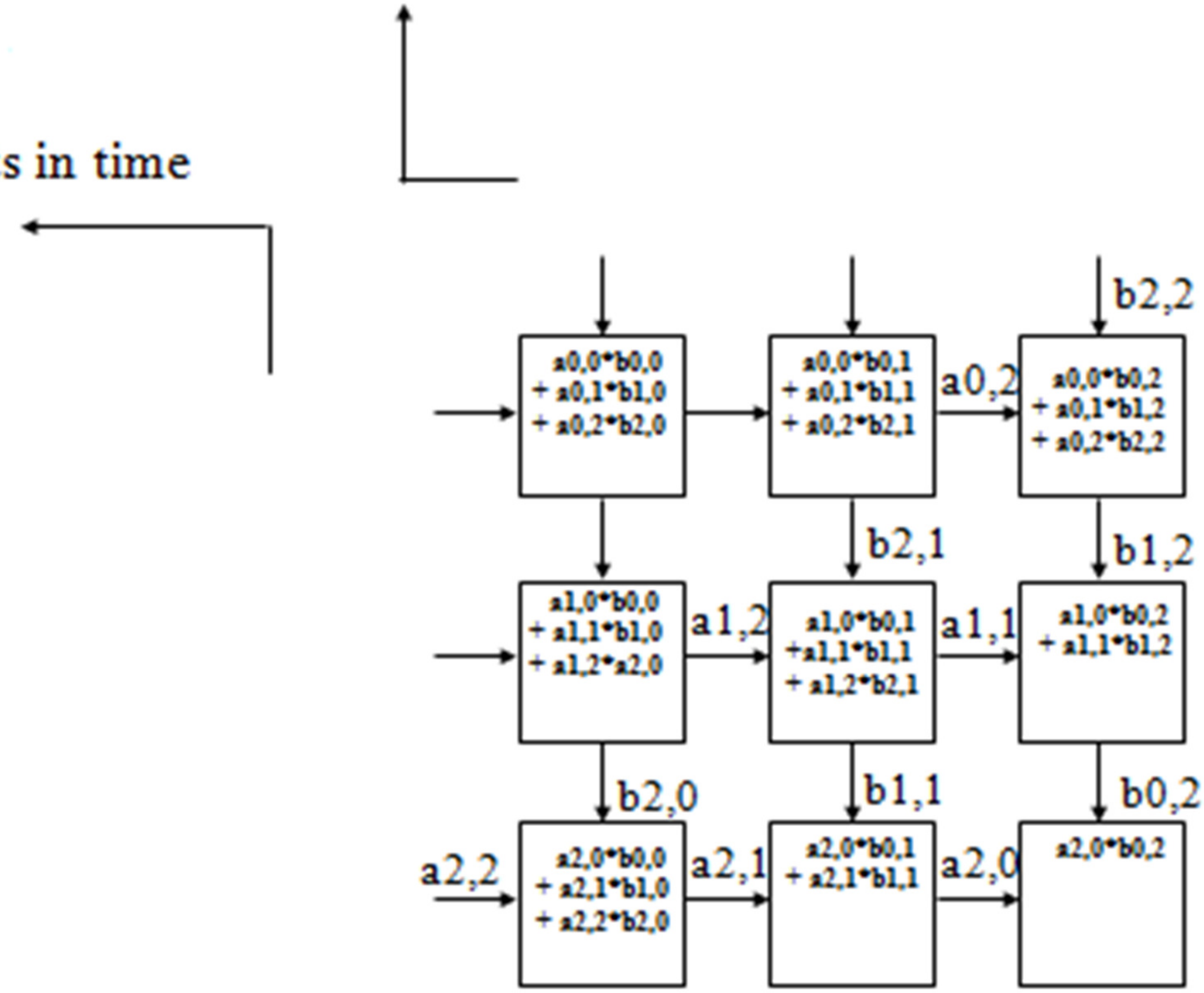
Systolic Architectures (Contd.)


Alignments in time

T = 4

Systolic Architectures (Contd.)

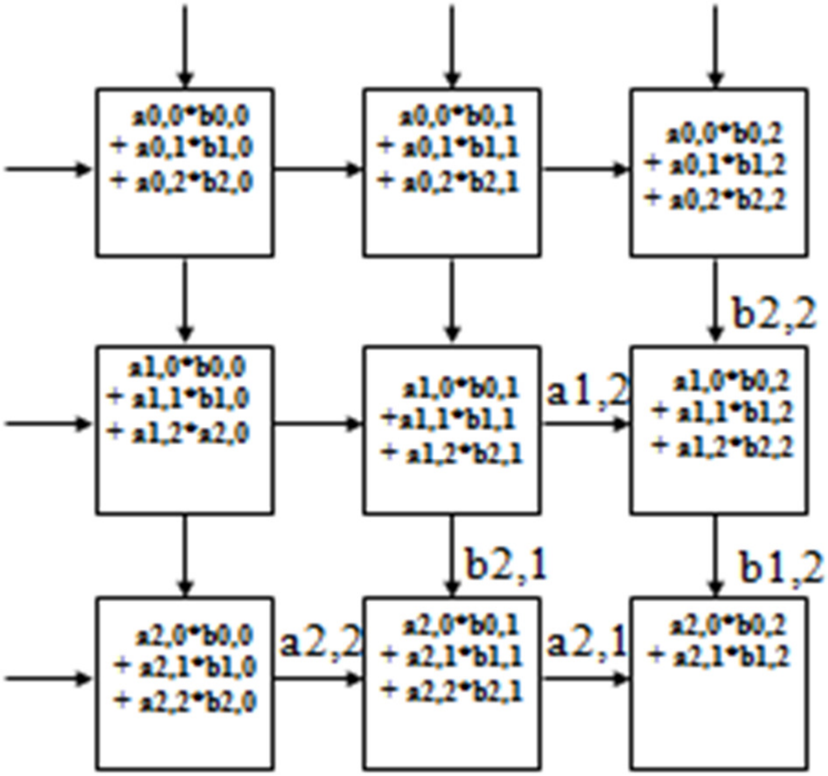
Alignments in time



T = 5

Systolic Architectures (Contd.)

Alignments in time



a0,0*b0,0
+ a0,1*b1,0
+ a0,2*b2,0

a0,0*b0,1
+ a0,1*b1,1
+ a0,2*b2,1

a0,0*b0,2
+ a0,1*b1,2
+ a0,2*b2,2

b2,2

a1,0*b0,0
+ a1,1*b1,0
+ a1,2*a2,0

a1,0*b0,1
+a1,1*b1,1
+ a1,2*b2,1

a1,2

a1,0*b0,2
+ a1,1*b1,2
+ a1,2*b2,2

b2,1

b1,2

a2,0*b0,0
+ a2,1*b1,0
+ a2,2*b2,0

a2,2

a2,0*b0,1
+ a2,1*b1,1
+ a2,2*b2,1

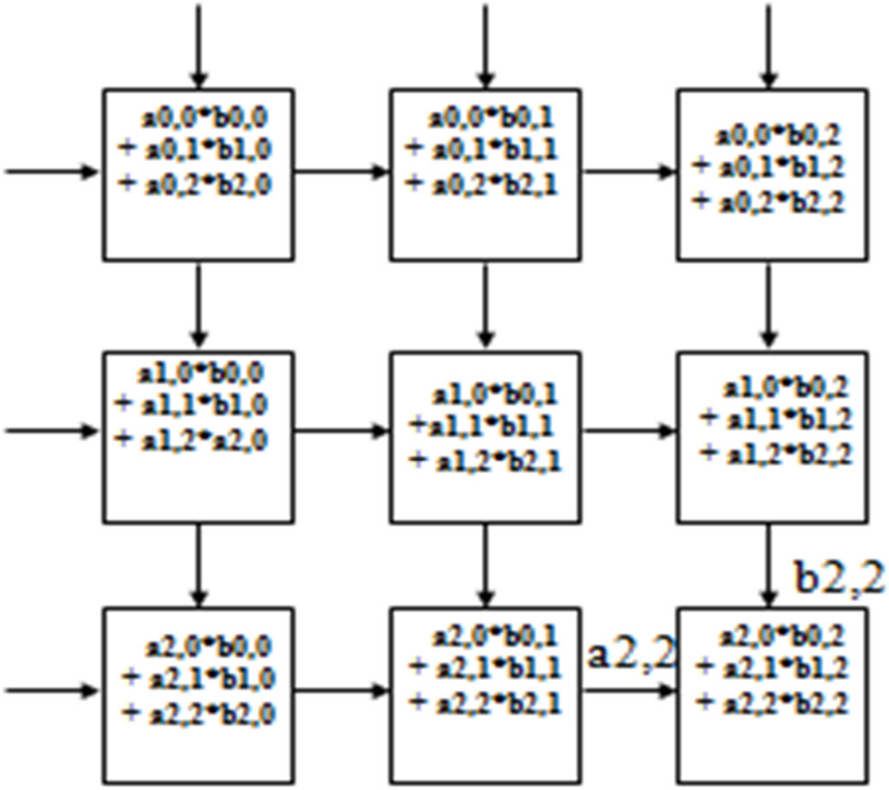
a2,1

a2,0*b0,2
+ a2,1*b1,2

T = 6

Systolic Architectures (Contd.)

Alignments in time



a0,0*b0,0 + a0,1*b1,0 + a0,2*b2,0	a0,0*b0,1 + a0,1*b1,1 + a0,2*b2,1	a0,0*b0,2 + a0,1*b1,2 + a0,2*b2,2
a1,0*b0,0 + a1,1*b1,0 + a1,2*a2,0	a1,0*b0,1 +a1,1*b1,1 + a1,2*b2,1	a1,0*b0,2 + a1,1*b1,2 + a1,2*b2,2
a2,0*b0,0 + a2,1*b1,0 + a2,2*b2,0	a2,0*b0,1 + a2,1*b1,1 + a2,2*b2,1	a2,0*b0,2 + a2,1*b1,2 + a2,2*b2,2

Done

T = 7

Part 2: Low Complexity Iterative LDPC solutions for Turbo Equalization

Outline

- LDPC Decoder
- Turbo Equalization, Local-Global Interleaver and Queuing
- Error Floor Mitigation (Brief)
- T-EMS (Brief)

LDPC DECODER

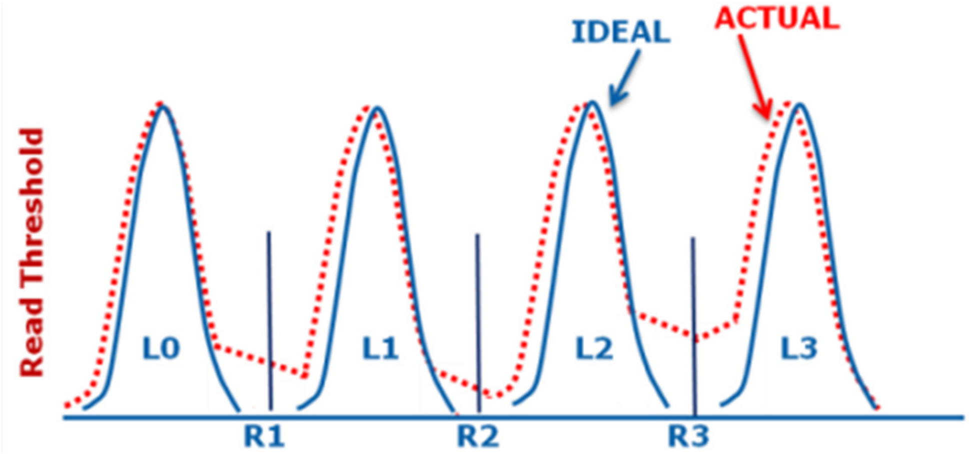
Requirements for Wireless systems and Storage Systems

Magnetic Recording systems for HDD

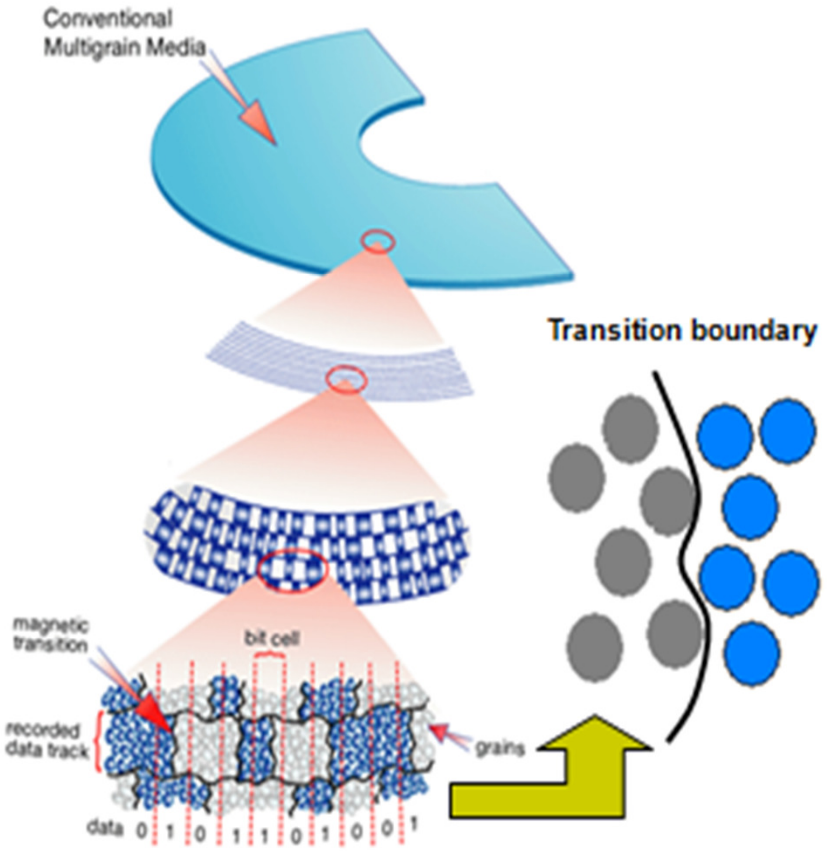
- Data rates are 3 to 5 Gbps.
- Real time BER requirement is 1e-10 to 1e-12
- Quasi real-time BER requirement is 1e-15 to 1e-18
- Main Channel impairments: ISI+ data dependent noise (jitter) + erasures
- Channel impairments are getting worse with the increasing recording densities.

Flash Channel (Flash Memory Storage Systems)

- Data rates are 0.3 to 5 GBps.
- Similar BER requirements as Hard Disk drives
- Main Channel impairments: Threshold voltage distributions change over time and due to Program/Erase cycles.



Transition Jitter Noise



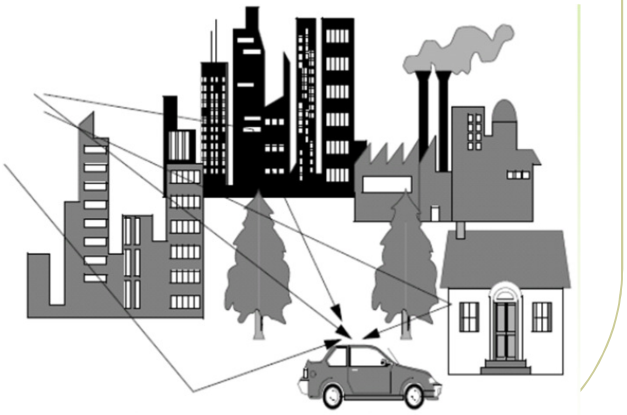
©Hitachi Global Storage Technologies

- Transitions meanders between random grains.
- This transition jitter causes noise.
- More grains at the boundary can make the transition smoother, and thus reduce noise.
- Normally, for each bit cell, there must be 100 or more grains to get good signal-to-noise ratio (SNR).

Wireless Systems

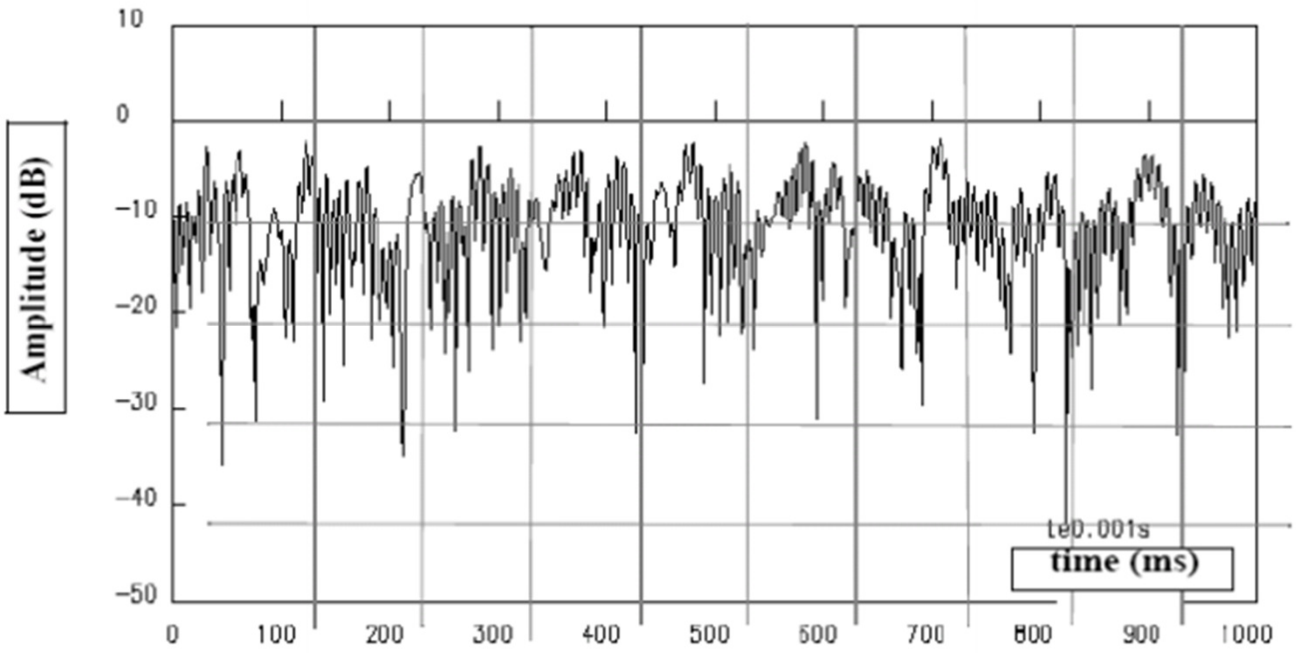
- Data rates are 326.4 Mbps (LTE UMTS/4GSM), 600 Mbps (802.11n), 1.3 Gbps WiFi(802.11ac)
- Real time BER requirement is 1e-6
- Main Channel impairments: ISI (frequency selective channel)
- + time varying fading channel
- + space selective channel

  + deep fades
- Increasing data rates require MIMO systems and more complex channel estimation and receiver algorithms.
- For ISI channels, the near optimal solution is turbo equalization using a detector and advanced ECC such as LDPC.
- In wireless communication the propagation channel is characterized by multipath propagation due to scattering on different obstacles



Transmission on a multipath channel

Fading:



- The received level variations result in SNR variations
- The received level is sensitive to the transmitter and receiver locations

Introduction to Channel Coding

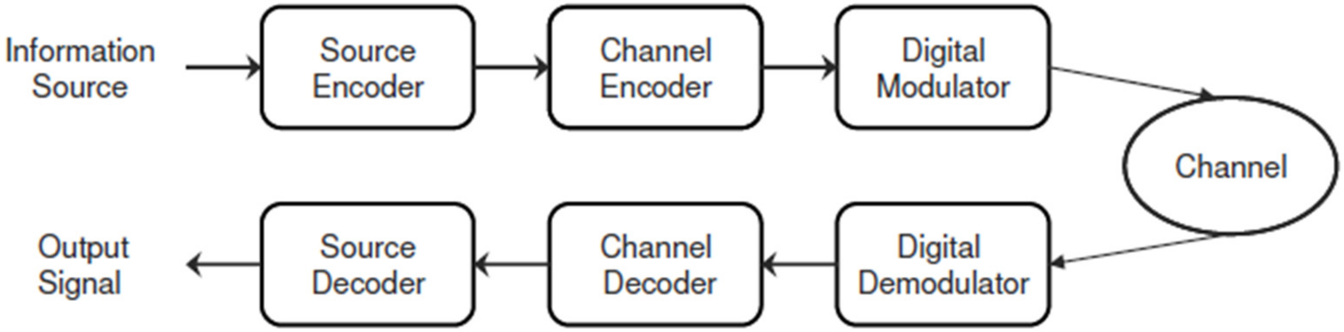


Figure 1: Block Diagram of Communication System

- ❑ Channel Encoder introduces redundancy in transmitted bit stream
- ❑ Channel decoder use the redundancy to correct the errors due to channel impairments and noise.
- ❑ Low-Density Parity-Check (LDPC) Code is the best available error correction code.

LDPC

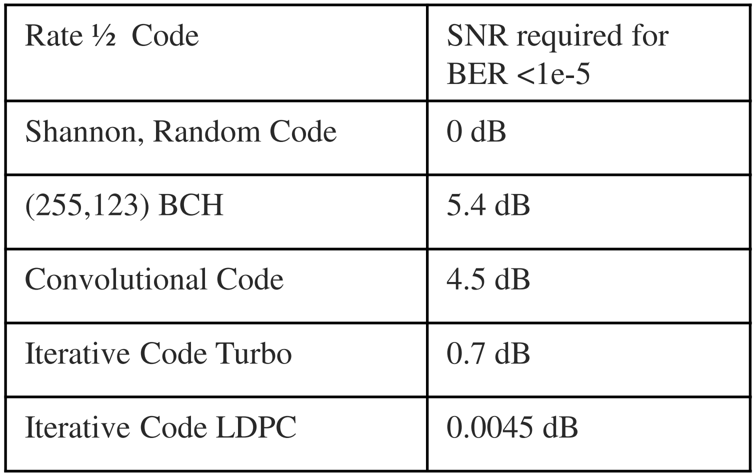
- ❑ Robert Gallager (1960) introduced the concept of low-density parity check codes.
- ❑ Rekindled interest in late 90s after Turbo codes and MacKay and Neal's rediscovery and new constructions of LDPC codes. Construction of practical LDPC happened more recently and several researchers contributed to this.
- ❑ My work :

Academic work: area and energy efficient LDPC decoders for different classes of LDPC codes (PhD work). Texas A&M System owns patents on several aspects of decoder hardware architectures, decoding schedules, code construction constraints.

Industry work: new LDPC encoders, new interleavers suited for LDPC, new queuing systems and error floor mitigation (built using decoders based on PhD work). PhD work (RTL, Matlab models and patents) licensed and commercialized through several industry products.

Standards work: Chair of IEEE Standards Workgroup for Error Correction Coding for Non-Volatile Memories

Table 1: BER performance for different codes



Rate ½ Code	SNR required for BER <1e-5
Shannon, Random Code	0 dB
(255,123) BCH	5.4 dB
Convolutional Code	4.5 dB
Iterative Code Turbo	0.7 dB
Iterative Code LDPC	0.0045 dB

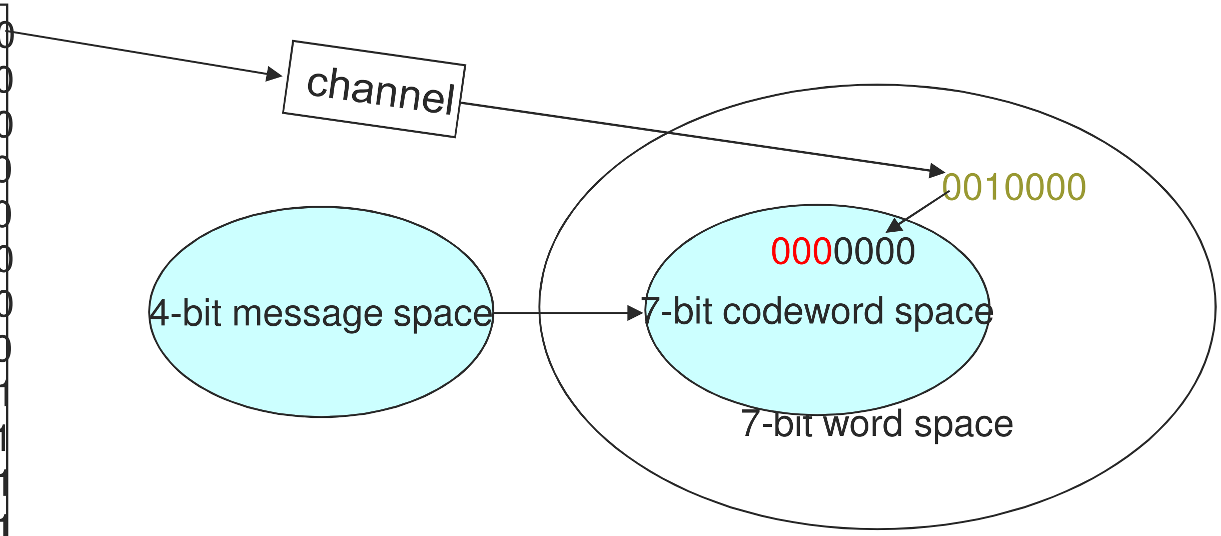
Error Correcting Codes (ECC)

User message bits

Parity bits (redundancy)

4-bit message	7-bit codeword
0000	0000000
1000	1101000
0100	0110100
1100	1011100
0010	1110010
1010	0011010
0110	1000110
1110	0101110
0001	1010001
1001	0111001
0101	1100101
1101	0001101
0011	0100011
1011	1001011
0111	0010111
1111	1111111

Rate = 4 / 7



- The **Hamming distance** between two strings of equal length is the number of positions at which the corresponding bits are different.
- Based on the minimum distance, the received code is decoded to 0000.

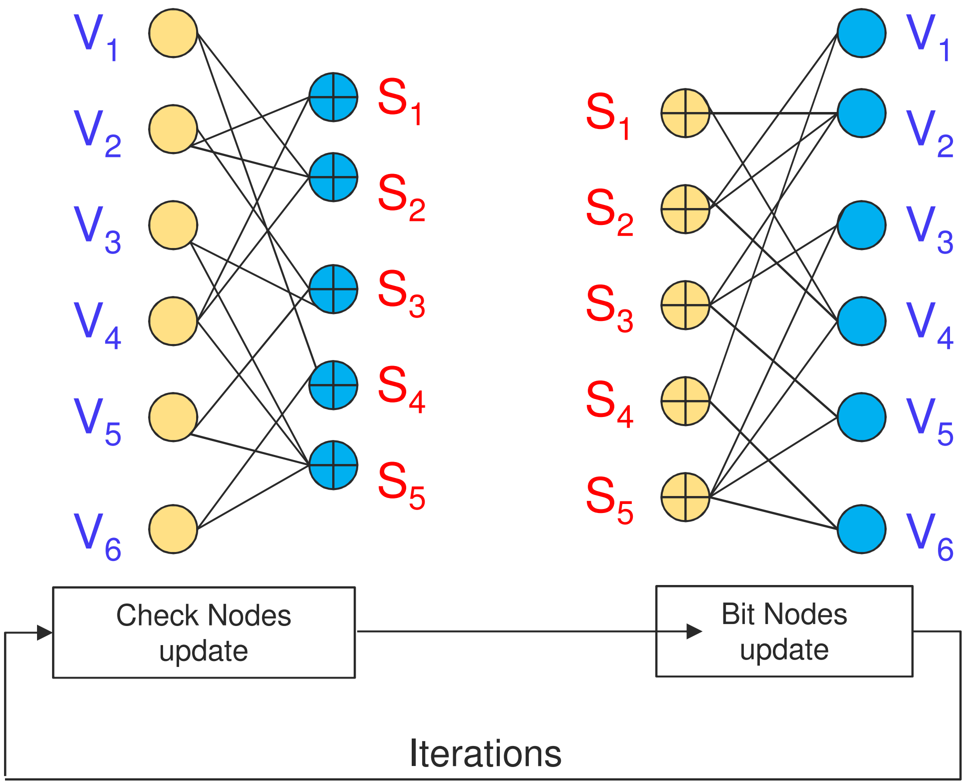
Courtesy: Ned Varnica

LDPC Decoding

Check Nodes

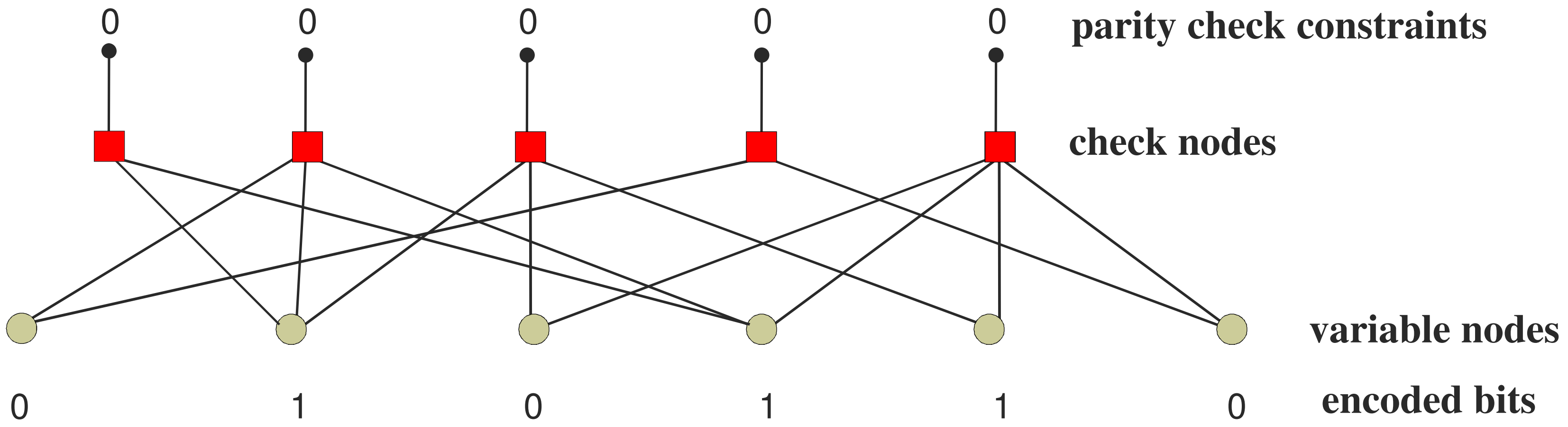
$$\mathbf{H} = \begin{array}{c|cccccc|}
 & V_1 & V_2 & V_3 & V_4 & V_5 & V_6 \\
\hline
S_1 & 0 & 1 & 0 & 1 & 0 & 0 \\
S_2 & 1 & 1 & 0 & 1 & 0 & 0 \\
S_3 & 0 & 1 & 1 & 0 & 1 & 0 \\
S_4 & 1 & 0 & 0 & 0 & 0 & 1 \\
S_5 & 0 & 0 & 1 & 1 & 1 & 1
\end{array}$$

Variable Nodes



- Variable nodes correspond to the soft information of received bits.
- Check nodes describe the parity equations of the transmitted bits.
- Eg. v2+v4= 0; v1+v2+v4 =0 and so on.
- The decoding is successful when all the parity checks are satisfied (i.e. zero).

Representation on Bi-Partite (Tanner) Graphs



Each bit “1” in the parity check matrix is represented by an edge between corresponding variable node (column) and check node (row)

$$\mathbf{H} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Hard Decision Decoding: Bit-Flipping Decoder

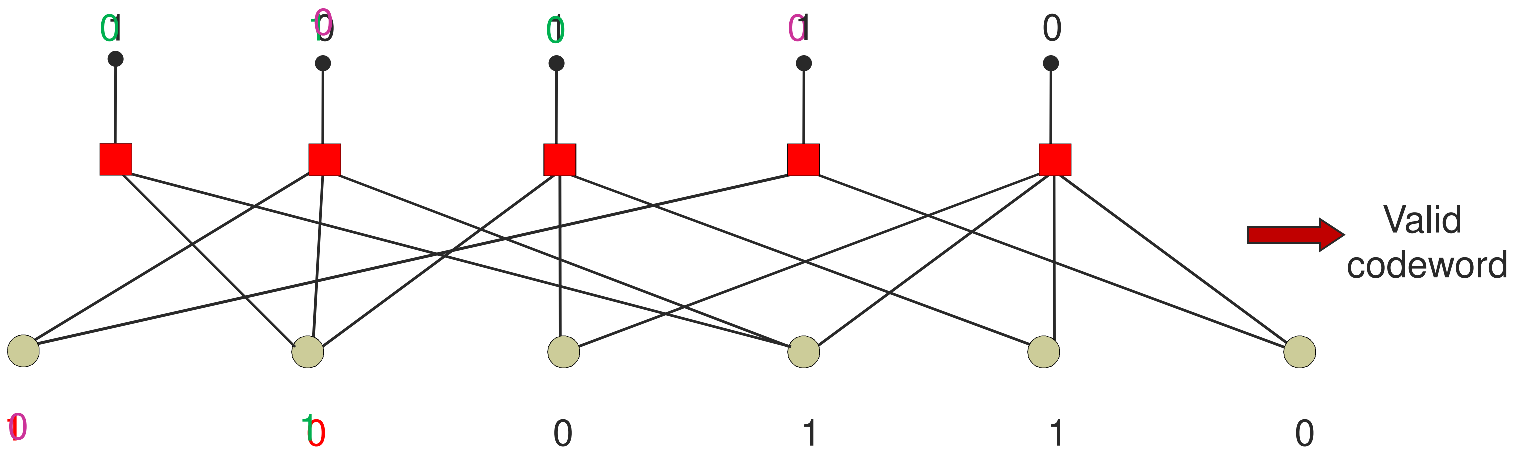
- Decision to flip a bit is made based on the number of unsatisfied checks connected to the bit

First step

End of first step

Second step

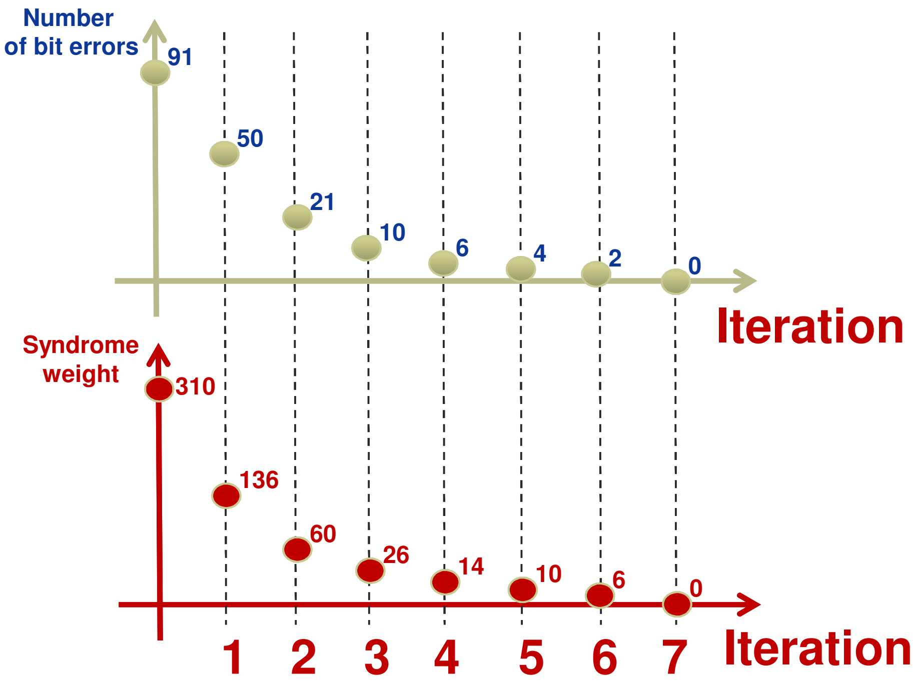
Expected Message: 010110
Received Message: 100110



The [illegible] neighbors

Bit-Flipping Decoder Progress on a Large LDPC Code

- **Decoder starts with a relatively large number of errors**
- **As decoder progresses, some bits are flipped to their correct values**
- **Syndrome weight improves**
  - **As this happens, it becomes easier to identify the bits that are erroneous and to flip the remaining error bits to actual (i.e. written / transmitted) values**



Soft Information Representation

- The information used in soft LDPC decoder represents bit reliability metric, LLR (log-likelihood-ratio)

$$LLR(b_i) = \log\left(\frac{P(b_i = 0)}{P(b_i = 1)}\right)$$

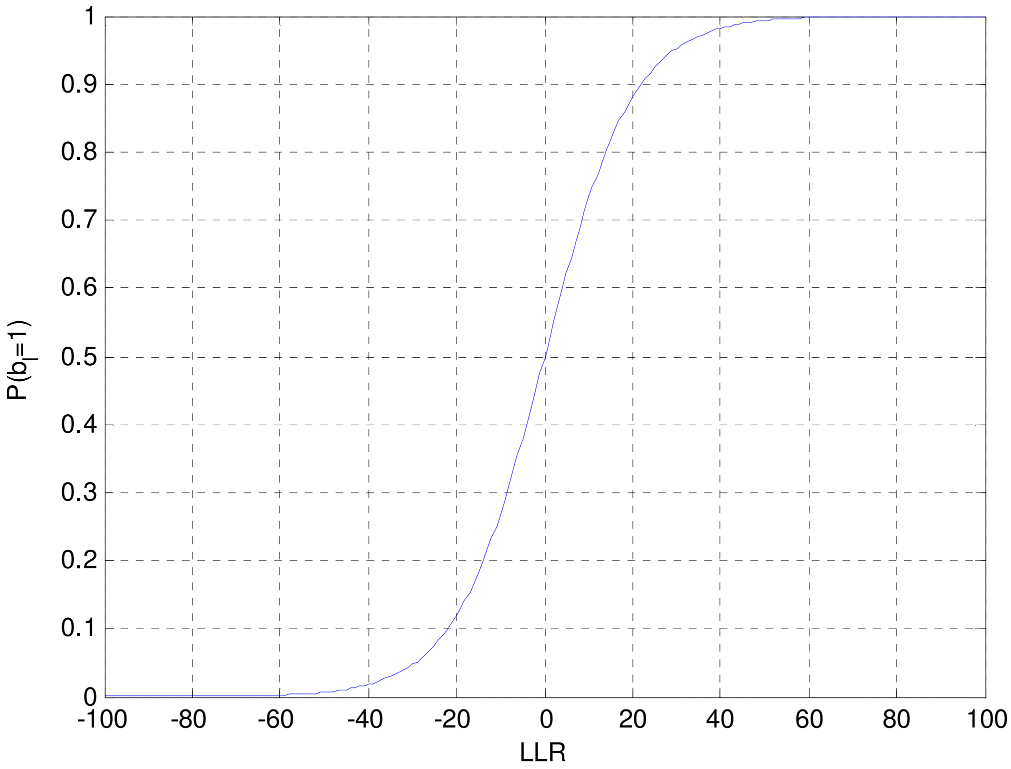
- The choice to represent reliability information in terms of LLR as opposed to probability metric is driven by HW implementation consideration

- The following chart shows how to convert LLRs to probabilities (and vice versa)

Soft Information Representation

- Bit LLR>0 implies bit=0 is more likely, while LLR<0 implies bit=1 is more likely

$P(b_i = 0)$

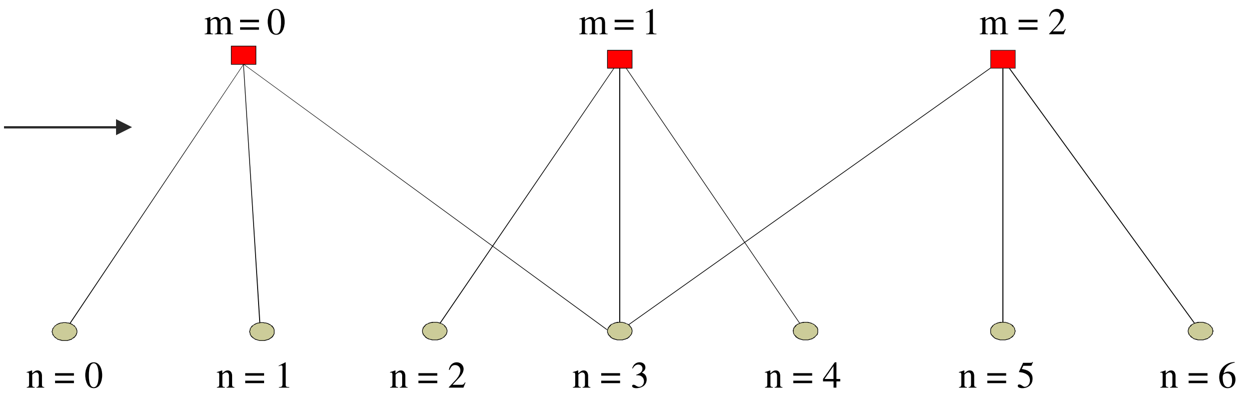


More likely, the bit is 1 → More likely, the bit is 0

Soft Message Passing Decoder

- LDPC decoding is carried out via message passage algorithm on the graph corresponding to a parity check matrix **H**

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \longrightarrow$$



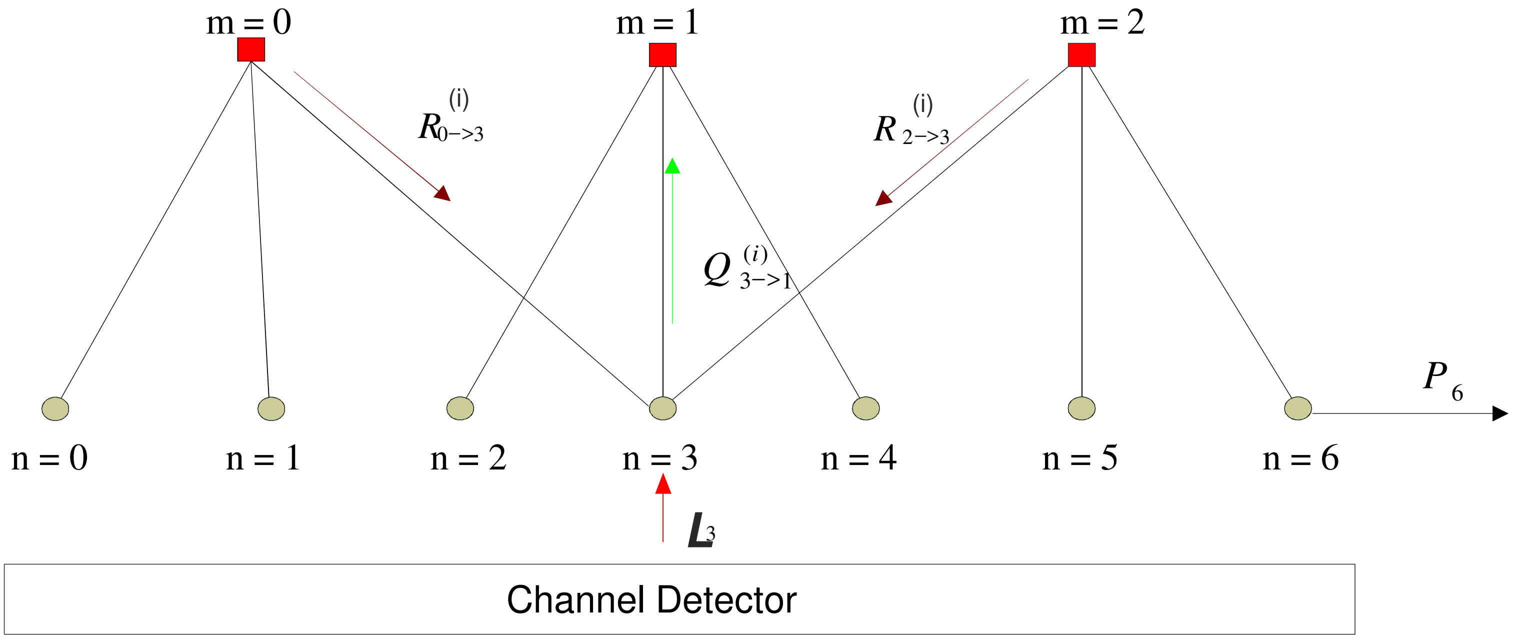
Check nodes (rows of **H**)

Bit nodes (columns of **H**)

- The messages are passed along the edges of the graph
  - First from the bit nodes to check nodes
  - And then from check nodes back to bit nodes

Soft LDPC Decoder

- There are four types of LLR messages
  - Message from the channel to the n-th bit node, L_n
  - Message from n-th bit node to the m-th check node $Q_{n->m}^{(i)}$ or simply $Q_{nm}^{(i)}$
  - Message from the m-th check node to the n-th bit node $R_{m->n}^{(i)}$ or simply $R_{mn}^{(i)}$
  - Overall reliability information for n-th bit-node P_n



Soft LDPC Decoder (cont.)

- Message passing algorithms are iterative in nature
- One iteration consists of
  - upward pass (bit node processing/variable node processing): bit nodes pass the information to the check nodes
  - downward pass (check node processing): check nodes send the updates back to bit nodes
- The process then repeats itself for several iterations

Courtesy: Joint presentation with Ned Varnica at Flash memory summit, slides 24-31 prepared by Ned.

Soft LDPC Decoder, cont.

Notation used in the equations

x_n is the transmitted bit n ,

L_n is the initial LLR message for a bit node (also called as variable node) n , received from channel/detector

P_n is the overall LLR message for a bit node n ,

$\widehat{x}_n$ is the decoded bit n (hard decision based on P_n) ,

[Frequency of P and hard decision update depends on decoding schedule]

$\mathrm{M}(n)$ is the set of the neighboring check nodes for variable node n ,

$\mathrm{N}(m)$ is the set of the neighboring bit nodes for check node m .

For the i^{th} iteration,

$Q_{nm}^{(i)}$ is the LLR message from bit node n to check node m ,

$R_{mn}^{(i)}$ is the LLR message from check node m to bit node n .

Soft LDPC Decoder, cont.

(A) check node processing: for each m and $n \in \mathrm{N}(m)$,

$$R_{mn}^{(i)} = \delta_{mn}^{(i)} \kappa_{mn}^{(i)} \qquad (1)$$

$$\kappa_{mn}^{(i)} = \left| R_{mn}^{(i)} \right| = \min_{n' \in \mathrm{N}(m) \setminus n} \left| Q_{n'm}^{(i-1)} \right| \qquad (2)$$

The sign of check node message $R_{mn}^{(i)}$ is defined as

$$\delta_{mn}^{(i)} = \left(\prod_{n' \in \mathrm{N}(m) \setminus n} \mathrm{sgn}\left(Q_{n'm}^{(i-1)} \right) \right) \qquad (3)$$

where $\delta_{mn}^{(i)}$ takes value of $+1$ or -1

Soft LDPC Decoder, cont.

(B) *Variable-node processing*: for each n and $m \in M(n)$:

$$Q_{nm}^{(i)} = L_n + \sum_{m' \in \mathrm{M}(n) \backslash m} R_{m'n}^{(i)} \qquad (4)$$

(C) P Update and Hard Decision

$$P_n = L_n + \sum_{m \in M(n)} R_{mn}^{(i)} \qquad (5)$$

A hard decision is taken where $\hat{x}_n = 0$ if $P_n \geq 0$, and $\hat{x}_n = 1$ if $P_n < 0$.

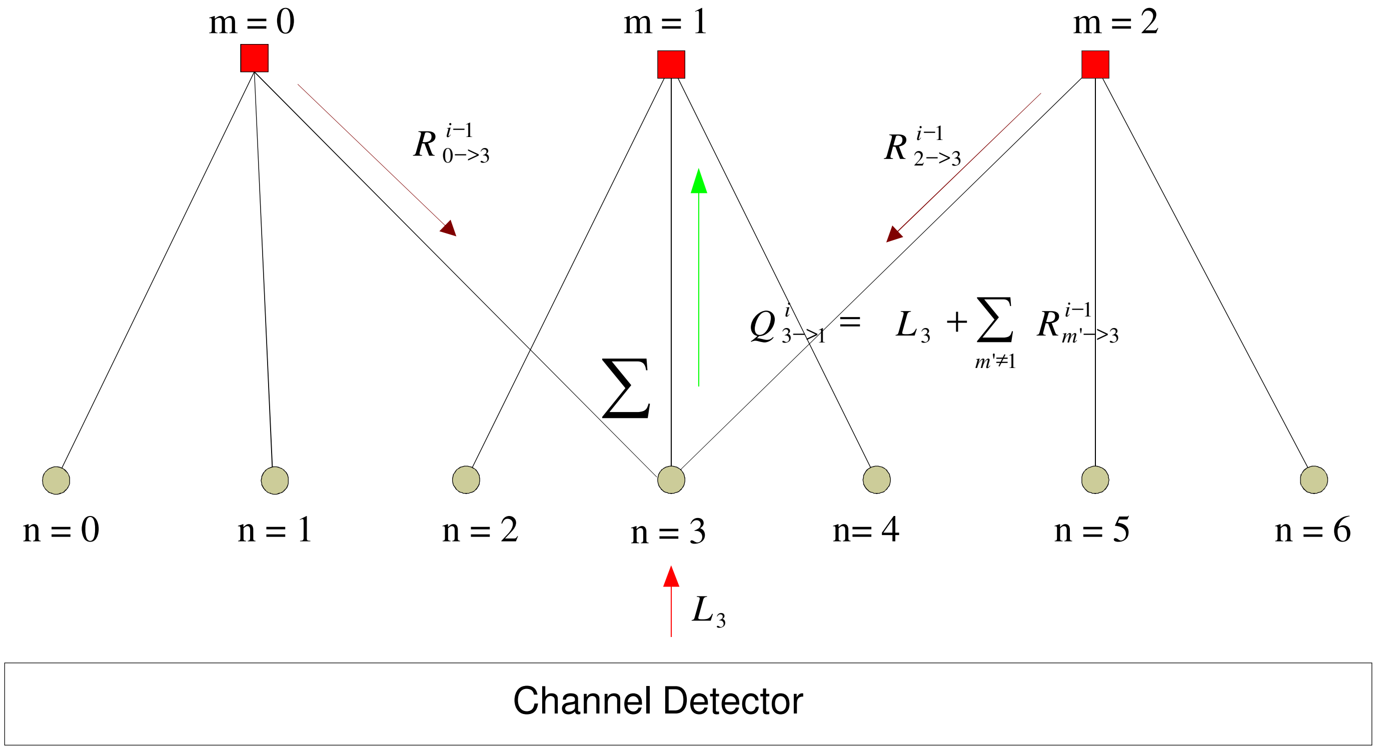
If $\hat{x}_n H^T = 0$, the decoding process is finished with $\hat{x}_n$ as the decoder output; otherwise, repeat steps (A) to (C).

If the decoding process doesn't end within some maximum iteration, stop and output error message.

Scaling or offset can be applied on R messages and/or Q messages for better performance.

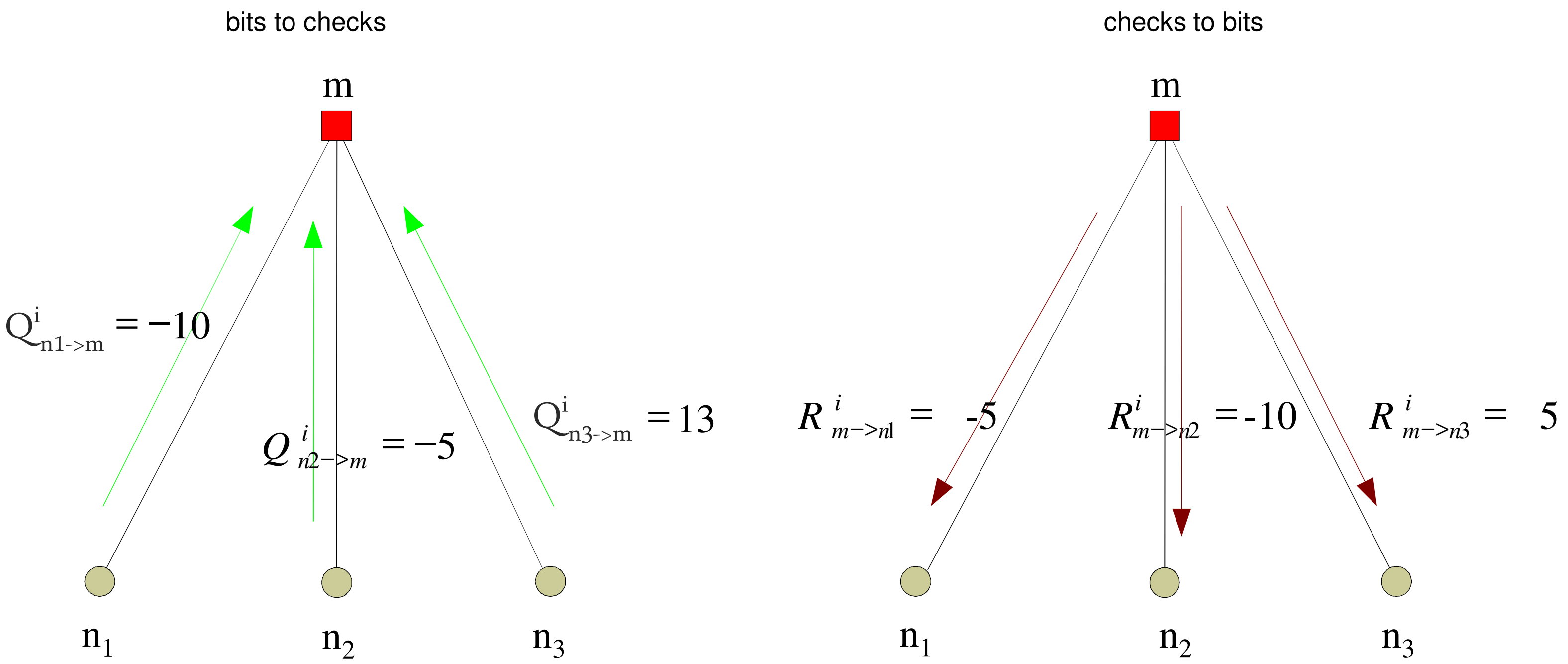
Soft LDPC Decoder (cont.)

- Bits-to-checks pass: $Q_{n->m}^{(i)}$: n-th bit node sums up all the information it has received at the end of last iteration, except the message that came from m-th check node, and sends it to m-th check node
  - At the beginning of iterative decoding all R messages are initialized to zero



Soft LDPC Decoder (cont.)

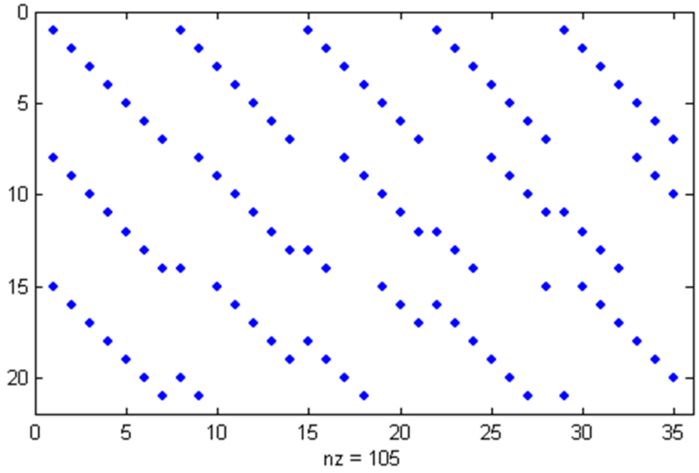
- Checks-to-bits pass:
  - Check node has to receive the messages from all participating bit nodes before it can start sending messages back
  - Least reliable of the incoming extrinsic messages determines magnitude of check-to-bit message.
  - Sign is determined so that modulo 2 sum is satisfied


bits to checks
m
$Q^i_{n1\text{->}m} = -10$
$Q^i_{n2\text{->}m} = -5$
$Q^i_{n3\text{->}m} = 13$
n_1
n_2
n_3
checks to bits
m
$R^i_{m\text{->}n1} = -5$
$R^i_{m\text{->}n2} = -10$
$R^i_{m\text{->}n3} = 5$
n_1
n_2
n_3

Example QC-LDPC Matrix

$$H = \begin{bmatrix} I & I & I & \dots & I \\ I & \sigma & \sigma^{2} & \dots & \sigma^{r-1} \\ I & \sigma^{2} & \sigma^{4} & \dots & \sigma^{2(r-1)} \\ : & & & & \\ I & \sigma^{c-1} & \sigma^{(c-1)2} & \dots & \sigma^{(c-1)(r-1)} \end{bmatrix}$$

$$\sigma = \begin{bmatrix} 0 & 0 & \dots & 0 & 1 \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & .0 & 0 \\ : & & & & \\ 0 & 0 & \dots & .1 & 0 \end{bmatrix}$$



Example H Matrix, Array LDPC

r row/ check node degree=5

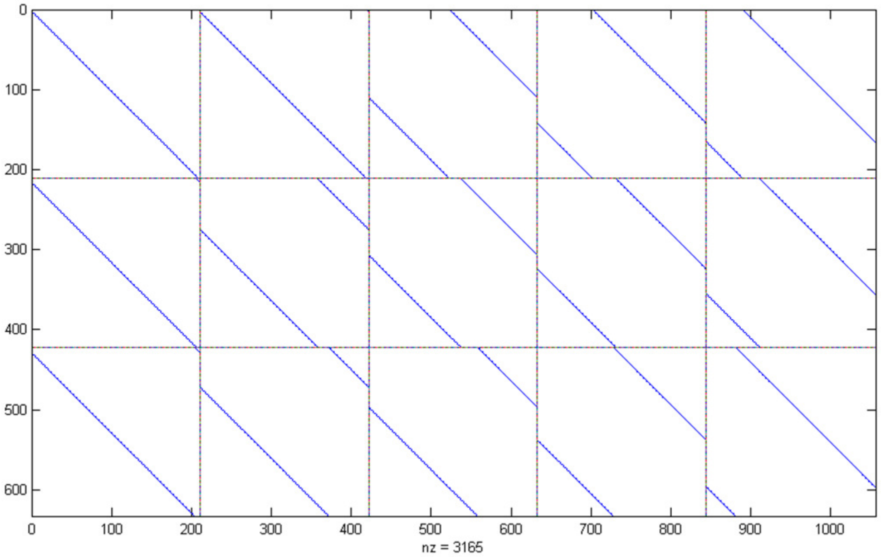
c columns/variable node degree=3

Sc circulant size=7

*N=Sc*r=35*

Example QC-LDPC Matrix

$$S_{3\times 5} = \begin{bmatrix} 2 & 3 & 110 & 142 & 165 \\ 5 & 64 & 96 & 113 & 144 \\ 7 & 50 & 75 & 116 & 174 \end{bmatrix}$$



Example H Matrix

r row degree=5

c column degree =3

Sc circulant size=211

*N=Sc*r=1055*

Decoder Architectures

- Parallelization is good-but comes at a steep cost for LDPC.
- Fully Parallel Architecture:
- All the check updates in one clock cycle and all the bit updates in one more clock cycle.
- Huge Hardware resources and routing congestion.
- The chip area is 52.5 sq mm.

decoder was designed for a maximum clock frequency of 64MHz under worst case slow operating conditions. The layout of the decoder is shown in Fig. 7. The chip area is 7.5mm × 7.0mm and the utilization of 50% was limited by routing congestion.

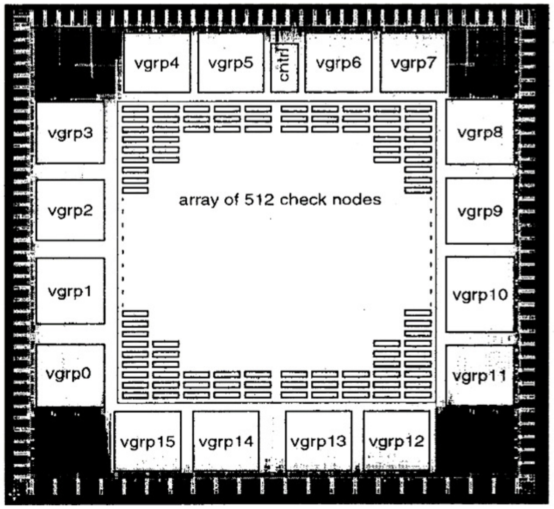


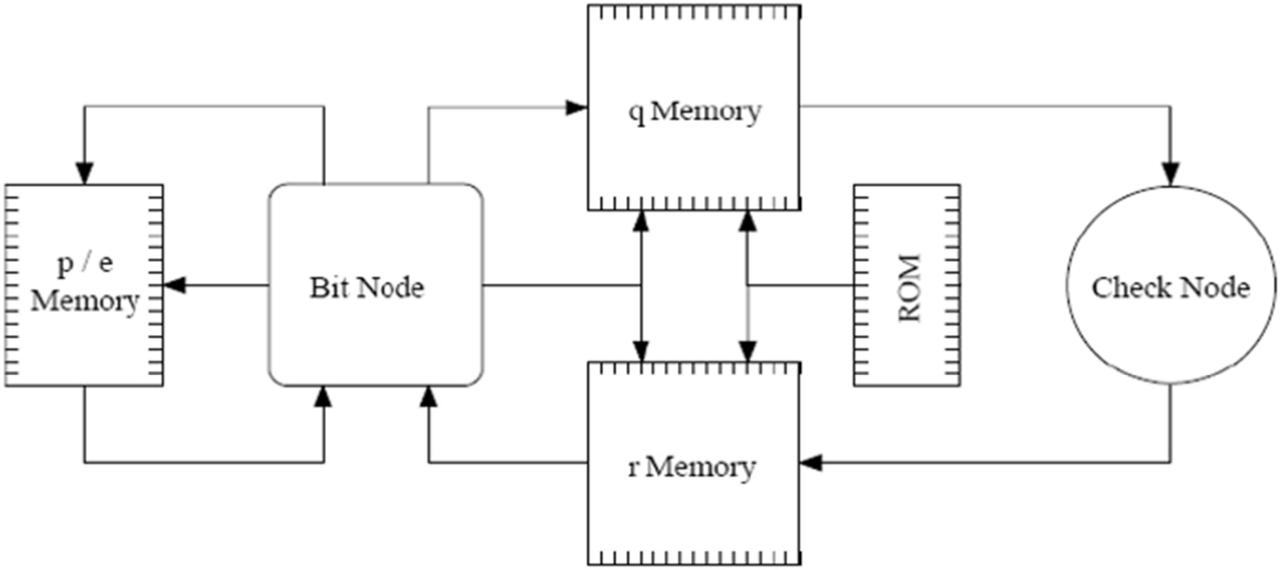
Fig. 7. Layout of the parallel 1024-bit, rate-1/2, soft decision LDPC decoder.

Work from Agere/LSI:

C. Howland and A. Blanksby, “A 220 mW 1 Gb/s 1024-bit rate-1/2 low density parity check code decoder,” Proc. 2001, IEEE Conf. Custom Integrated Circuits, pp. 293–296, 2001.

Decoder Architectures, Serial

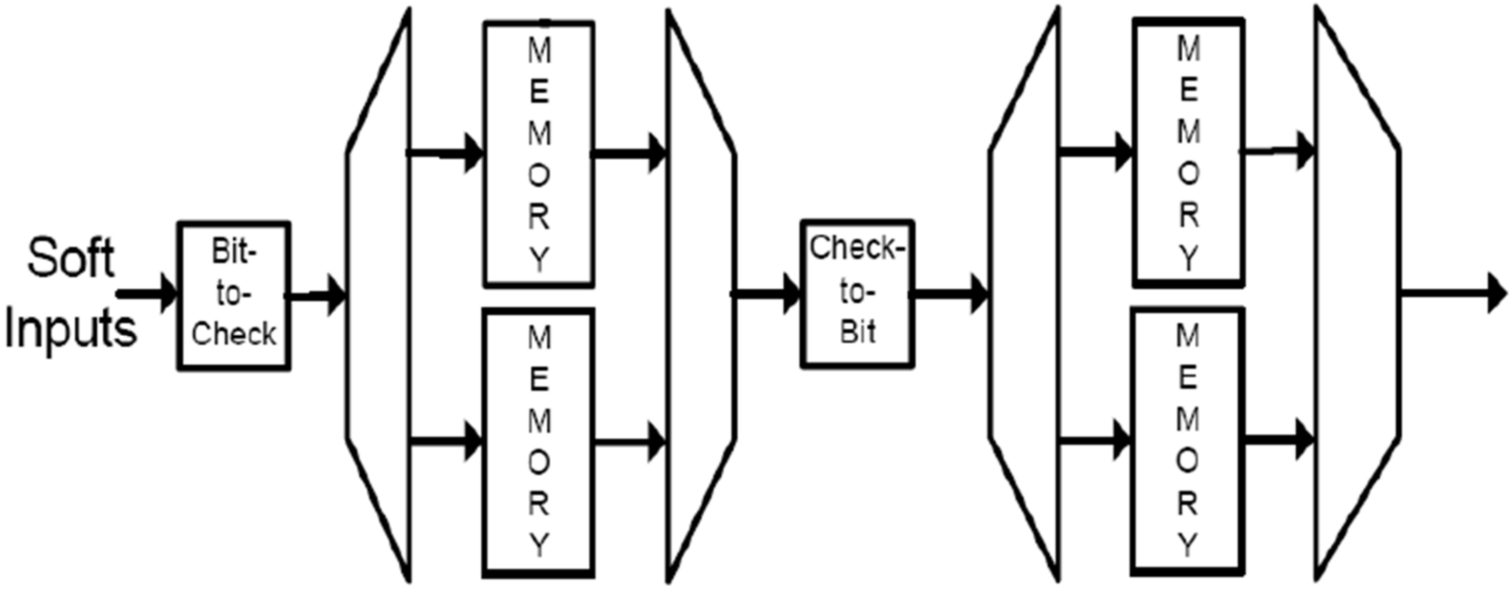
- Check updates and bit updates in a serial fashion.
- Huge Memory requirement. Memory in critical path.



Serial Architecture

[1] Levine,.etal Implementation of near Shannon limit error-correcting codes using reconfigurable hardware IEEE Field-Programmable Custom Computing Machines, 2000

Decoder Architectures, Serial



Serialized and fully pipelined implementation requires two memory buffers per stage, alternating between read/write.

Serial Architecture.

[2] E. Yeo, "VLSI architectures for iterative decoders in magnetic recording channels," IEEE Trans. Magnetics, vol.37, no.2, pp. 748-55, March 2001.

Semi-parallel Architectures

- Check updates and bit updates using several units.
- Partitioned memory by imposing structure on H matrix.
- There are several semi-parallel architectures proposed.
- Initial semi-parallel architectures were complex and very low throughput. Needed huge amounts of memory.

On-the-fly Computation

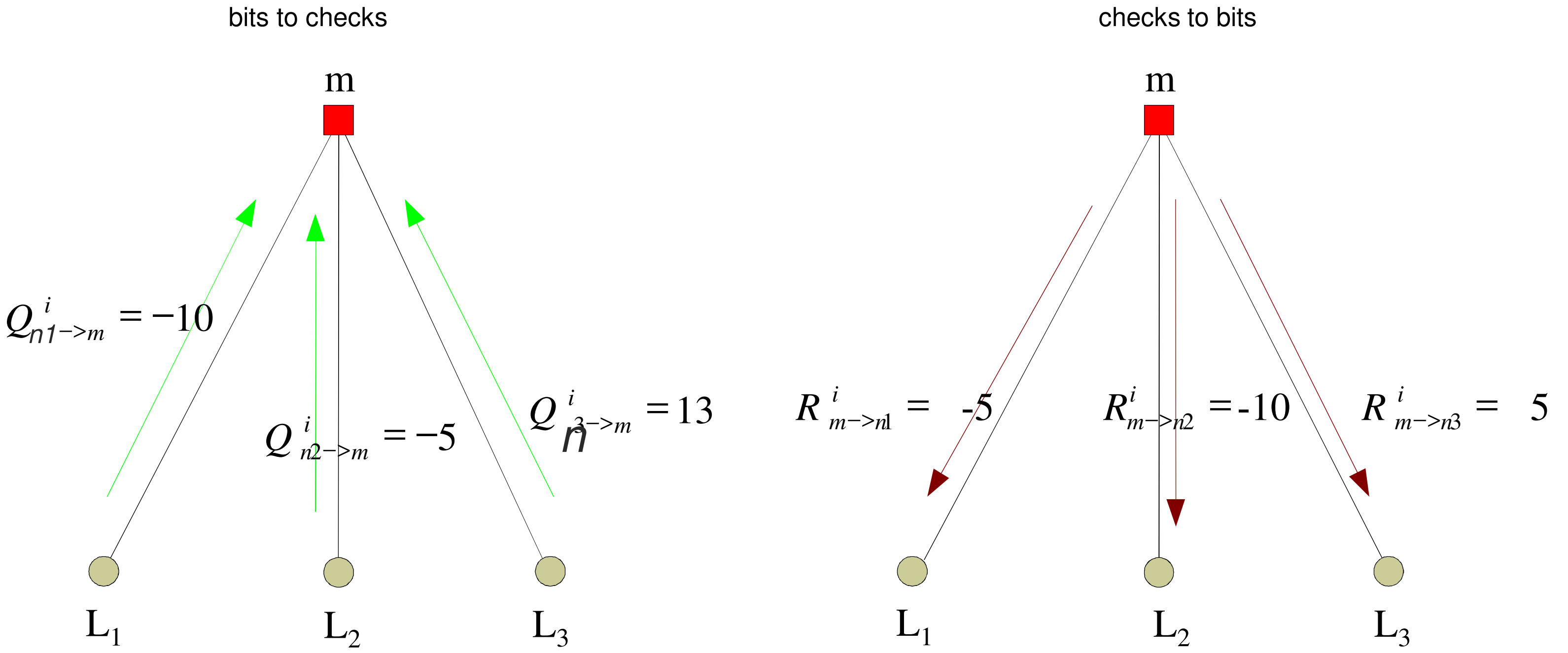
Our previous research ([1-13]) introduced the following concepts to LDPC decoder implementation. References P1-P5 are more comprehensive and are the basis for this presentation.

1. Block serial scheduling
2. Value-reuse,
3. Scheduling of layered processing,
4. Out-of-order block processing,
5. Master-slave router,
6. Dynamic state,
7. Speculative Computation
8. Run-time Application Compiler [support for different LDPC codes with in a class of codes. Class:802.11n,802.16e,Array, etc. Off-line re-configurable for several regular and irregular LDPC codes]

All these concepts are termed as On-the-fly computation as the core of these concepts are based on minimizing memory and re-computations by employing just in-time scheduling. For this presentation, we will focus on Value-reuse and Out-of-order block processing.

Check Node Update

$$\kappa^{(i)}_{m\to l} = \left|R^{(i)}_{m\to l}\right| = \min_{l' \in \mathrm{N}(m)\setminus l} \left|Q^{(i-1)}_{l'\to m}\right|$$



Check Node Unit (CNU) Design

$$\kappa_{mn}^{(i)} = \left|R_{mn}^{(i)}\right| = \min_{n' \in \mathrm{N}(m) \backslash n} \left|Q_{n'm}^{(i-1)}\right| \qquad (2)$$

The above equation (2) can be reformulated as the following set of equations.

$$M1_{m}^{(i)} = \min_{n' \in \mathrm{N}(m)} \left|Q_{mn'}^{(i-1)}\right| \qquad (6)$$

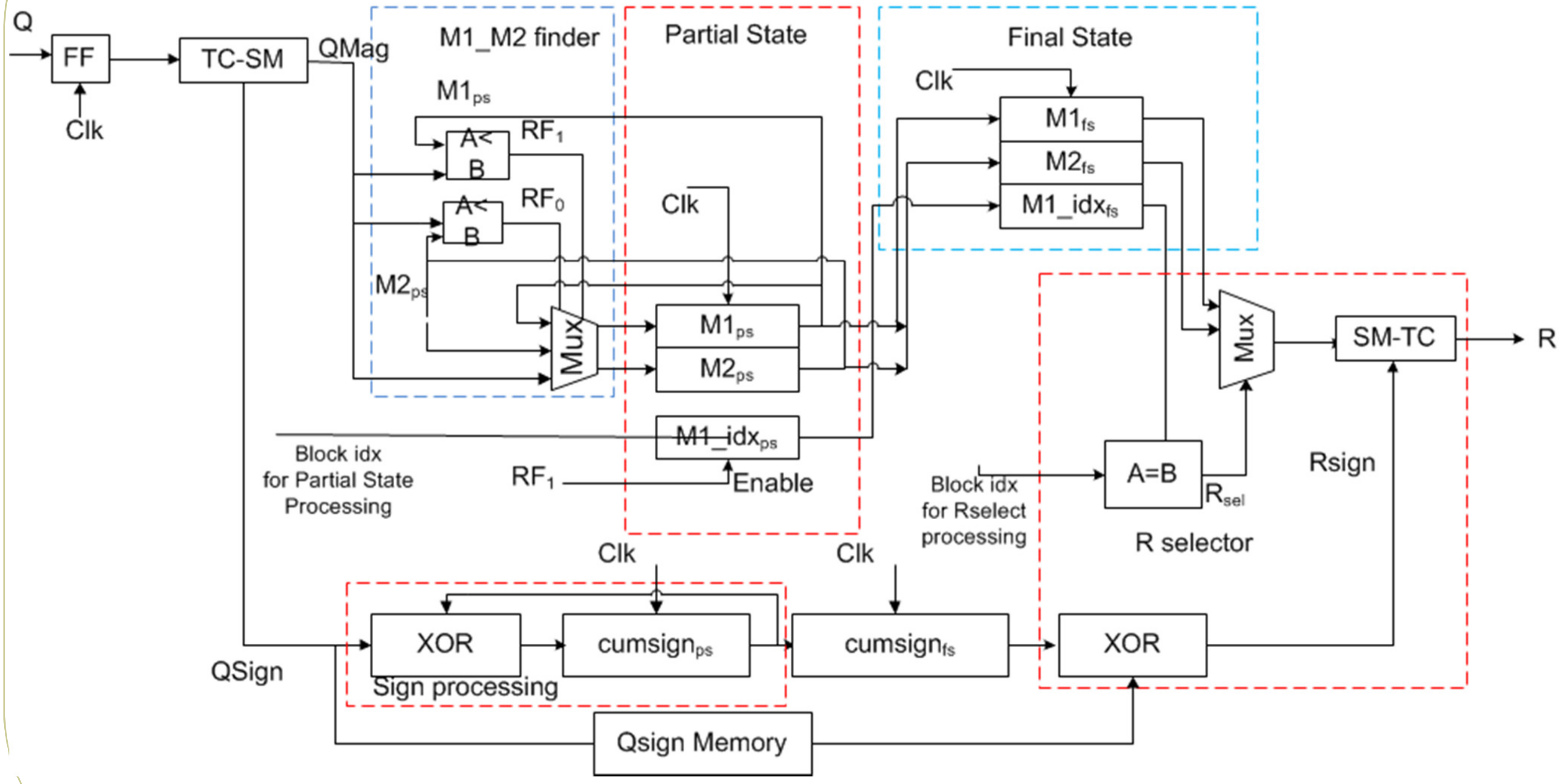
$$M2_{m}^{(i)} = 2nd \min_{n' \in \mathrm{N}(m)} \left|Q_{mn'}^{(i-1)}\right| \qquad (7)$$

$$k = Min1Index \qquad (8)$$

$$\begin{aligned}\kappa_{mn}^{(i)} &= M1_{m}^{(i)}, \ \forall n \in \mathrm{N}(m) \backslash k \\ &= M2_{m}^{(i)}, n = k \qquad (9)\end{aligned}$$

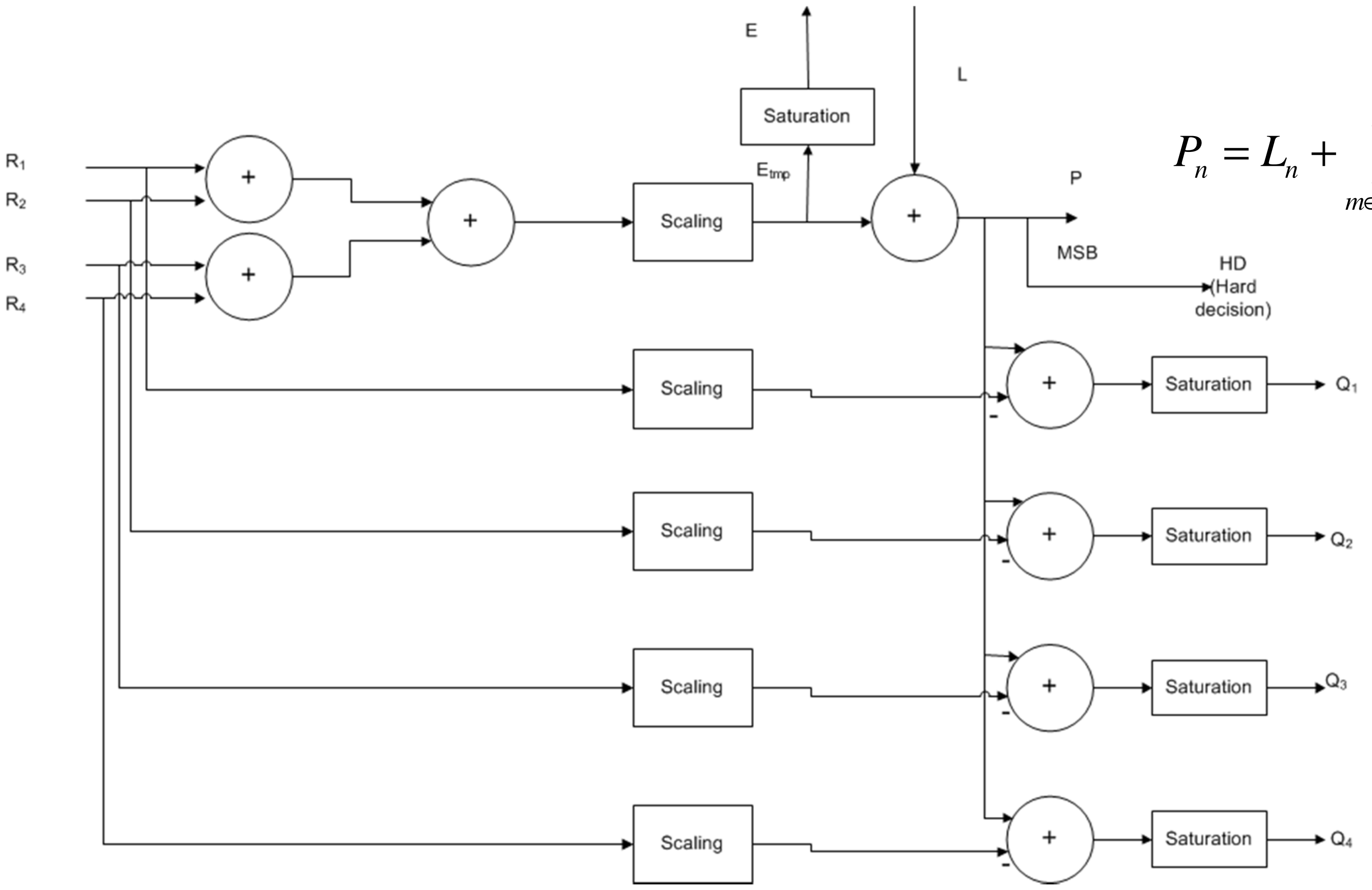
- Simplifies the number of comparisons required as well as the memory needed to store CNU outputs.
- Additional design choices: The correction has to be applied to only two values instead of distinct values. Need to apply 2's complement to only 1 or 2 values instead of values at the output of CNU.

CNU Micro Architecture for min-sum



5/1/2014

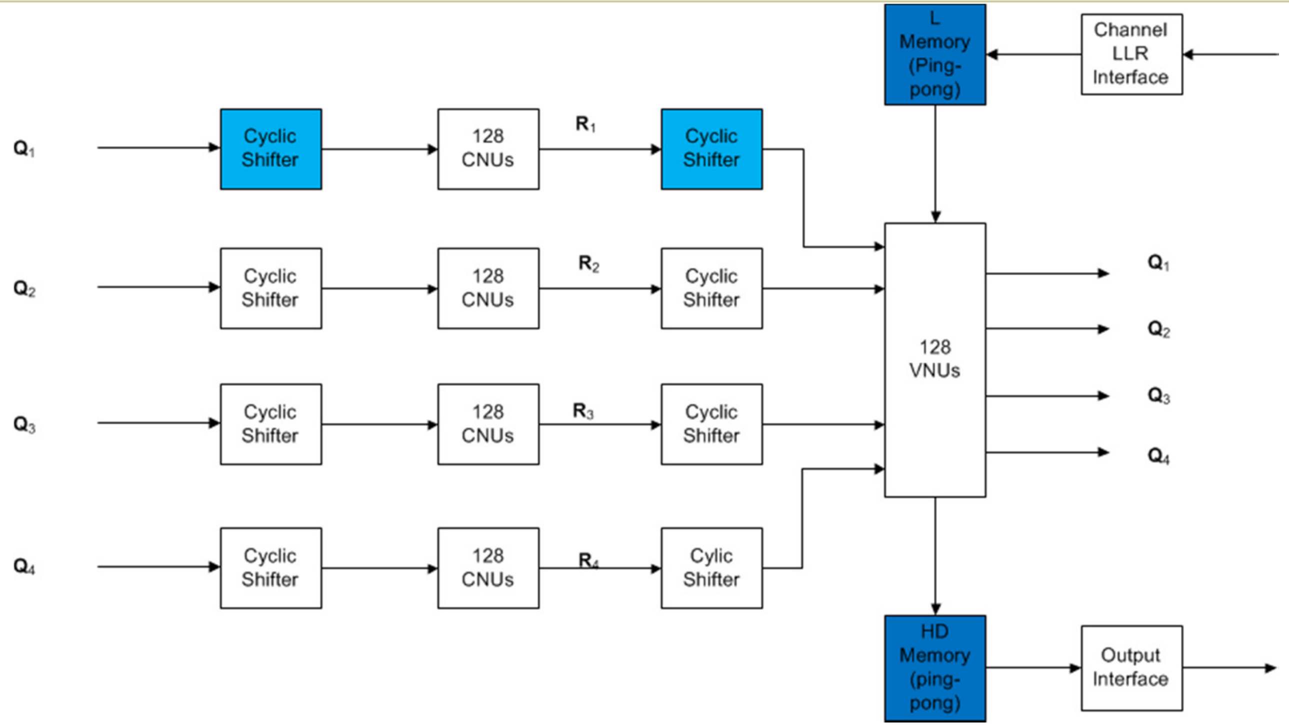
VNU Micro Architecture



$$P_n = L_n + \sum_{m \in M(n)} R_{mn}^{(i)}$$

$$Q_{nm}^{(i)} = L_n + \sum_{m' \in \mathrm{M}(n) \backslash m} R_{m'n}^{(i)}$$

Non-Layered Decoder Architecture



Supported H matrix parameters

r row degree=36

c column degree =4

Sc ciruclant size=128

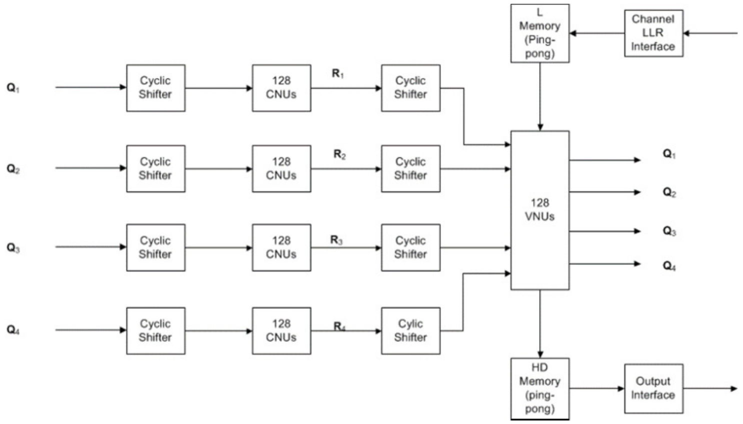
*N=Sc*r=4608*

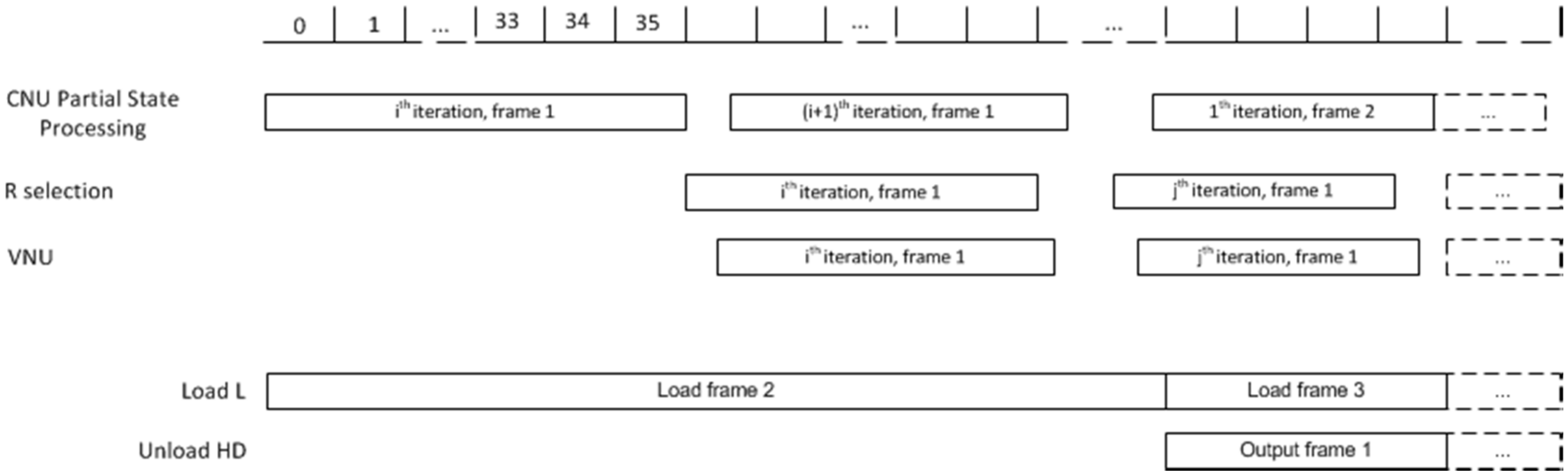
*L Memory=> Depth 36, Width=128*5*

HD Memory=> Depth 36, Width=128

Possible to remove the shifters (light-blue) by re-arranging H matrix's first layer to have zero shift coefficients.

Pipeline for Non-layered Decoder




0
1
...
33
34
35
...
...
CNU Partial State Processing
i^{th} iteration, frame 1
$(i+1)^{th}$ iteration, frame 1
1^{th} iteration, frame 2
...
R selection
i^{th} iteration, frame 1
j^{th} iteration, frame 1
...
VNU
i^{th} iteration, frame 1
j^{th} iteration, frame 1
...
Load L
Load frame 2
Load frame 3
...
Unload HD
Output frame 1
...

Layered Decoder Architecture

Optimized Layered Decoding with algorithm transformations for reduced memory and computations

$\vec{R}_{l,n}^{(0)} = 0, \vec{P}_n = \vec{L}_n^{(0)}$ [Initialization for each new received data frame], (9)

$\forall i = 1,2,\cdots,it_{\max}$ [Iteration loop],

$\forall l = 1,2,\cdots,j$ [Sub-iteration loop],

$\forall n = 1,2,\cdots,k$ [Block column loop],

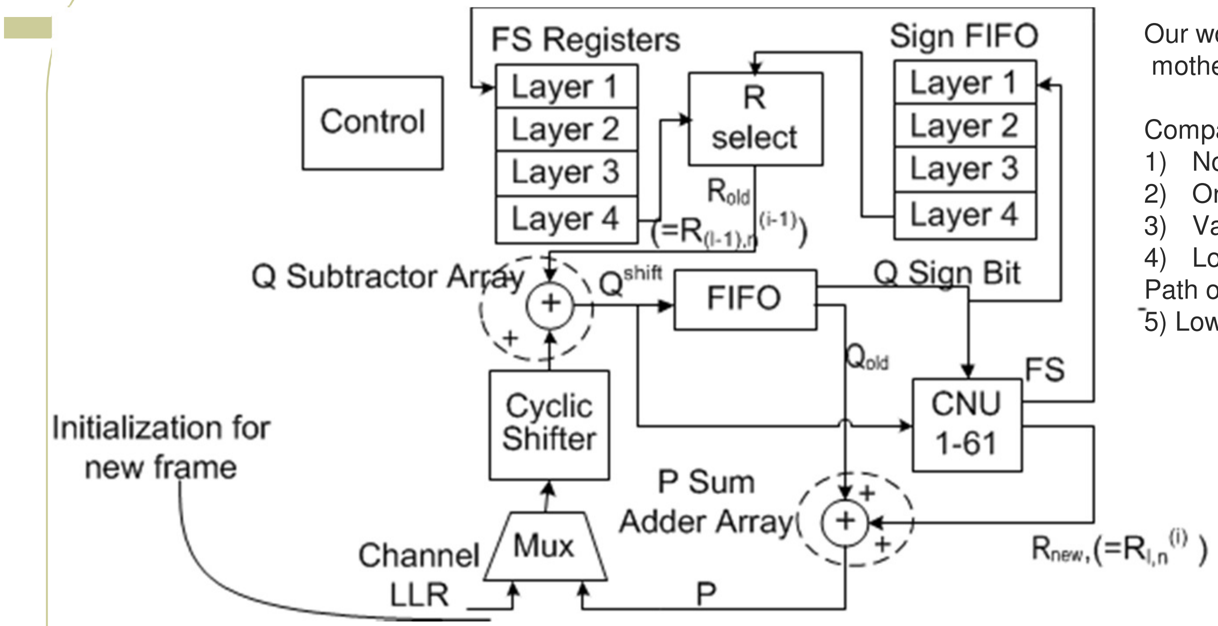
$$\left[\vec{Q}_{l,n}^{(i)}\right]^{S(l,n)} = \left[\vec{P}_n\right]^{S(l,n)} - \vec{R}_{l,n}^{(i-1)}, \tag{10}$$

$$\vec{R}_{l,n}^{(i)} = f\left(\left[\vec{Q}_{l,n'}^{(i)}\right]^{S(l,n')}, \forall n' = 1,2,\cdots,k\right), \tag{11}$$

$$\left[\vec{P}_n\right]^{S(l,n)} = \left[\vec{Q}_{l,n}^{(i)}\right]^{S(l,n)} + \vec{R}_{l,n}^{(i)}, \tag{12}$$

where the vectors $\vec{R}_{l,n}^{(i)}$ and $\vec{Q}_{l,n}^{(i)}$ represent all the R and Q messages in each $p \times p$ block of the H matrix,
$s(l,n)$ denotes the shift coefficient for the block in l^{th} block row and n^{th} block column of the H matrix.
$\left[\vec{Q}_{l,n}^{(i)}\right]^{S(l,n)}$ denotes that the vector $\vec{Q}_{l,n}^{(i)}$ is cyclically shifted up by the amount $s(l,n)$
k is the check-node degree of the block row.
A negative sign on $s(l,n)$ indicates that it is a cyclic down shift (equivalent cyclic left shift).
$f(\cdot)$ denotes the check-node processing, which embodiments implement using, for example, a Bahl-Cocke-Jelinek-Raviv algorithm (“BCJR”) or sum-of-products (“SP”) or Min-Sum with scaling/offset.

Layered Decoder Architecture



Our work proposed this for H matrices with regular mother matrices.

Compared to other work, this work has several advantages

1) No need of separate memory for P.
2) Only one shifter instead of 2 shifters
3) Value-reuse is effectively used for both Rnew and Rold
4) Low complexity data path design-with no redundant data Path operations.
5) Low complexity CNU design.

$$\vec{R}_{l,n}^{(i)} = f\left(\left[\vec{Q}_{l,n'}^{(i)}\right]^{S(l,n')}, \ \forall n' = 1,2,\cdots,k\right)$$

$$\left[\vec{Q}_{l,n}^{(i)}\right]^{S(l,n)} = \left[\vec{P}_n\right]^{S(l,n)} - \vec{R}_{l,n}^{(i-1)}$$

$Q=P-R_{old}$

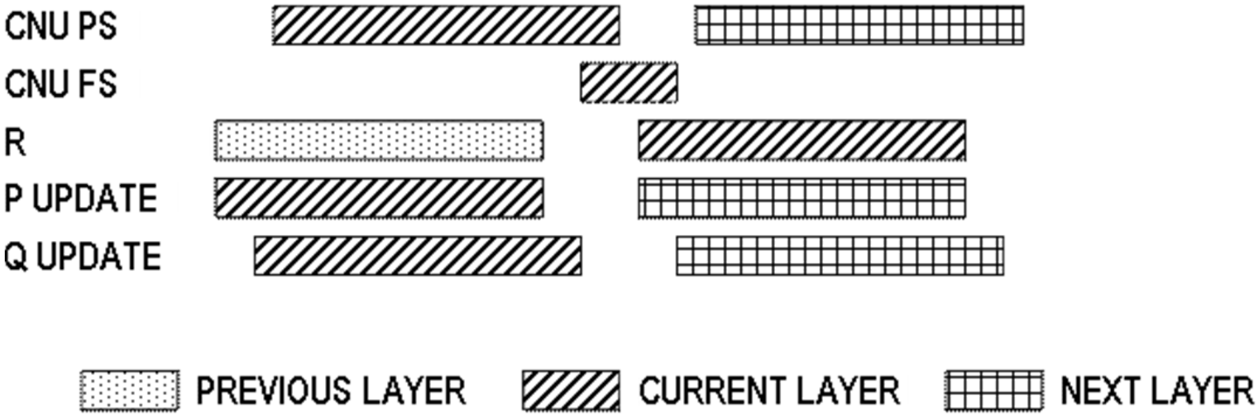
$$\left[\vec{P}_n\right]^{S(l,n)} = \left[\vec{Q}_{l,n}^{(i)}\right]^{S(l,n)} + \vec{R}_{l,n}^{(i)}$$

$P=Q_{old}+R_{new}$

Data Flow Diagram

$P=Q_{old}+R_{new}$

$Q=P-R_{old}$



Irregular QC-LDPC H Matrices

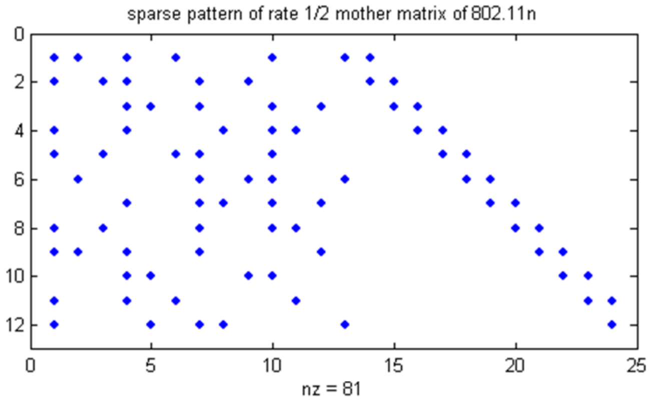
$$\mathbf{H} = \begin{bmatrix} \mathbf{P}_{0,0} & \mathbf{P}_{0,1} & \mathbf{P}_{0,2} & \cdots & \mathbf{P}_{0,n_b-2} & \mathbf{P}_{0,n_b-1} \\ \mathbf{P}_{1,0} & \mathbf{P}_{1,1} & \mathbf{P}_{1,2} & \cdots & \mathbf{P}_{1,n_b-2} & \mathbf{P}_{1,n_b-1} \\ \mathbf{P}_{2,0} & \mathbf{P}_{2,1} & \mathbf{P}_{2,2} & \cdots & \mathbf{P}_{2,n_b-2} & \mathbf{P}_{0,n_b-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \mathbf{P}_{m_b-1,0} & \mathbf{P}_{m_b-1,1} & \mathbf{P}_{m_b-1,2} & \cdots & \mathbf{P}_{m_b-1,n_b-2} & \mathbf{P}_{m_b-1,n_b-1} \end{bmatrix} = \mathbf{P}^{H_b}$$

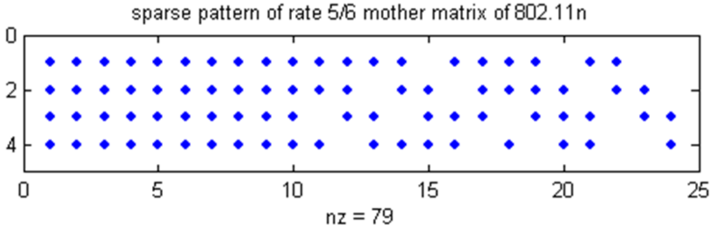
Different base matrices to support different rates.

Different expansion factors (z) to support multiple lengths.

All the shift coefficients for different codes for a given rate are obtained from the same base matrix using modulo arithmetic

Irregular QC-LDPC H Matrices



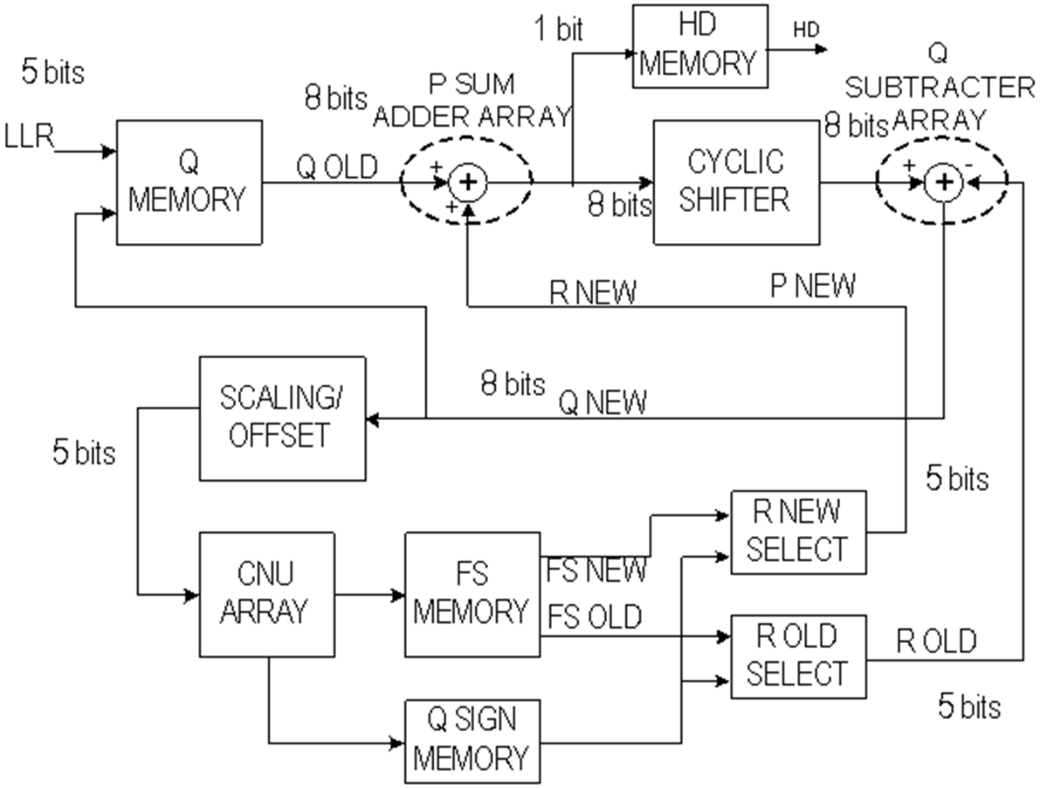


Irregular QC-LDPC H Matrices

- ❑ Existing implementations [*Hocevar*] for irregular QC-LDPC codes are still very complex to implement.
- ❑ These codes have the better BER performance and selected for IEEE 802.16e and IEEE 802.11n.
- ❑ It is anticipated that these type of codes will be the default choice for most of the standards.
- ❑ We show that with out-of-order processing and scheduling of layered processing, it is possible to design very efficient architectures.
- ❑ The same type of codes can be used in storage applications (holographic, flash and magnetic recording) if variable node degrees of 2 and 3 are avoided in the code construction for low error floor

Hocevar, D.E., "A reduced complexity decoder architecture via layered decoding of LDPC codes," IEEE Workshop on Signal Processing Systems, 2004. SIPS 2004. .pp. 107- 112, 13-15 Oct. 2004

Layered Decoder Architecture



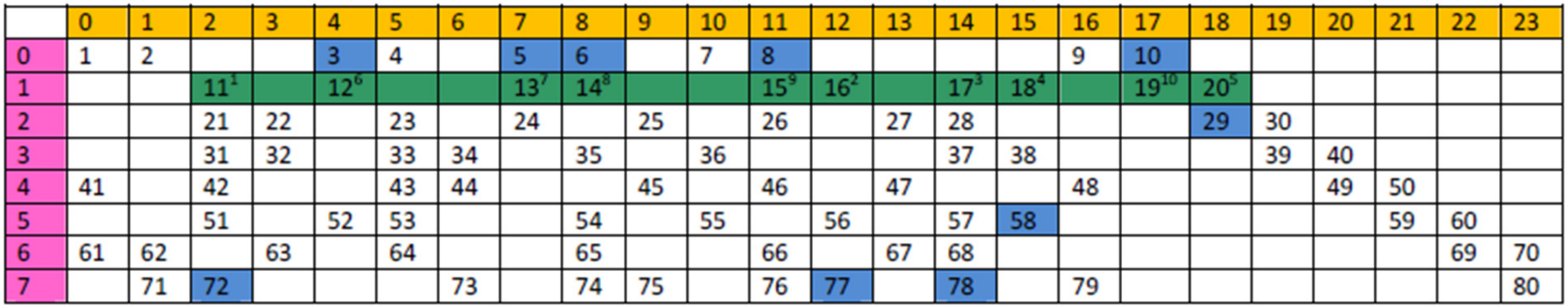
Advantages

1) Q memory (some times we call this as LPQ memory) can be used to store L/Q/P instead of 3 separate memories- memory is managed at circulant level as at any time for a given circulant we need only L or Q or P.
2) Only one shifter instead of 2 shifters
3) Value-reuse is effectively used for both Rnew and Rold
4) Low complexity data path design-with no redundant data

Path operations.

5) Low complexity CNU design.
6) Out-of-order processing at both layer and circulant level for all the processing steps such as Rnew and PS processing to eliminate the pipeline and memory access stall cycles.

Out-of-order layer processing for R Selection



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0	1	2			3	4		5	6		7	8					9	10						
1			11^{1}		12^{6}			13^{7}	14^{8}			15^{9}	16^{2}		17^{3}	18^{4}		19^{10}	20^{5}					
2			21	22		23		24		25		26		27	28				29	30				
3			31	32		33	34		35		36				37	38				39	40			
4	41		42			43	44			45		46		47			48				49	50		
5			51		52	53			54		55		56		57	58						59	60	
6	61	62		63		64			65			66		67	68								69	70
7		71	72				73		74	75		76	77		78		79							80

Normal practice is to compute R new messages for each layer after CNU PS processing.

However, here we decoupled the execution of R new messages of each layer with the execution of corresponding layer's CNU PS processing. Rather than simply generating Rnew messages per layer, we compute them on basis of circulant dependencies.

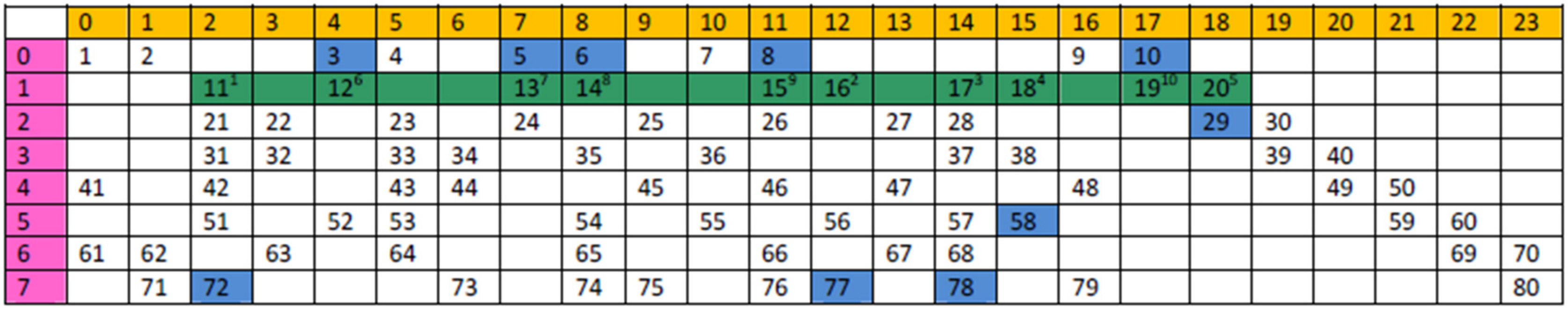
R selection is out-of-order so that it can feed the data required for the PS processing of the second layer. For instance Rnew messages for circulant 29 which belong to layer 3 are not generated immediately after layer 3 CNU PS processing .

Rather, Rnew for circulant 29 is computed when PS processing of circulant 20 is done as circulant 29 is a dependent circulant of circulant of 20.

Similarly, Rnew for circulant 72 is computed when PS processing of circulant 11 is done as circulant 72 is a dependent circulant of circulant of 11.

Here we execute the instruction/computation at precise moment when the result is needed!

Out-of-order block processing for Partial State



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0	1	2			3	4		5	6		7	8					9	10						
1			11^{1}		12^{6}			13^{7}	14^{8}			15^{9}	16^{2}		17^{3}	18^{4}		19^{10}	20^{5}					
2			21	22		23		24		25		26		27	28				29	30				
3			31	32		33	34		35		36				37	38				39	40			
4	41		42			43	44			45		46		47			48				49	50		
5			51		52	53			54		55		56		57	58						59	60	
6	61	62		63		64			65			66		67	68								69	70
7		71	72				73		74	75		76	77		78		79							80

Re-ordering of block processing . While processing the layer 2,

the blocks which depend on layer 1 will be processed last to allow for the pipeline latency.

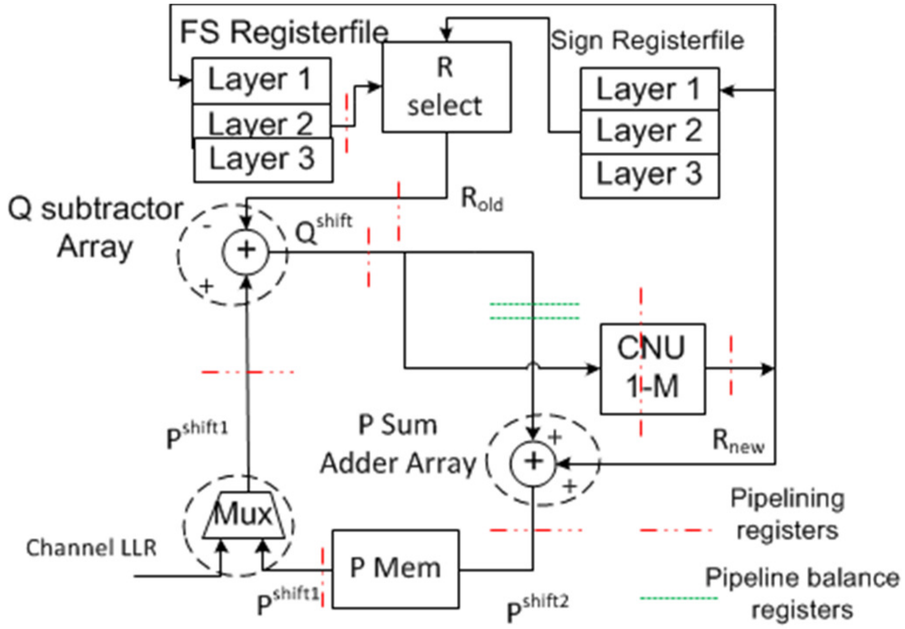
In the above example, the pipeline latency can be 5.

The vector pipeline depth is 5.so no stall cycles are needed while processing the layer 2 due to the pipelining. [In other implementations, the stall cycles are introduced – which will effectively reduce the throughput by a huge margin.]

Also we will sequence the operations in layer such that we process the block first that has dependent data available for the longest time.

This naturally leads us to true out-of-order processing across several layers. In practice we wont do out-of-order partial state processing involving more than 2 layers.

Block Parallel Layered Decoder



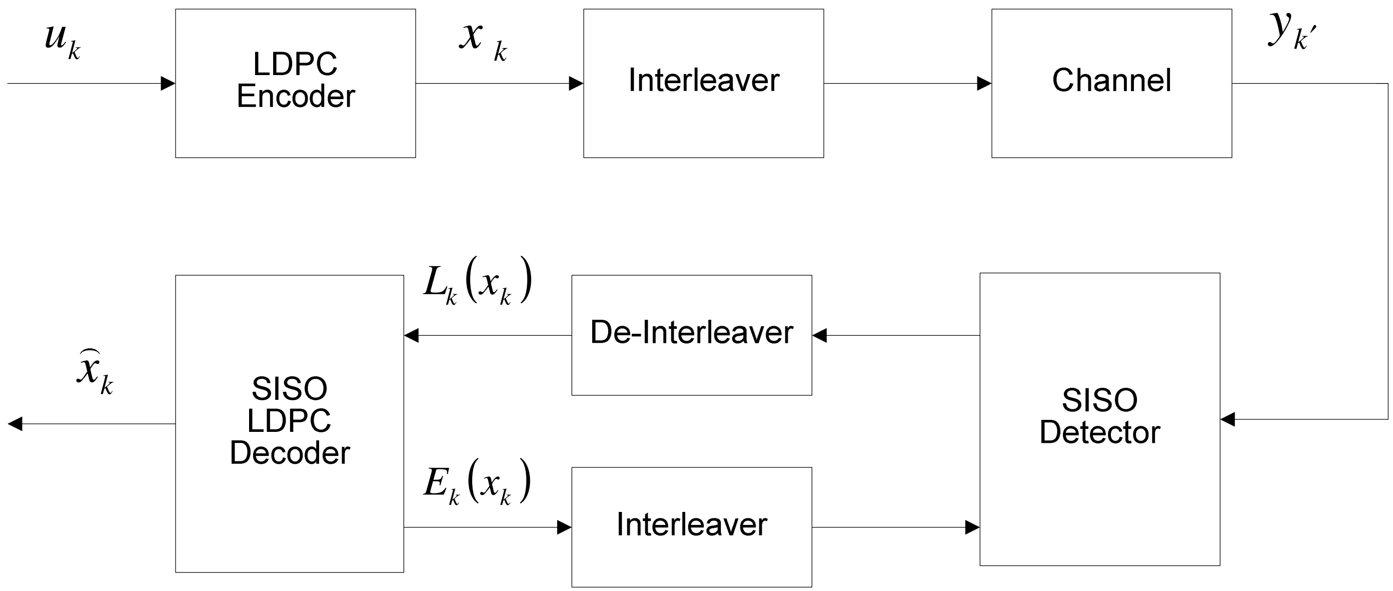
Compared to other work, this work has several advantages

1) Only one memory for holding the P values.
2) Shifting is achieved through memory reads. Only one memory multiplexer network is needed instead of 2 to achieve delta shifts
3) Value-reuse is effectively used for both Rnew and Rold
4) Low complexity data path design-with no redundant data Path operations.
5) Low complexity CNU design with high parallelism.
6) Smaller pipeline depth
7) Out-of-order row processing to hide the pipeline latencies.

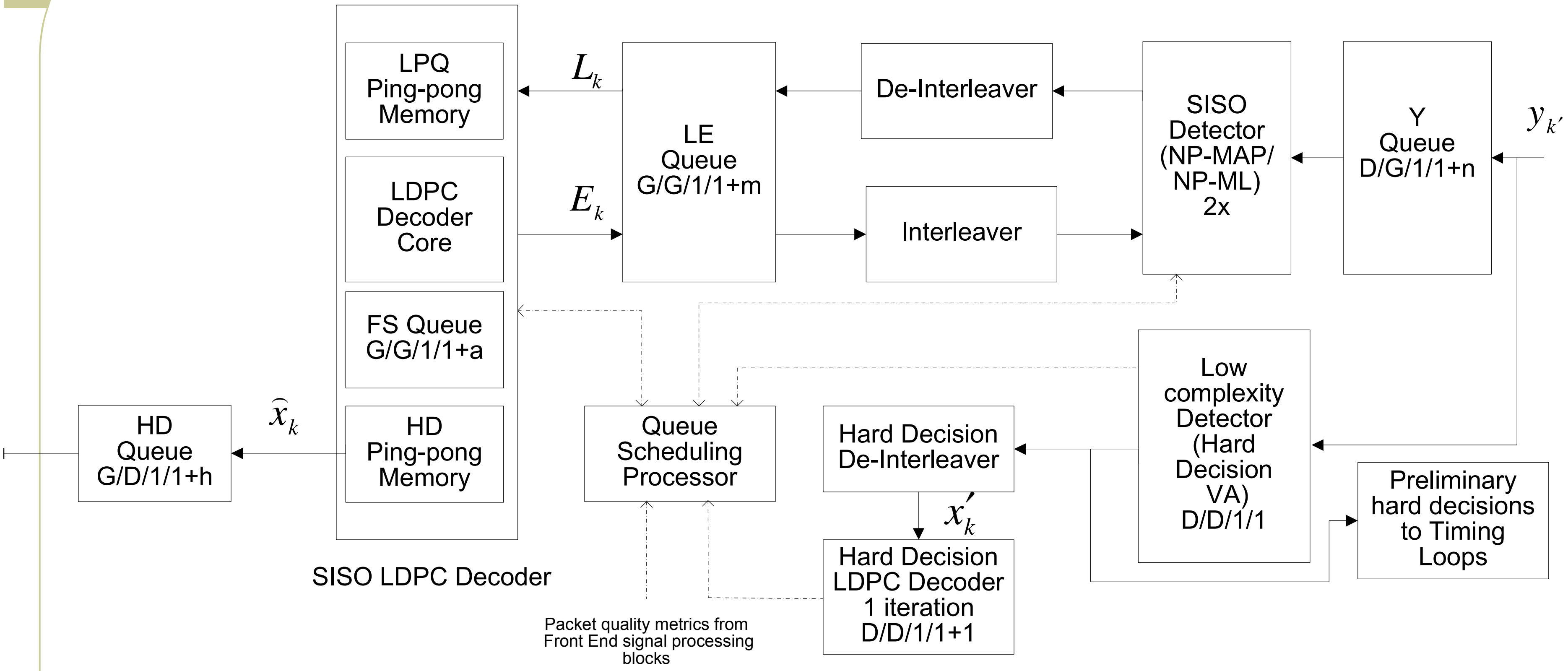
Here M is the row parallelization (i.e. number of rows in H matrix Processed per clock).

TURBO EQUALIZATION

System Model for Turbo Equalization



Proposed System Level Architecture for Turbo Equalization


LPQ Ping-pong Memory
LDPC Decoder Core
FS Queue G/G/1/1+a
HD Ping-pong Memory
SISO LDPC Decoder
L_k
E_k
LE Queue G/G/1/1+m
De-Interleaver
Interleaver
SISO Detector (NP-MAP/ NP-ML) 2x
Y Queue D/G/1/1+n
$y_{k'}$
$\hat{x}_k$
HD Queue G/D/1/1+h
Queue Scheduling Processor
Hard Decision De-Interleaver
x'_k
Hard Decision LDPC Decoder 1 iteration D/D/1/1+1
Low complexity Detector (Hard Decision VA) D/D/1/1
Preliminary hard decisions to Timing Loops
Packet quality metrics from Front End signal processing blocks

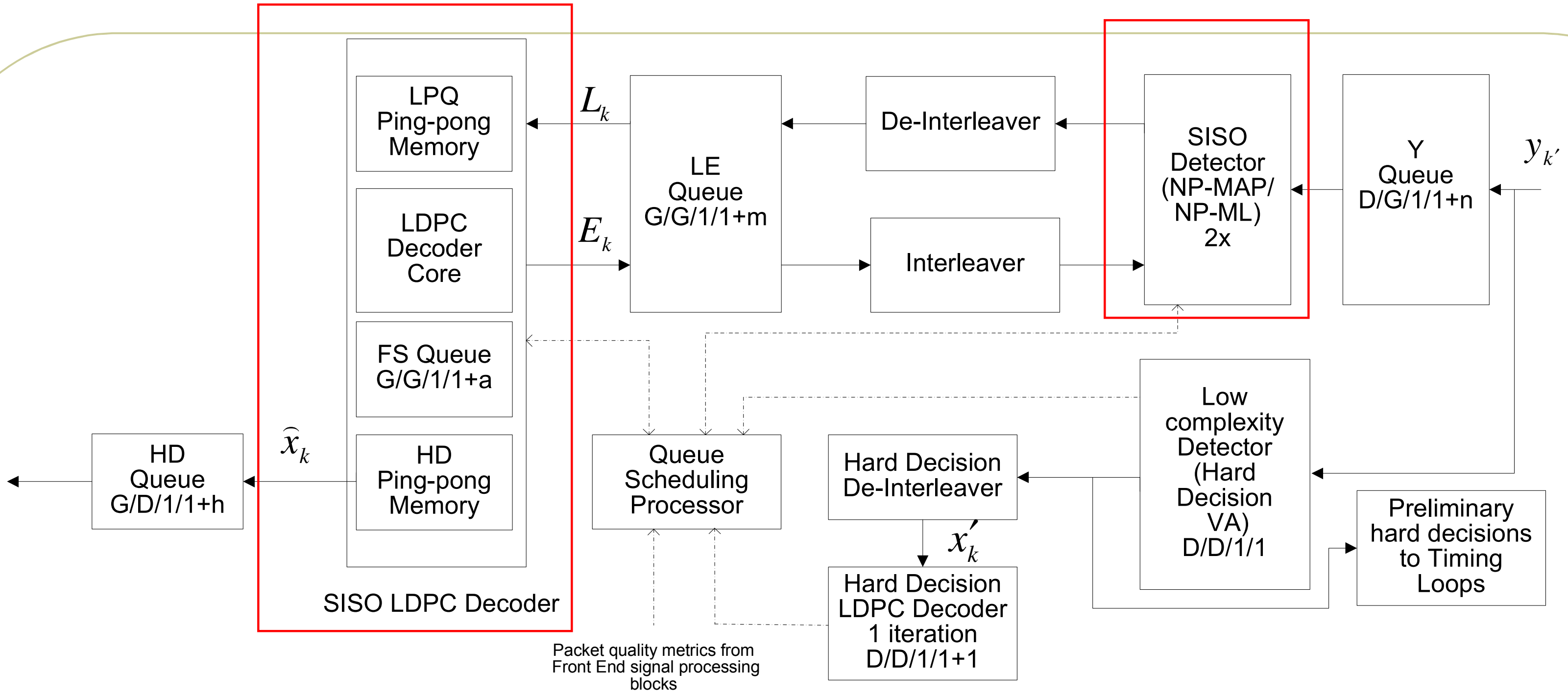
Gunnam et. al, “Next generation iterative LDPC solutions for magnetic recording storage,” Signals, Systems and Computers, **Invited Paper**, 42nd Asilomar Conference on, Publication Year: 2008 , Page(s): 1148 – 1152

Why Statistical Buffering?

- The innovation here is the novel and efficient arrangement of queue structures such that we would get the performance of a hardware system that is configured to run h (which is set to 20 in the example configuration) maximum global iterations while the system complexity is proportional to the hardware system that can 2 maximum global iterations.

- D/G/1/1+n is Kendall's notation of a queuing model. The first part represents the input process, the second the service distribution, and the third the number of servers.
  D- Deterministic
  G- General

Primary Data path



LPQ Ping-pong Memory
LDPC Decoder Core
FS Queue G/G/1/1+a
HD Ping-pong Memory
SISO LDPC Decoder
L_k
E_k
LE Queue G/G/1/1+m
De-Interleaver
Interleaver
SISO Detector (NP-MAP/ NP-ML) 2x
Y Queue D/G/1/1+n
$y_{k'}$
$\hat{x}_k$
HD Queue G/D/1/1+h
Queue Scheduling Processor
Hard Decision De-Interleaver
x'_k
Hard Decision LDPC Decoder 1 iteration D/D/1/1+1
Low complexity Detector (Hard Decision VA) D/D/1/1
Preliminary hard decisions to Timing Loops
Packet quality metrics from Front End signal processing blocks

- The primary data-path contains one SISO LDPC decoder and one SISO detector.
- The LDPC decoder is designed such that it can handle the total amount of average iterations in two global iterations for each packet.
- The SISO detector is in fact two detector modules that operate on the same packet but different halves of the packet thus ensuring one packet can be processed in 50% of the inter-arrival time T. Each detector processes 4 samples per clock cycle.
- Thus both the detector and the LDPC decoder can sustain maximum of two global iterations per each packet if no statistical buffering is employed.

Local-Global Interleaver

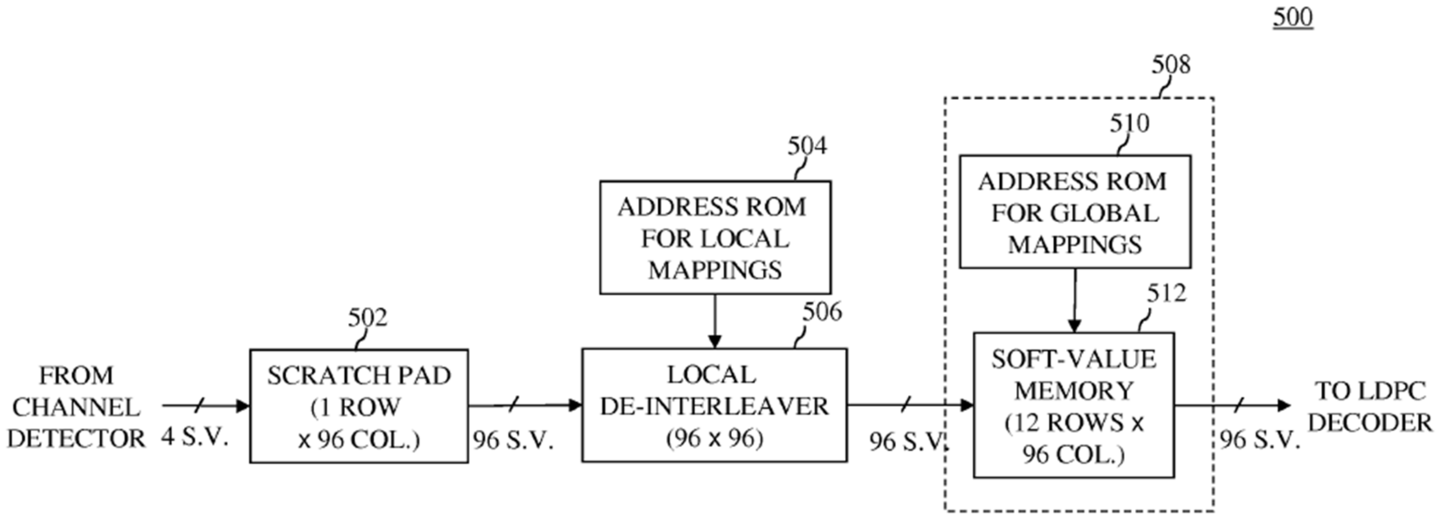

500
508
510
504
ADDRESS ROM FOR GLOBAL MAPPINGS
ADDRESS ROM FOR LOCAL MAPPINGS
512
502
506
FROM CHANNEL DETECTOR
4 S.V.
SCRATCH PAD (1 ROW x 96 COL.)
96 S.V.
LOCAL DE-INTERLEAVER (96 x 96)
96 S.V.
SOFT-VALUE MEMORY (12 ROWS x 96 COL.)
96 S.V.
TO LDPC DECODER

FIG. 5

Row-Column interleavers need to have memory organized such that it can supply the data samples for both row and column access.
Low latency memory efficient interleaver compared to traditional row-column interleaver. Only one type of access (i.e row access) is needed for both detector and decoder.

Data flow in Local-Global Interleaver

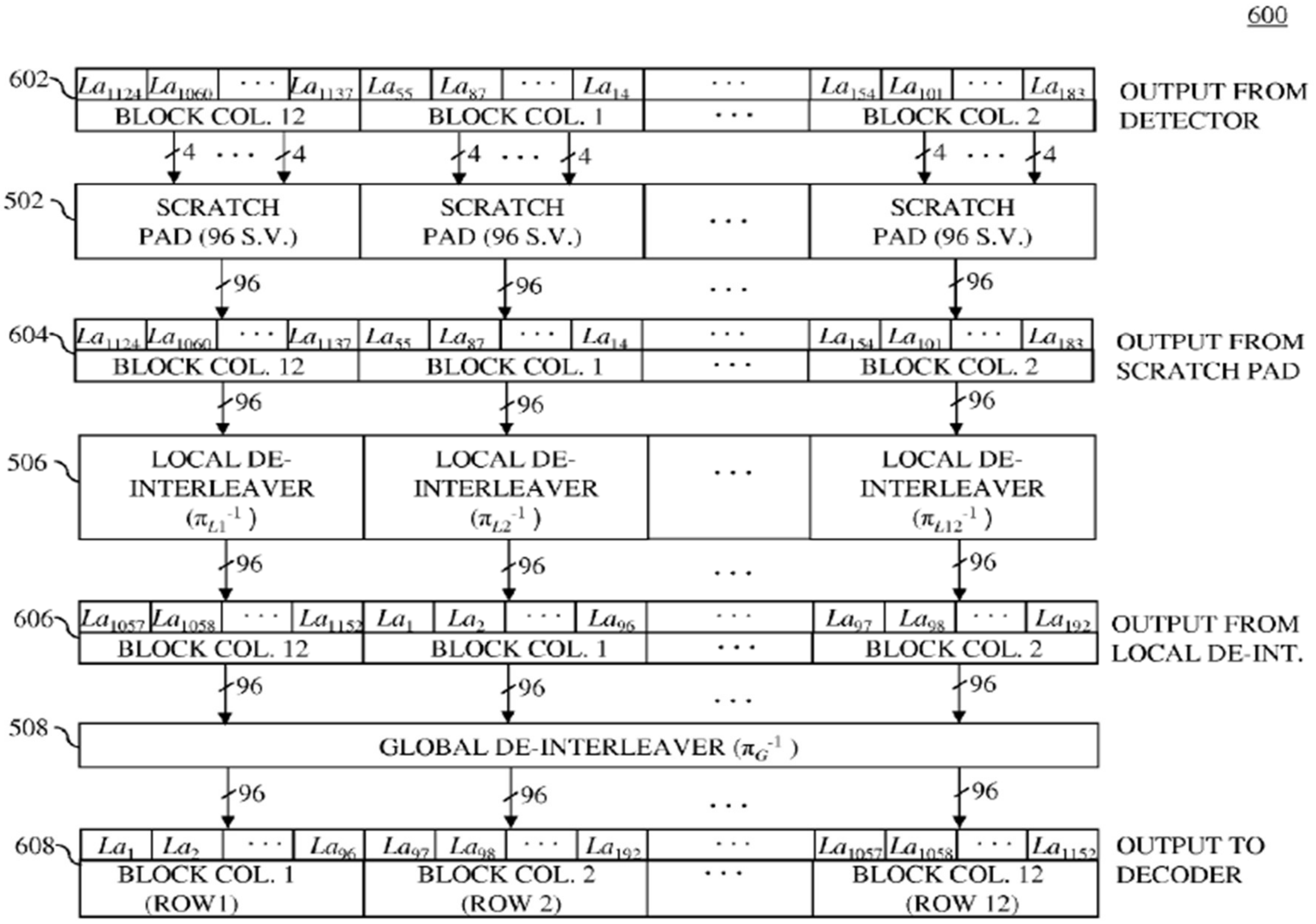

600
602
La_{1124} La_{1060} · · · La_{1137} La_{55} La_{87} · · · La_{14} · · · La_{154} La_{101} · · · La_{183}
BLOCK COL. 12
BLOCK COL. 1
BLOCK COL. 2
OUTPUT FROM DETECTOR
4 · · · 4
502
SCRATCH PAD (96 S.V.)
96
604
La_{1124} La_{1060} · · · La_{1137} La_{55} La_{87} · · · La_{14} · · · La_{154} La_{101} · · · La_{183}
BLOCK COL. 12
BLOCK COL. 1
BLOCK COL. 2
OUTPUT FROM SCRATCH PAD
506
LOCAL DE-INTERLEAVER (π_{L1}^{-1})
LOCAL DE-INTERLEAVER (π_{L2}^{-1})
LOCAL DE-INTERLEAVER (π_{L12}^{-1})
606
La_{1057} La_{1058} · · · La_{1152} La_1 La_2 · · · La_{96} · · · La_{97} La_{98} · · · La_{192}
BLOCK COL. 12
BLOCK COL. 1
BLOCK COL. 2
OUTPUT FROM LOCAL DE-INT.
508
GLOBAL DE-INTERLEAVER (π_G^{-1})
608
La_1 La_2 · · · La_{96} La_{97} La_{98} · · · La_{192} · · · La_{1057} La_{1058} · · · La_{1152}
BLOCK COL. 1 (ROW1)
BLOCK COL. 2 (ROW 2)
BLOCK COL. 12 (ROW 12)
OUTPUT TO DECODER

FIG. 6

Secondary Data path

- The secondary data path contains the low complexity detector based on hard decision Viterbi algorithm, a hard decision interleaver followed by hard decision LDPC decoder that is sized for doing only one iteration.

- The secondary path thus does one reduced complexity global iteration and operates on the incoming packets immediately.

1) it can generate preliminary decisions immediately (with a latency equal to T) to drive the front end timing loops thus making the front end processing immune from the variable time processing in the primary data path

2) it can generate quality metrics to the queue scheduling processor.

- The low complexity detector is in the arrangement D/D/1/1 according to Kendall Notation[8]: the arrival times are deterministic, the processing time/service times are deterministic, one processor and one memory associated with the processor.

- The low complexity decoder is in the arrangement D/D/1/1+1 – this is similar to the low complexity detector except that there is one additional input buffer to enable the simultaneous filling of the input buffer while the hard decision iteration is being performed on a previous packet. Note that LDPC decoder needs the complete codeword before it can start the processing

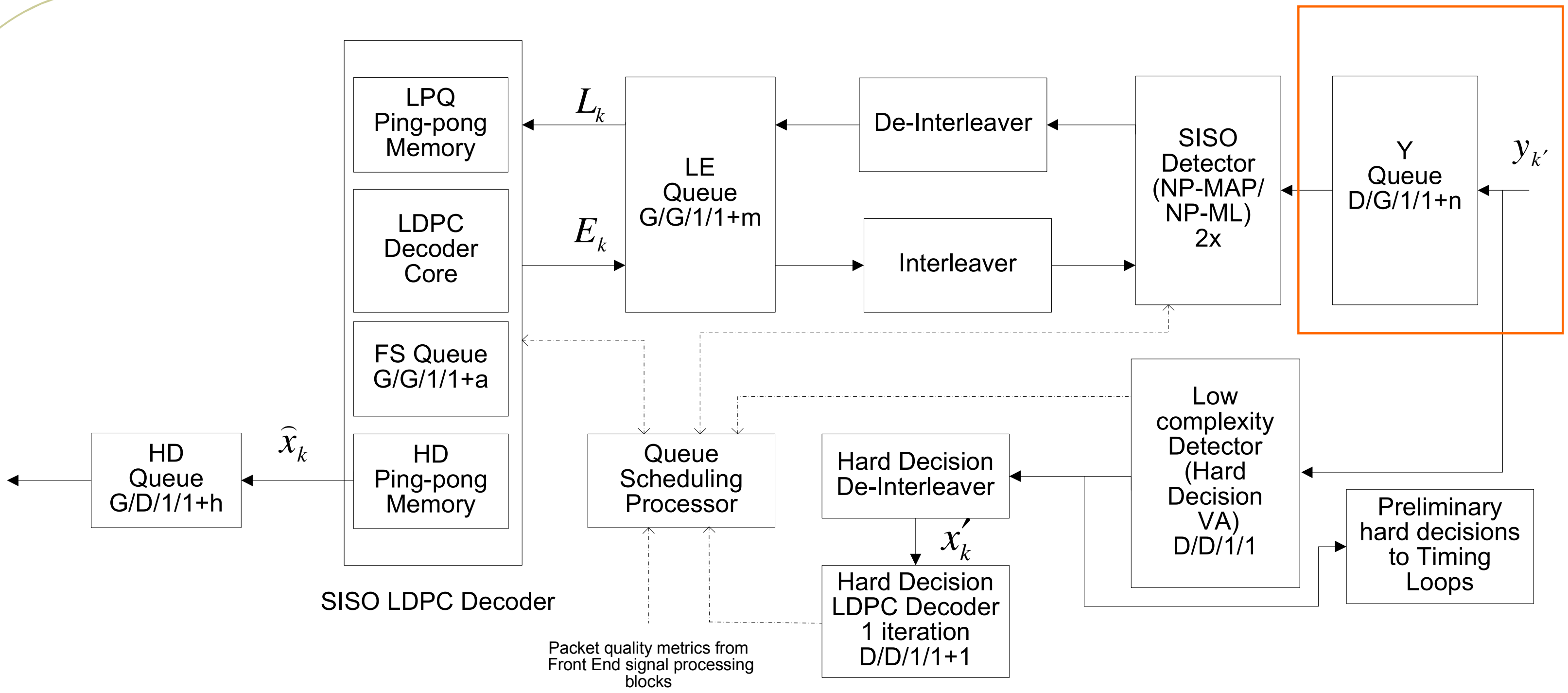
Variations in number of global and local iterations

- In the last successful global iteration the LDPC decoder does the variable number of local iterations.

- The left-over LDPC decoder processing time is shared to increase the number of local iterations in following global iterations for the next packet.

- For each packet, at least one global iteration is performed and the distribution of required global iterations follows a general distribution that heavily depends on the signal to noise ratio.

Y Queue

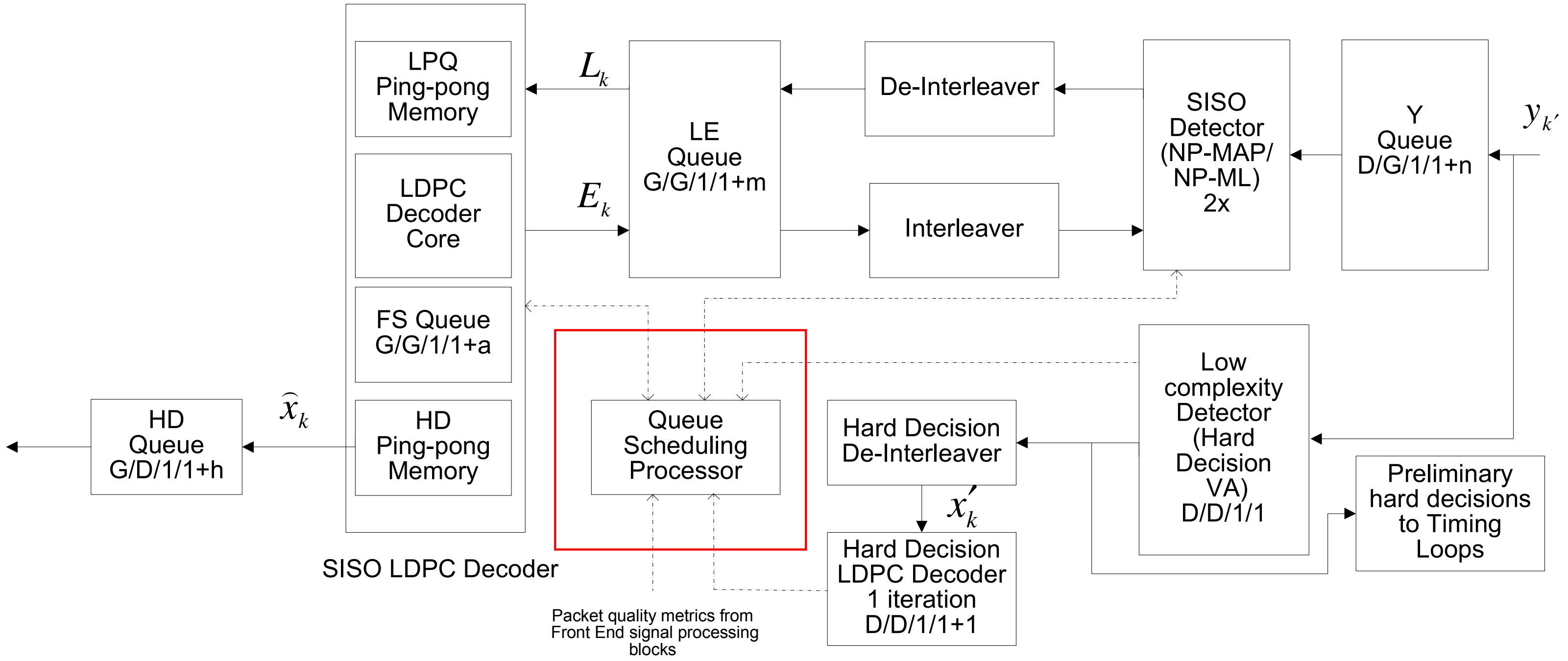


LPQ Ping-pong Memory
LDPC Decoder Core
FS Queue G/G/1/1+a
HD Ping-pong Memory
SISO LDPC Decoder
L_k
E_k
LE Queue G/G/1/1+m
De-Interleaver
Interleaver
SISO Detector (NP-MAP/ NP-ML) 2x
Y Queue D/G/1/1+n
$y_{k'}$
HD Queue G/D/1/1+h
$\hat{x}_k$
Queue Scheduling Processor
Hard Decision De-Interleaver
x'_k
Hard Decision LDPC Decoder 1 iteration D/D/1/1+1
Low complexity Detector (Hard Decision VA) D/D/1/1
Preliminary hard decisions to Timing Loops
Packet quality metrics from Front End signal processing blocks

- The y-sample data for each arriving packet is buffered in Y queue. Since the data comes at deterministic time intervals in a real-time application, the arrival process is D and the inter-arrival time is T.

- The overall processing/service time for each packet is variable and is a general distribution G. Assume that 4 y-samples per clock in each packet are arriving and the packets are coming continuously. In real-time applications, we need to be able to process 4-samples per clock though some latency is permitted.

Queue scheduling processor


LPQ Ping-pong Memory
LDPC Decoder Core
FS Queue G/G/1/1+a
HD Ping-pong Memory
SISO LDPC Decoder
L_k
E_k
LE Queue G/G/1/1+m
De-Interleaver
Interleaver
SISO Detector (NP-MAP/ NP-ML) 2x
Y Queue D/G/1/1+n
$y_{k'}$
HD Queue G/D/1/1+h
$\widehat{x}_k$
Queue Scheduling Processor
Hard Decision De-Interleaver
x_k'
Hard Decision LDPC Decoder 1 iteration D/D/1/1+1
Low complexity Detector (Hard Decision VA) D/D/1/1
Preliminary hard decisions to Timing Loops
Packet quality metrics from Front End signal processing blocks

- The queue scheduling processor takes the various quality metrics from the secondary reduced complexity data path as well as the intermediate processing from the primary data path.
- One example of a quality metric is the number of unsatisfied checks from LDPC decoder. All the queues in the primary data path are preemptive such that the packets are processed according to the quality metric obtained through preprocessing.

One example configuration

- In the example configuration of y-queues D/G/c/1+n, c=1 as we have one LDPC processor that can complete the processing of packet, the number of additional y-buffers, n.
- Assume that all the other queues are optimized and have the values m=4,a=3, h=20.
- Here rho = lambda*E(S) where lambda is the average arrival rate and E(S) is the average service time.
- The performance measures are calculated under the assumption that lambda is 1/T (i.e. 1 packet is coming every T time units) and constant and the average service time E(S) (is less than or equal to T time units) varying based on the SNR.
- Thus the value of rho is between 0 and 1. one minus rho represents 1-rho and is indicator of the system's average availability.
- The main requirement is that rejection probability should be kept low.

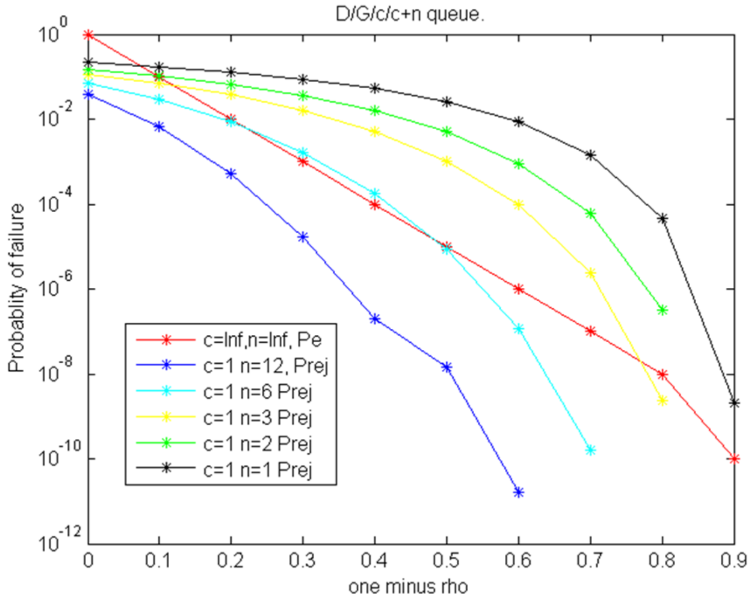
A) Should be less than 1e-6 at rho of 0.5

B) Should be less than 1e-9 at rho of 0.9

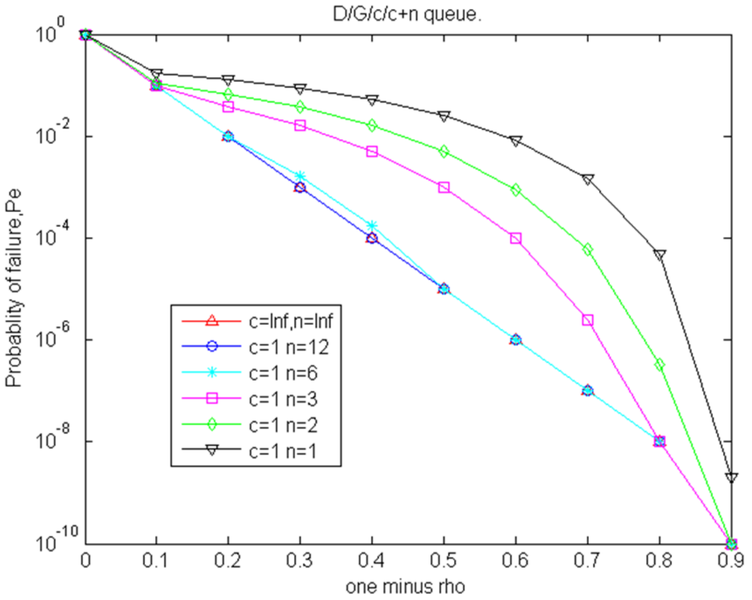
C) Asymptotically should reach 0 as rho increases beyond 0.9

The above requirements are based on the magnetic recording channel.

Different queue configurations



Probability of packet rejection for different queue configurations



Probability of overall packet failure (Pe) for different queue configurations

Queuing systems summary

- By comparing the previous two figures, we can see that we can either increase the processing power by 3 times (which is more expensive) or increase the number of y-buffers in the system n to 12 to achieve identical results. While it is not shown, we can get more benefits by doing both of them!

- Note that both the configurations in the previous two figures are still statistically buffered systems with m=4,a=3, h=20.

- If statistical buffering is disabled for other buffers in the system, then we need much higher number of processors up to 10 to gain the performance of a system that has no statistical buffering.

- As the average number of global iteration varies from 1 to 2 based on the SNR and the required number of global iterations vary from 1 to 20, the system with 10 processors with no statistical buffering would be idle for most of the time the proposed system with statistical buffering needs to have only one processor and can do the global iterations from 1 to 20.

- In conclusion, we show that statistical buffering if carefully done brings significant performance gains to the system while maintaining the low system complexity.

ERROR FLOOR MITIGATION

Error Floors of LDPC Codes

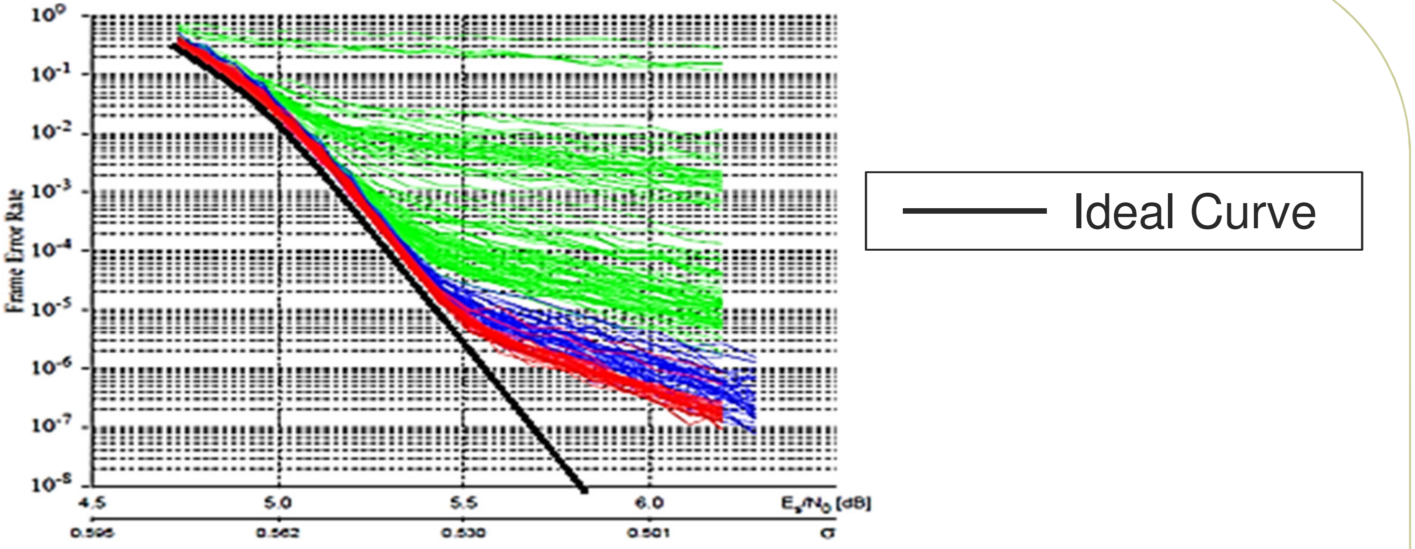


Figure 2: Frame error rates for random graphs (green), girth optimized (blue), and neighborhood optimized (red) graphs all of identical degree structure (nearly regular (3,15)).

When the BER/FER is plotted for conventional codes using classic decoding techniques, the BER steadily decreases in the form of a curve as the SNR condition becomes better. For LDPC codes and turbo codes that uses iterative decoding, there is a point after which the curve does not fall as quickly as before, in other words, there is a region in which performance flattens. This region is called the *error floor region*. The region just before the sudden drop in performance is called the *waterfall region*. Error floors are usually attributed to low-weight codewords (in the case of Turbo codes) and trapping sets or near-codewords (in the case of LDPC codes).

Richardson, “Error Floors of LDPC Codes”

Getting the curve steeper again: Error Floor Mitigation Schemes

- The effect of trapping sets is influenced by noise characteristics, H matrix structure, order of layers decoded, fixed point effects, quality of LLRs from detector. Several schemes are developed considering above factors. Some of them are

1) Changing the LLRs given to the decoder by using the knowledge of USCs, detector metrics and front end signal processing markers

2) With the knowledge of USCs, match the error pattern to a known trapping set. If the trapping set information is completely known, simply flip the bits and do the CRC.

If the trapping set information is partially known (i.e. only few bit error locations are stored due to storage issues), then do target bit adjustment using this information.

If no information on trapping set is stored, then identify the bits connected to USC based on H matrix information. Simply try TBE on each bit group.

Targetted bit adjustment on a bit/a bit group refers to the process of flipping the sign of these bits to opposite value and setting the magnitude of bit LLRs to maximum while keeping the other bit sign values unaltered but limiting their magnitude to around 5% of maximum LLR.

Couple of ways to reduce the number of experiments.

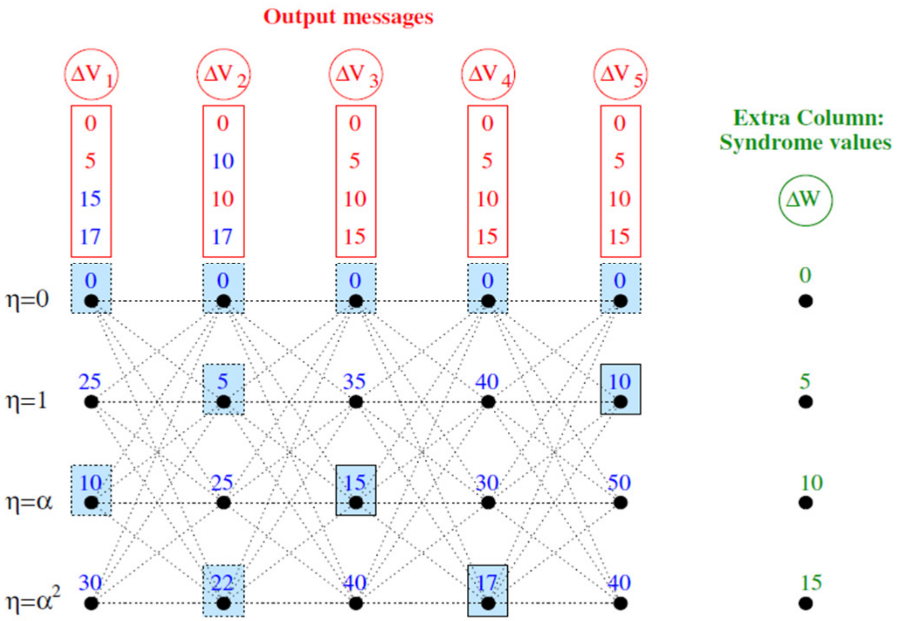
3) When multi-way interleaving is used, use of the separate inteleavers on each component codeword.

4) Skip-layer decoding: Conditionally decode a high row weight layer only when trapping set signature is present (USC <32).

T-EMS, CHECK NODE UPDATE FOR NON-BINARY LDPC

T-EMS

- Make use of the full trellis representation of messages in the Delta-domain messages.
- Select only a small subset of trellis nodes to build the most reliable configurations
  ⇒ minimization of the complexity.
- Add an extra-column to the trellis composed of Syndrome reliabilities for the parallel update of the output messages
  ⇒ improved decoding latency.



References

[1] Gunnam, K.K.; Choi, G.S.; Yeary, M.B.; Shaohua Yang; Yuanxing Lee, “Next generation iterative LDPC solutions for magnetic recording storage,” Signals, Systems and Computers, 2008 42nd Asilomar Conference on, Publication Year: 2008 , Page(s): 1148 – 1152

[2] Kiran Gunnam, "LDPC Decoding: VLSI Architectures and Implementations", Invited presentation at Flash Memory Summit, Santa Clara, August 2012. http://www.flashmemorysummit.com/English/Collaterals/Proceedings/2012/20120822_LDPC%20Tutorial_Module2.pdf

[3] Ned Varnica, "LDPC Decoding: VLSI Architectures and Implementations", Invited presentation at Flash Memory Summit, Santa Clara, August 2012. http://www.flashmemorysummit.com/English/Collaterals/Proceedings/2012/20120822_LDPC%20Tutorial_Module1.pdf

[4] K. K. Gunnam, G. S. Choi, and M. B. Yeary, “Value-reuse properties of min-sum for GF(q),” Texas A&M University Technical Note, Oct.2006, published around August 2010.

Extended papers on [3]:

[5] E. Li, K. Gunnam and D. Declercq, "Trellis based Extended Min-Sum for Decoding Nonbinary LDPC codes",in the proc. of ISWCS'11, Aachen, Germany, November 2011

http://perso-etis.ensea.fr/~declercq/PDF/ConferencePapers/Li_2011_ISWCS.pdf.gz

[6] E. Li, D. Declercq and K. Gunnam, "Trellis based Extended Min-Sum Algorithm for Non-binary LDPC codes and its Hardware Structure", IEEE Trans. Communications., 2013

More references

6. Gunnam, KK; Choi, G. S.; Yeary, M. B.; Atiquzzaman, M.; “VLSI Architectures for Layered Decoding for Irregular LDPC Codes of WiMax,” Communications, 2007. ICC '07. IEEE International Conference on 24-28 June 2007 Page(s):4542 - 4547

7. Gunnam, K.; Gwan Choi; Weihuang Wang; Yeary, M.; “Multi-Rate Layered Decoder Architecture for Block LDPC Codes of the IEEE 802.11n Wireless Standard,” Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on 27-30 May 2007 Page(s):1645 – 1648

8. Gunnam, K.; Weihuang Wang; Gwan Choi; Yeary, M.; “VLSI Architectures for Turbo Decoding Message Passing Using Min-Sum for Rate-Compatible Array LDPC Codes,” Wireless Pervasive Computing, 2007. ISWPC '07. 2nd International Symposium on 5-7 Feb. 2007

9. Gunnam, Kiran K.; Choi, Gwan S.; Wang, Weihuang; Kim, Euncheol; Yeary, Mark B.; “Decoding of Quasi-cyclic LDPC Codes Using an On-the-Fly Computation,” Signals, Systems and Computers, 2006. ACSSC '06. Fortieth Asilomar Conference on Oct.-Nov. 2006 Page(s):1192 - 1199

10 Gunnam, K.K.; Choi, G.S.; Yeary, M.B.; “A Parallel VLSI Architecture for Layered Decoding for Array LDPC Codes,” VLSI Design, 2007. Held jointly with 6th International Conference on Embedded Systems., 20th International Conference on Jan. 2007 Page(s):738 – 743

11. Gunnam, K.; Gwan Choi; Yeary, M.; “An LDPC decoding schedule for memory access reduction,” Acoustics, Speech, and Signal Processing, 2004. Proceedings. (ICASSP '04). IEEE International Conference on Volume 5, 17-21 May 2004 Page(s):V - 173-6 vol.5

12 GUNNAM, Kiran K., CHOI, Gwan S., and YEARY, Mark B., "Technical Note on Iterative LDPC Solutions for Turbo Equalization," Texas A&M Technical Note, Department of ECE, Texas A&M University, College Station, TX 77843, Report dated July 2006. Available online at http://dropzone.tamu.edu March 2010, Page(s): 1-5.

13. K. Gunnam, G. Choi, W. Wang, and M. B. Yeary, “Parallel VLSI Architecture for Layered Decoding ,” Texas A&M Technical Report, May 2007. Available online at http://dropzone.tamu.edu

Check http://dropzone.tamu.edu for technical reports.

Several features in this presentation and in references[1-13] are covered by the following 3 patents and other pending patent applications by Texas A&M University System (TAMUS).

[P1] K. K. Gunnam and G. S. Choi, “Low Density Parity Check Decoder for Regular LDPC Codes,” U.S. Patent 8,359,522

[P2] K. K. Gunnam and G. S. Choi, “Low Density Parity Check Decoder for Irregular LDPC Codes,” U.S. Patent 8,418,023

[P3] K. K. Gunnam and G. S. Choi , “Low Density Parity Check Decoder for Irregular LDPC Codes,” US Patent 8,555,140

[P4] K. K. Gunnam and G. S. Choi, “Low Density Parity Check Decoder for Regular LDPC Codes “, US Patent Application 20130097469

[P5] K. K. Gunnam and G. S. Choi, “Low Density Parity Check Decoder “, US Patent Application 14/141,508

BACKUP SLIDES