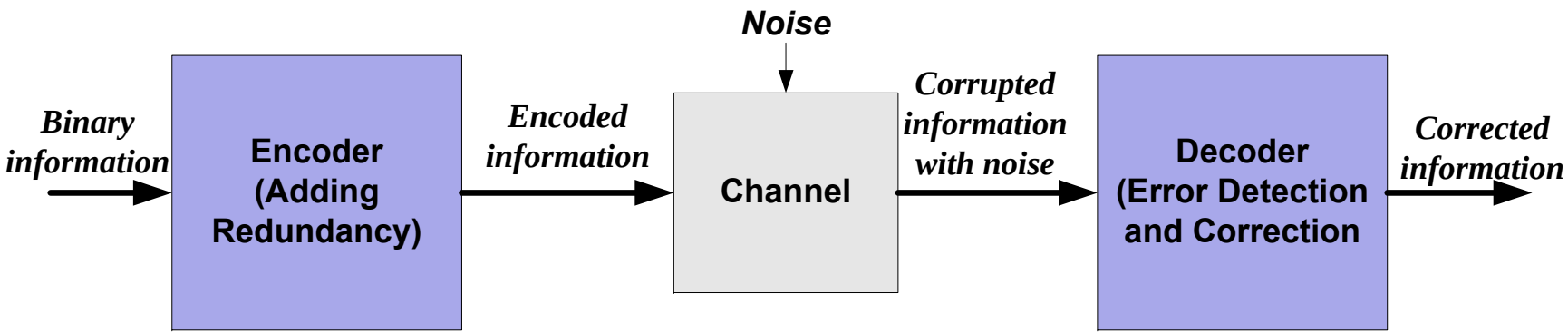
A 32 Gbps 2048-bit 10GBASE-T Ethernet Energy Efficient LDPC Decoder with Split-Row Threshold Decoding Method

Tinoosh Mohsenin and Bevan M. Baas
VLSI Computation Lab, ECE Department
University of California, Davis

Outline

- **Introduction to LDPC Codes and Iterative Decoding**
- Goals and Key Ideas
- Split-Row Threshold Decoding Method
- Error Performance Results
- Multi-Split-Row Threshold Decoder Implementations and Results
- Conclusion

Error Correction in Communication Systems



- Error correction is widely used in communication systems
- Low-density parity-check (LDPC) code has been demonstrated to have a very good error correction performance

LDPC Code Applications

- Standards
  - Digital Video Broadcasting (DVB-S2): 2005
  - 10 Gigabit Ethernet (10GBASE-T): 2006
  - WiMAX (802.16e)
  - WiFi (802.11n)
  - WPANs (802.15.3c)
- Applications
  - Flash memory
  - Hard disks
  - Deep-space satellite communications

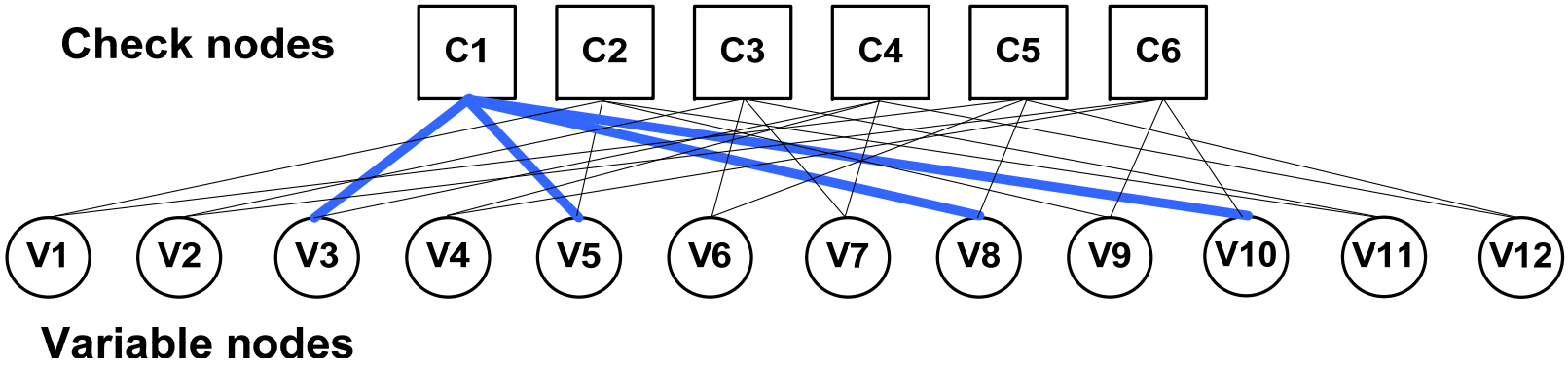
LDPC Codes

- Defined by a large binary matrix, called parity check matrix or *H* matrix

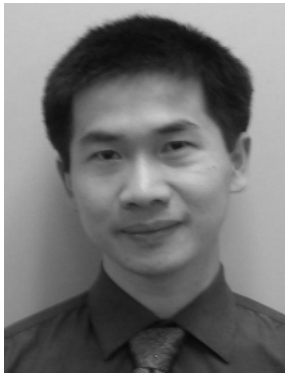
- Example (12,6) LDPC code
  - Code length (*N*)=12
  - Information length (*K*)=6
  - Row weight (W_r)=4
  - Column weight (W_c)=2
  - Row size (No. of parity checks)=6

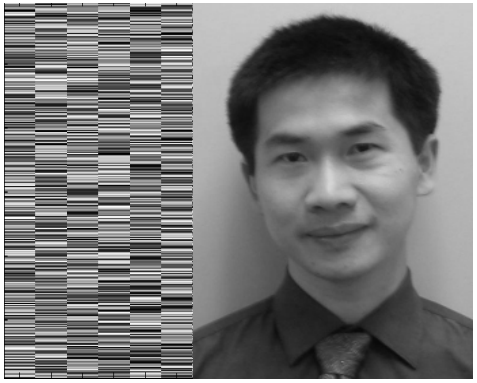
$H =$

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	
0	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	C1
1	0	0	0	1	0	0	0	1	0	1	0	C2
0	1	0	0	0	1	1	0	0	0	1	0	C3
0	0	1	1	0	0	1	0	0	0	0	1	C4
1	0	0	0	0	1	0	1	0	0	0	1	C5
0	1	0	1	0	0	0	0	1	1	0	0	C6



Encoding Picture Example





$$H=\begin{bmatrix} 1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,\ldots\,1\,1\,0\,0\,0\,1\,1\,1\,1\,1\,0\,0\,0\,0\,0\,1\,0 \\ 0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,\ldots\,1\,1\,1\,0\,0\,0\,1\,1\,1\,1\,1\,0\,0\,0\,0\,0\,1 \\ 0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,\ldots\,0\,0\,1\,1\,0\,1\,1\,0\,0\,0\,1\,1\,0\,0\,0\,1\,0 \\ 0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,\ldots\,0\,0\,0\,1\,1\,0\,1\,1\,0\,0\,0\,1\,1\,0\,0\,0\,1 \\ \ldots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \ldots \end{bmatrix}$$

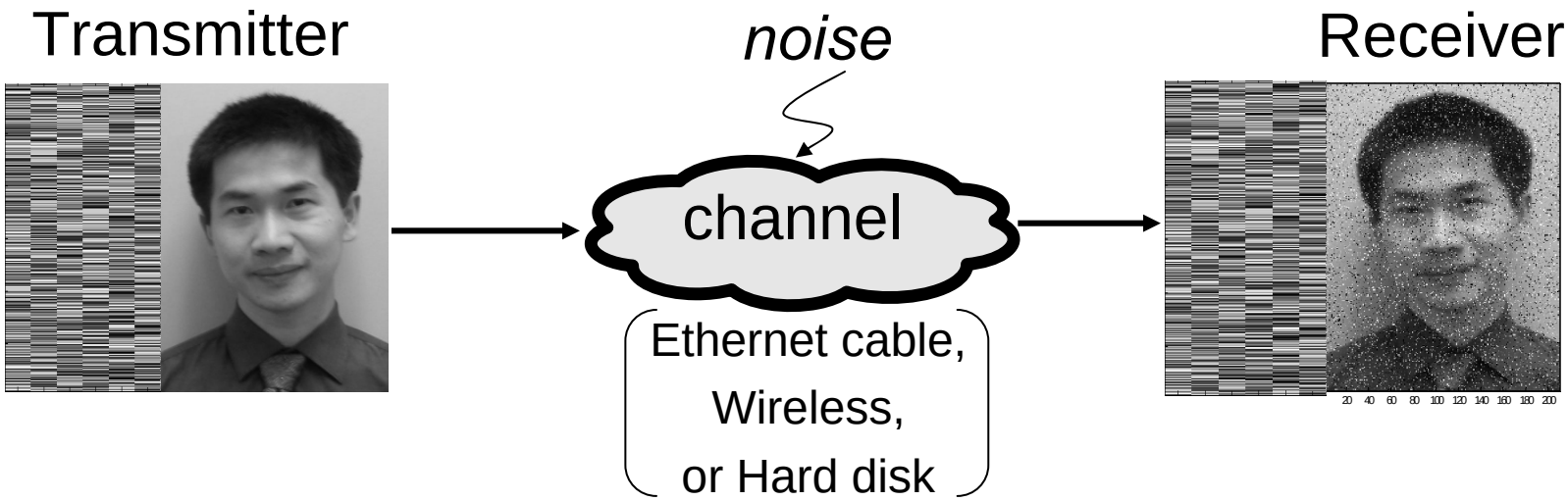
$$V=\begin{bmatrix} 1\,0\,1\,1\,1\,0\,1\,1\,0\,0\,0\,0\,\ldots\,1\,1\,1\,1\,0\,0\,0\,1\,1\,1\,1\,1 \\ 0\,1\,0\,0\,1\,0\,0\,0\,1\,0\,1\,1\,\ldots\,0\,1\,1\,0\,0\,1\,0\,0\,0\,1\,1\,0 \\ 0\,1\,0\,1\,0\,0\,0\,1\,1\,1\,1\,1\,\ldots\,1\,0\,0\,1\,0\,0\,1\,1\,1\,0\,0\,1 \\ 0\,1\,0\,0\,0\,1\,1\,0\,0\,1\,0\,0\,\ldots\,0\,1\,0\,0\,1\,1\,1\,0\,0\,1\,0\,0 \\ \ldots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \ldots \end{bmatrix}$$

V_1

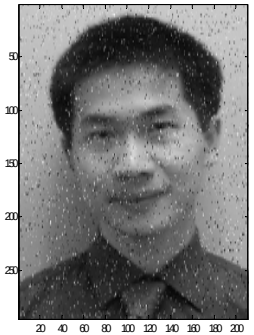
← Parity → ← Image →

$$H\times V_i^T=0$$

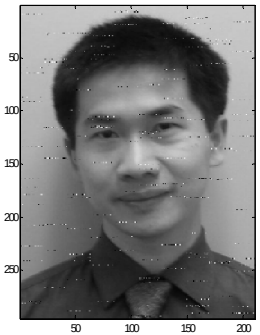
Decoding Picture Example



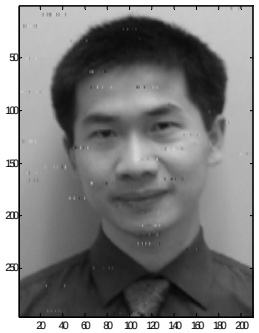
Iterative decoding



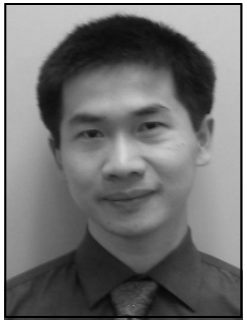
Iteration 1



Iteration 5



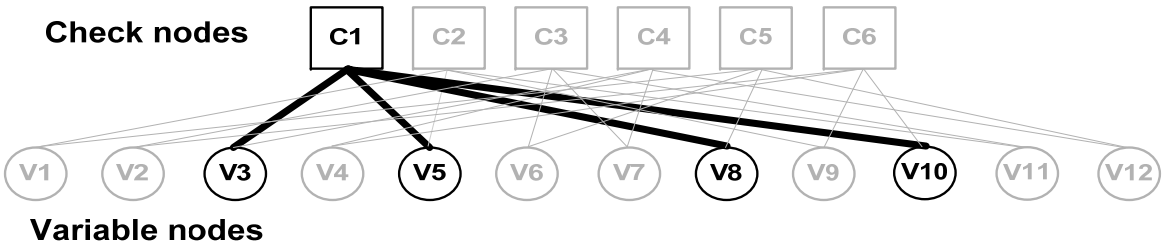
Iteration 15

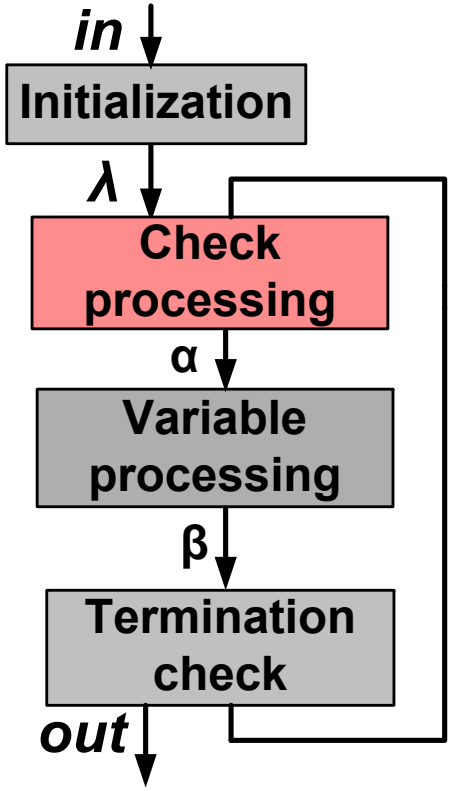


Iteration 16

Message Passing (Check node processing)

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





SPA $\alpha_{ij} = \left(\prod_{j', h_{ij'}=1, } sign\beta_{ij'} \right) \times \varphi \left(\sum_{j,' h_{ij}'=1, j' \neq j} \varphi \left(\left| \beta_{ij'} \right| \right) \right)$ $\varphi = \log[\tanh(\frac{x}{2})]$

MinSum: $\alpha_{ij} = \left(\prod_{j', h_{ij'}=1, } sign\beta_{ij'} \right) \times \min_{j', h_{ij'}=1, j' \neq j} \left(\left| \beta_{ij'} \right| \right)$

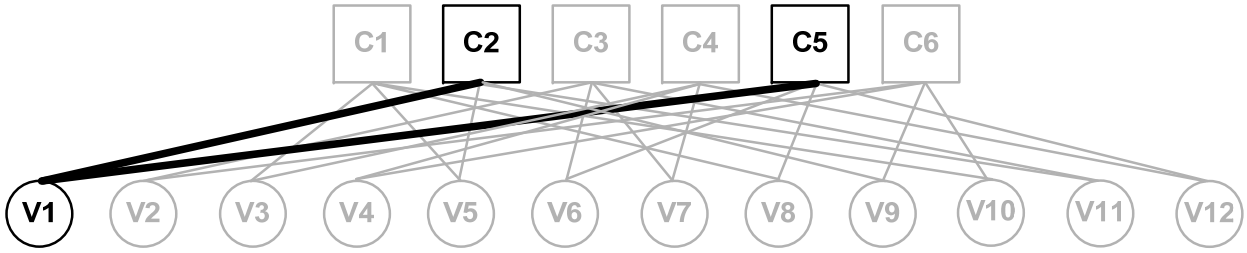
$$|\alpha_{ij}| = \begin{cases} Min2_i, & \text{if } j == Min1\_index \\ Min1_i, & \text{if } j \neq Min1\_index \end{cases}$$

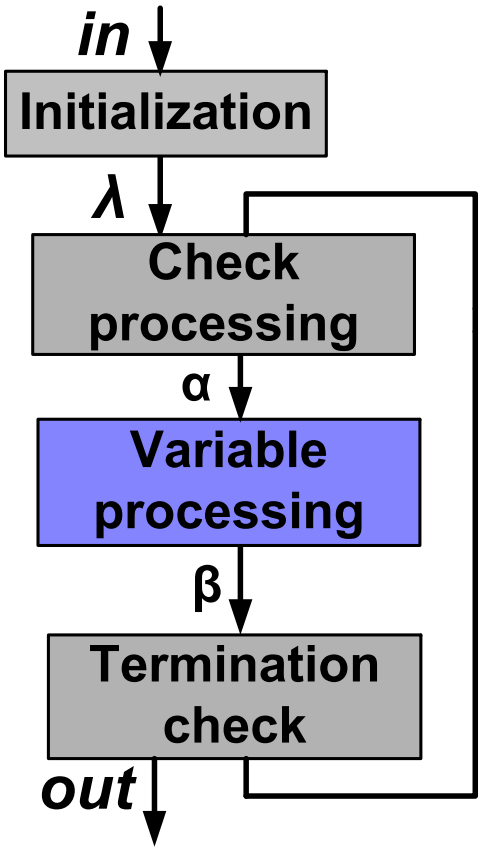
$$Min1_i = \min_{j \in V(i)} (|\beta_{ij}|)$$

$$Min2_i = 2nd \min_{j \in V(i)} (|\beta_{ij}|)$$

Message Passing (Variable node processing)

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



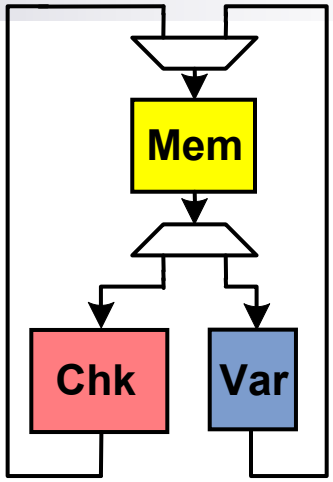


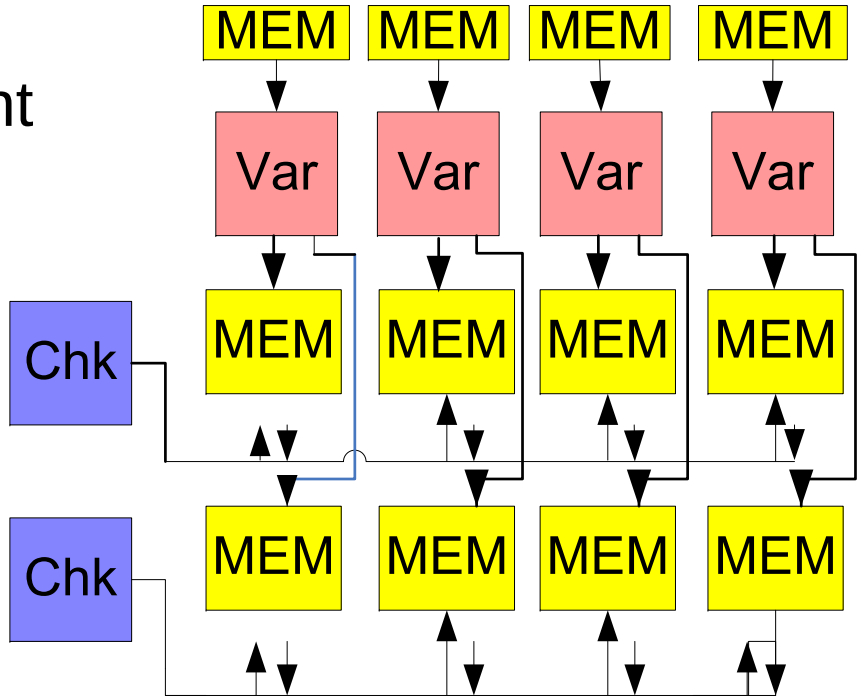
$$\beta_{ij} = \sum_{j', h_{ij'}=1} \alpha_{ij'} + \lambda_j$$

λ is the received information from the channel

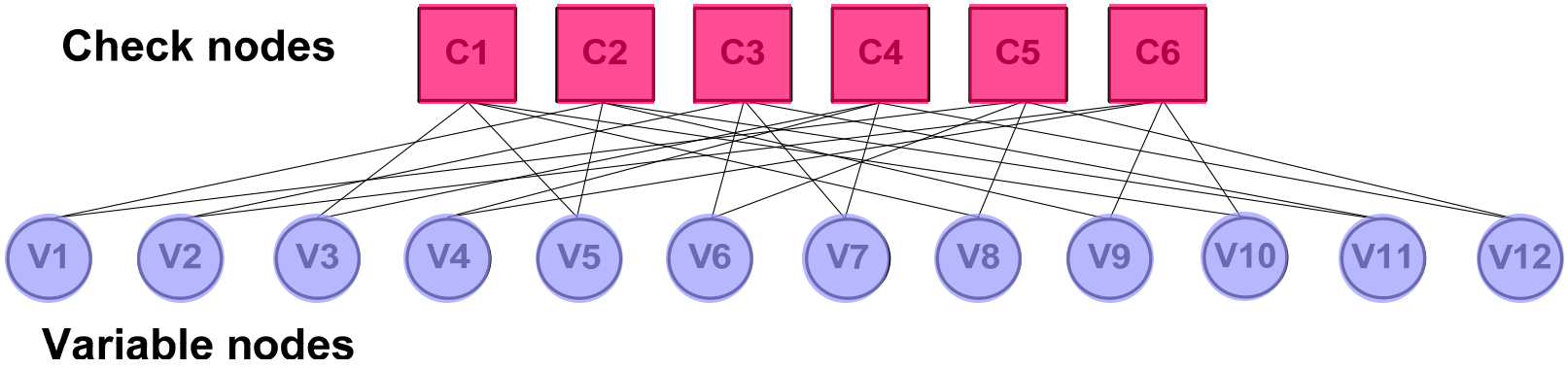
Decoding Architectures

- Serial and partial parallel decoders
  - One or multiple row and column processors, share a few memory banks
  - Throughput in the range of a few 100 Mbps
  - Large memory requirement

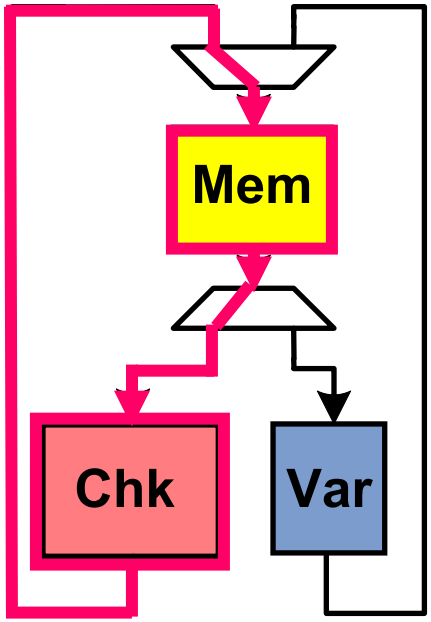




Serial Decoding



(1) *initialize memory (clear contents)*

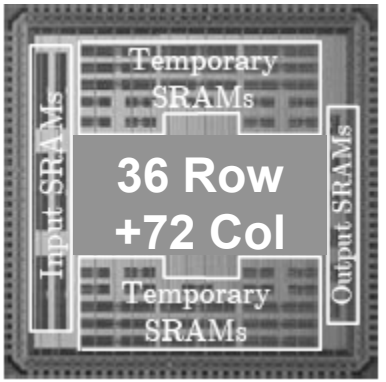


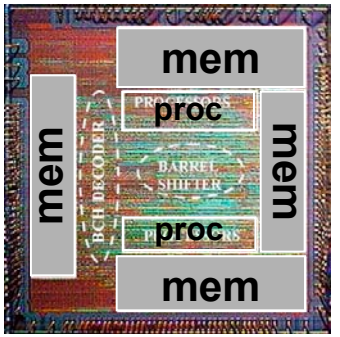
(2) *compute V1 V2 V3 and store V4 V5 V6 V7 V8 V9 V10 V11 V12*

(3) *…now compute C1 C2 C3 and store C4 C5 C6*

Partial Parallel Decoder Examples

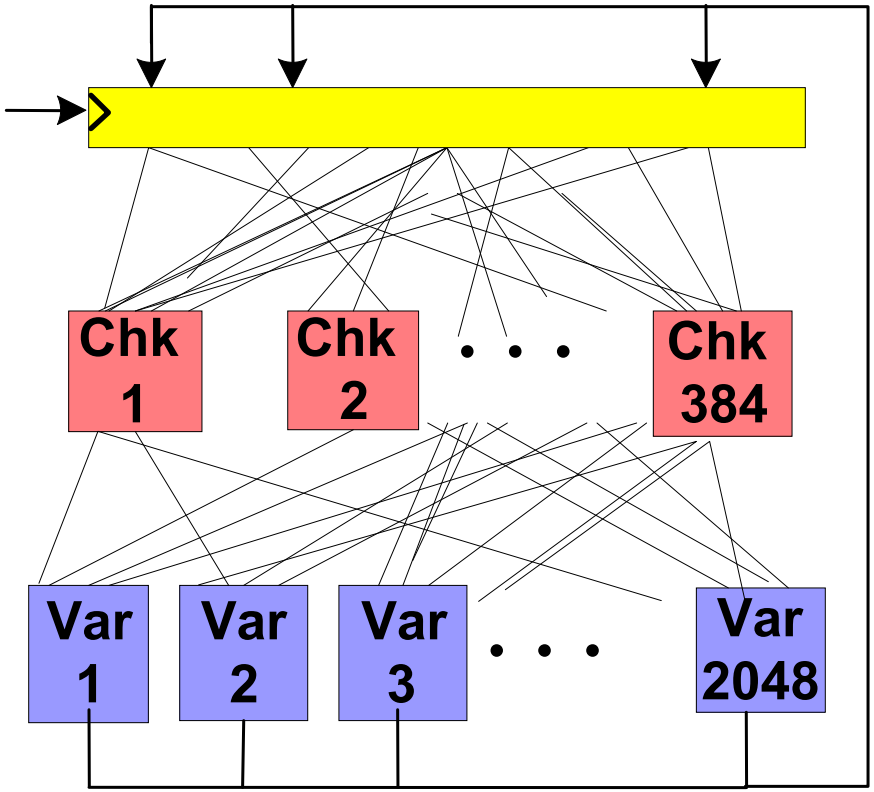
- Example 1: 2304b, rate-1/2, (3,6) decoder [T. Ishikawa et al., *ASP DAC*, 2006]
  - 36 row, 72 column processors, 85 Kb mem
  - 36 mm^2, 180 nm CMOS
  - 530 Mbps
  - 3.6 W @1.8 V
- Example 2: 64800b DVB-S2 Compliant [P.Urad et al., *ISSCC*, Feb 2008]
  - 180 processors, 3.18Mb mem
  - 6.07mm^2, 65 nm CMOS
  - 105 Mbps
  - 360 mW @1.2 V



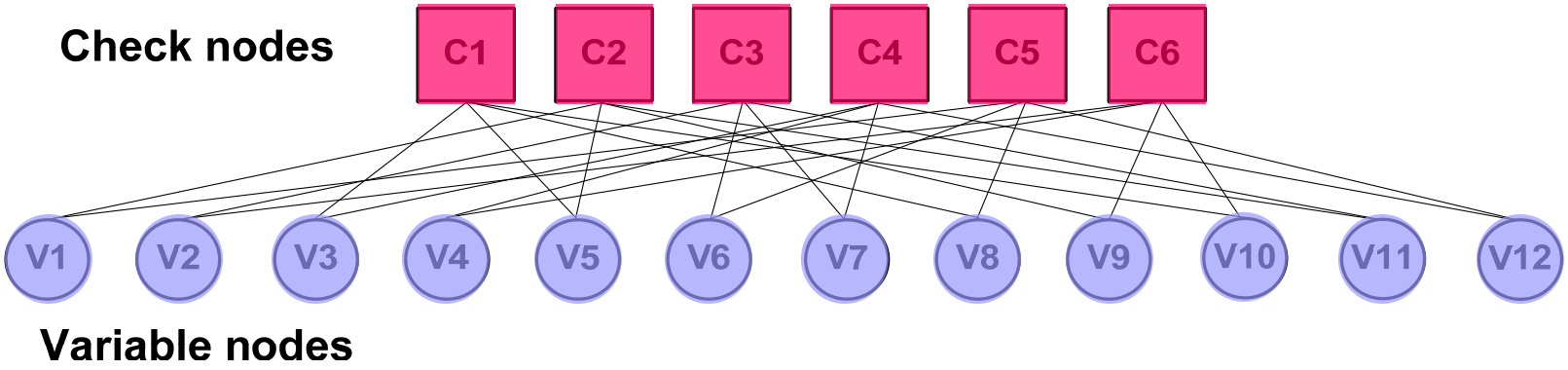


Decoding Architectures- Continued

- Full-parallel decoders
  - Row and column processors connected according to the parity check matrix
  - Highest throughput, no memory
- Major challenges
  - Routing congestion
  - Large delay, area, and power caused by **long global wires**



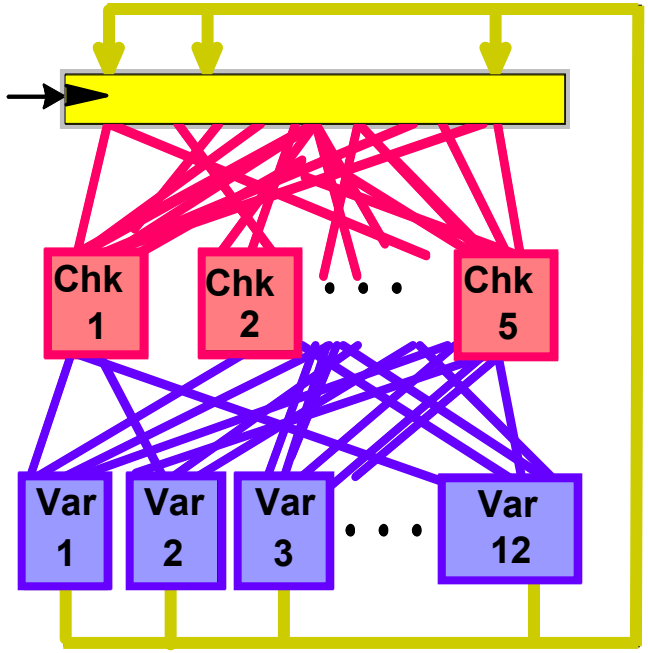
Full-Parallel Decoding



(1) *initialize registers (clear contents)*

(2) *compute C1,2,3,4,5,6*

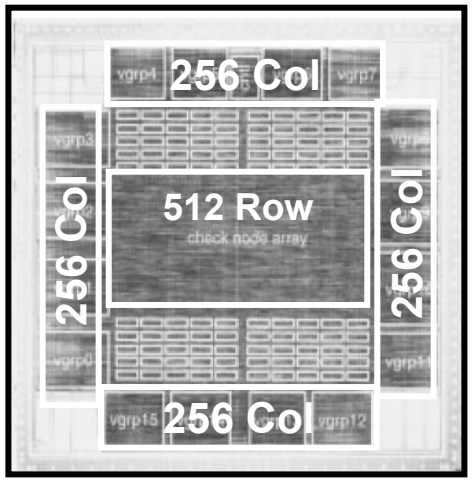
(3) *…now compute V1,2,3,4,5,6,7,8,9, 10,11,12*



(4) *Store into registers*

Full-parallel Decoder Examples

- Example 1: 1024-bit, irregular code, 4 bits per symbol, [A. Blanksby et al., *JSSC*, Mar 2002]
  - 52.5 mm^2, 160 nm CMOS
  - 64 MHz, 1Gbit/sec
  - 690 mW @ 1.5 V
- Example 2: 660-bit [A. Darabiha et al., *CICC*, Sep 2007]
  - 9 mm^2,130 nm CMOS
  - 300 MHz, 3.3 Gbps
  - 1408 mW @1.3 V



Outline

- Introduction to LDPC Codes and Iterative Decoding
- **Goals and Key Ideas**
- Split-Row Threshold Decoding Method
- Error Performance Results
- Multi-Split-Row Threshold Decoder Implementations and Results
- Conclusion

LDPC Decoder Design Goals and Features

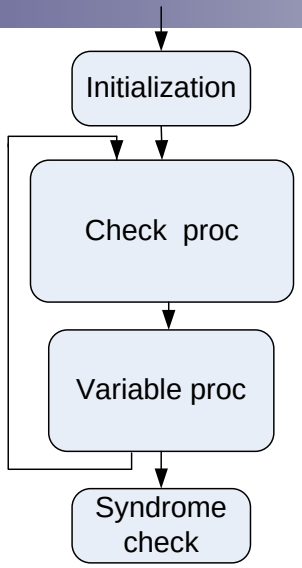
- Key goals
  - Very high throughput and high energy efficiency
  - Area efficient (small circuit area)
  - Well suited for long-length and large row weight LDPC codes
  - Easy implementation with automatic CAD tools
  - Good error performance

- Split-Row decoding key features
  - Reduced **interconnect complexity**
  - Reduced **processor complexity**

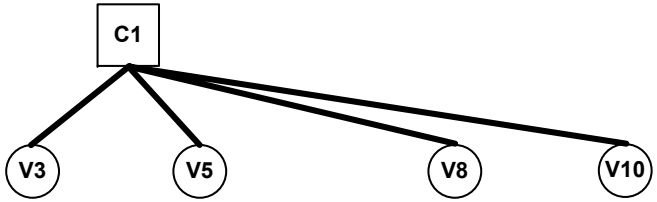
T. Mohsenin and B. Baas, “Split-row: A reduced complexity, high throughput LDPC decoder architecture,” in *ICCD*, 2006
T. Mohsenin and B. Baas, “High-throughput LDPC decoders using a multiple Split-Row method,” in *ICASSP*, 2007

Standard MinSum vs. Split-Row Decoding

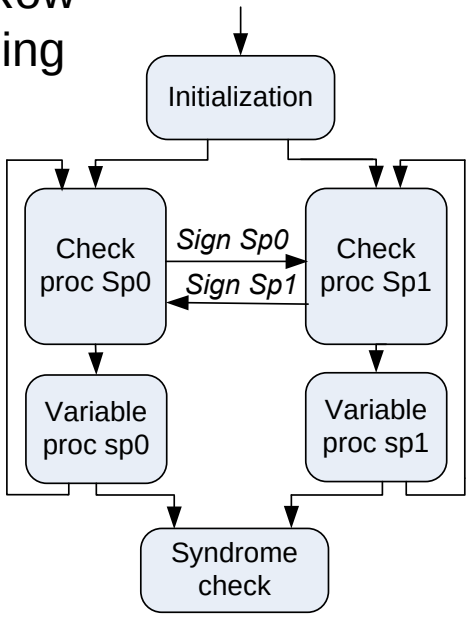
Standard MinSum decoding



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



Split-Row decoding

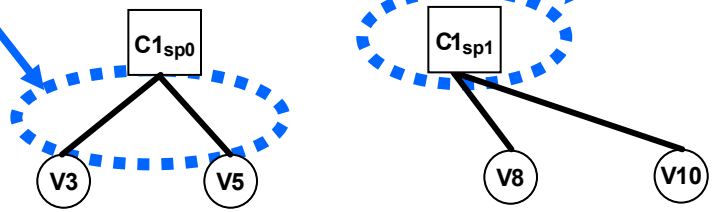


$$H = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1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

$H_{split\text{-}sp0}$ $H_{split\text{-}sp1}$

reduction of input wires to check processor

reduction of check processor area



MinSum vs. MinSum Split-Row

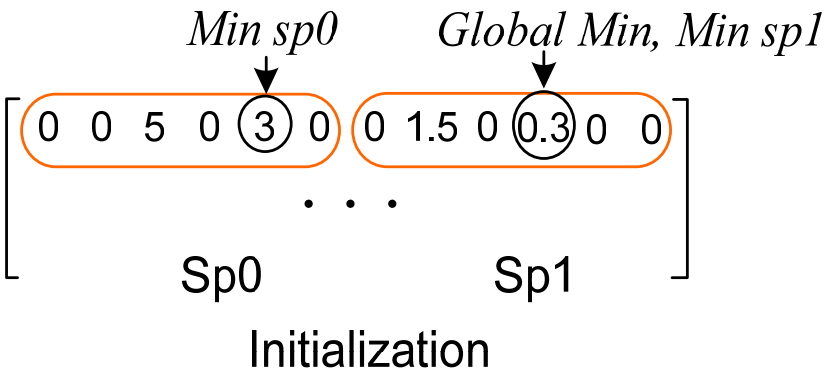
MinSum:

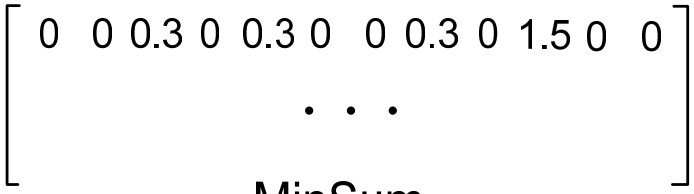
Sign | Magnitude

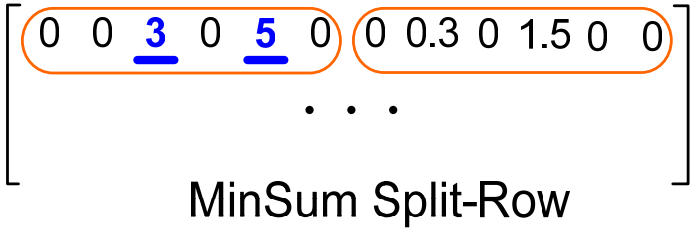
$$\alpha_{ij\,MS} = S_{MS} \times \left(\prod_{j', h_{ij'}=1, j' \neq j} sign\,\beta_{ij'} \right) \times \min_{j', h_{ij'}=1, j' \neq j} \left(|\beta_{ij'}| \right)$$

MinSum Split-Row:

$$\alpha_{ij\,MS\,Split-Row} = S_{MS\,Split-Row} \times \left(\prod_{j', h_{ij'}=1, j' \neq j} sign\,\beta_{ij'} \right) \times \min_{j', h_{ij'\,Split-Row}=1, j' \neq j} \left(|\beta_{ij'}| \right)$$





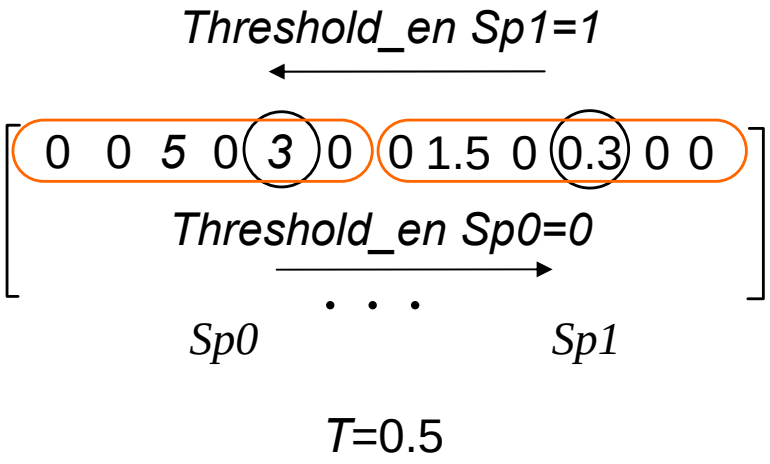


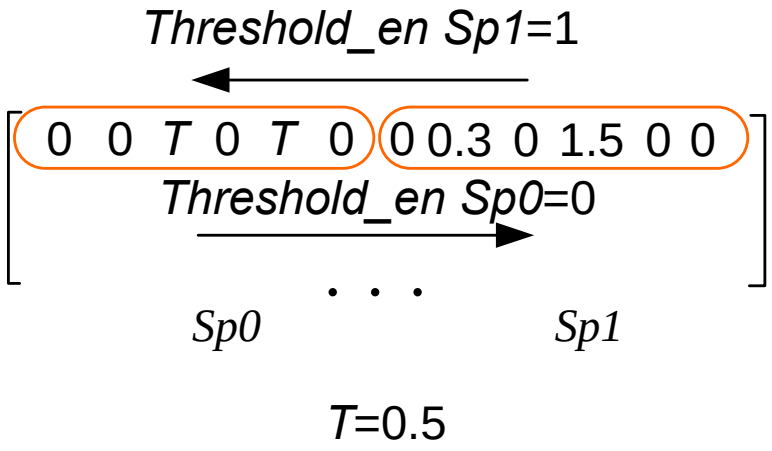
Outline

- Introduction to LDPC Codes and Iterative Decoding
- Goals and Key Features
- **Split-Row Threshold Decoding Method**
- Error Performance Results
- Split-Row Threshold Decoder Implementation and Results
- Conclusion

MinSum Split-Row Threshold Algorithm

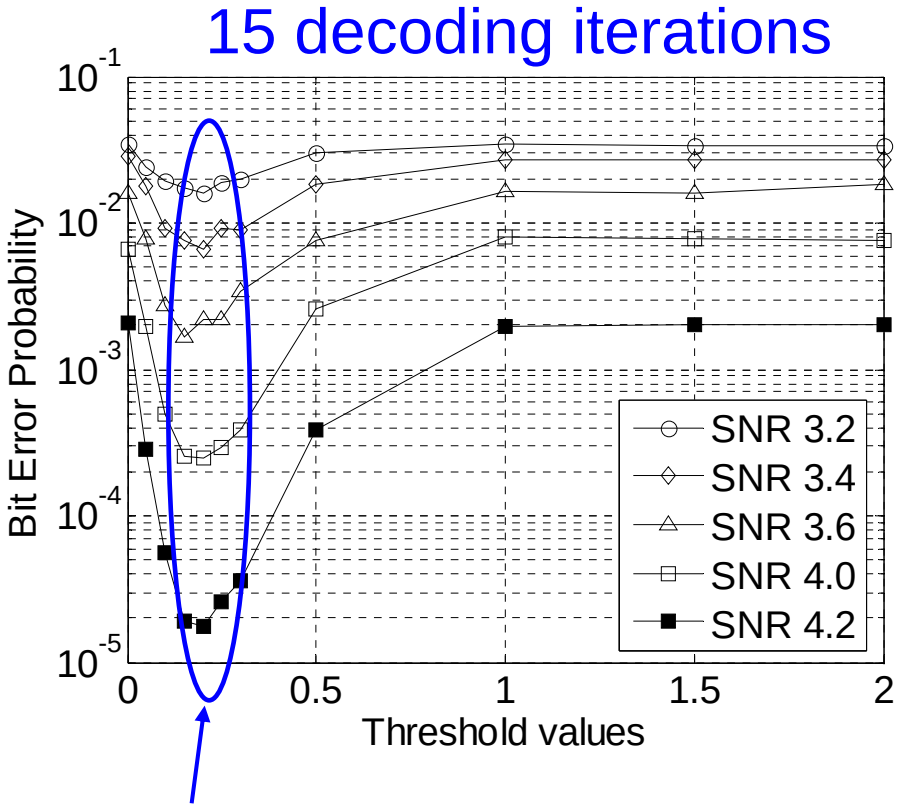
- A signal (*Threshold_en*) is passed from each partition, which indicates whether a partition has a minimum less than a given threshold (*T*).
- Based on *Threshold_en* status, the check nodes take as their minimum of their own local *Min* or *T*.
- Optimum threshold value (*T*) is obtained by empirical simulations

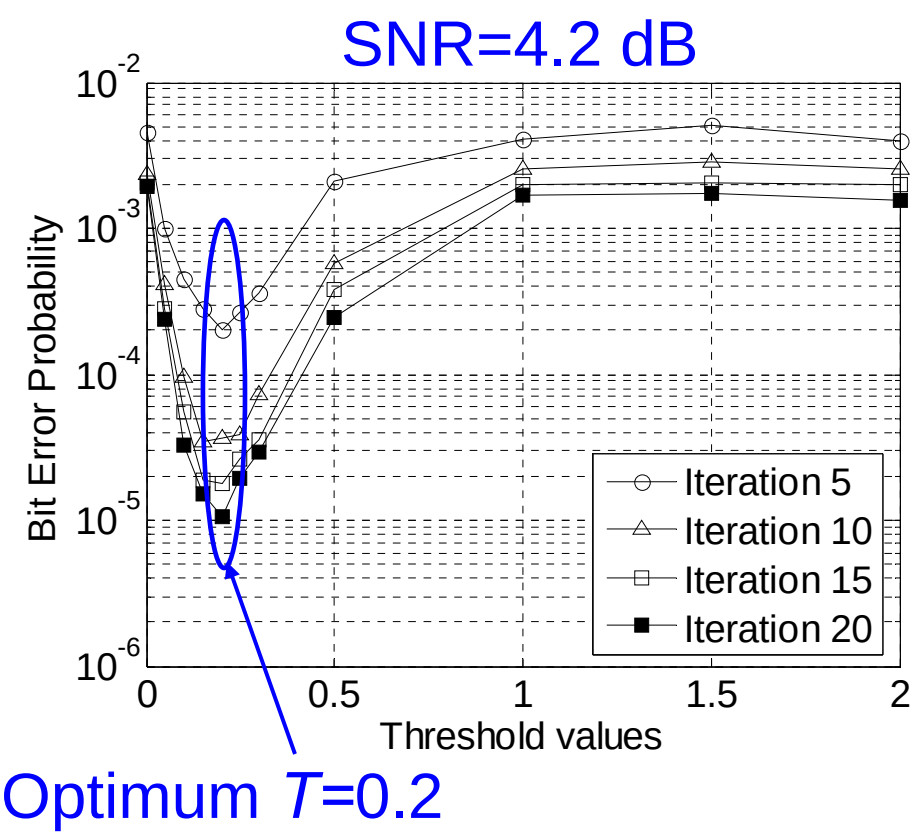




Mohsenin et al: *Asilomar 2008, ICC 2009, ISCAS 2009*

Impact of Threshold Selection





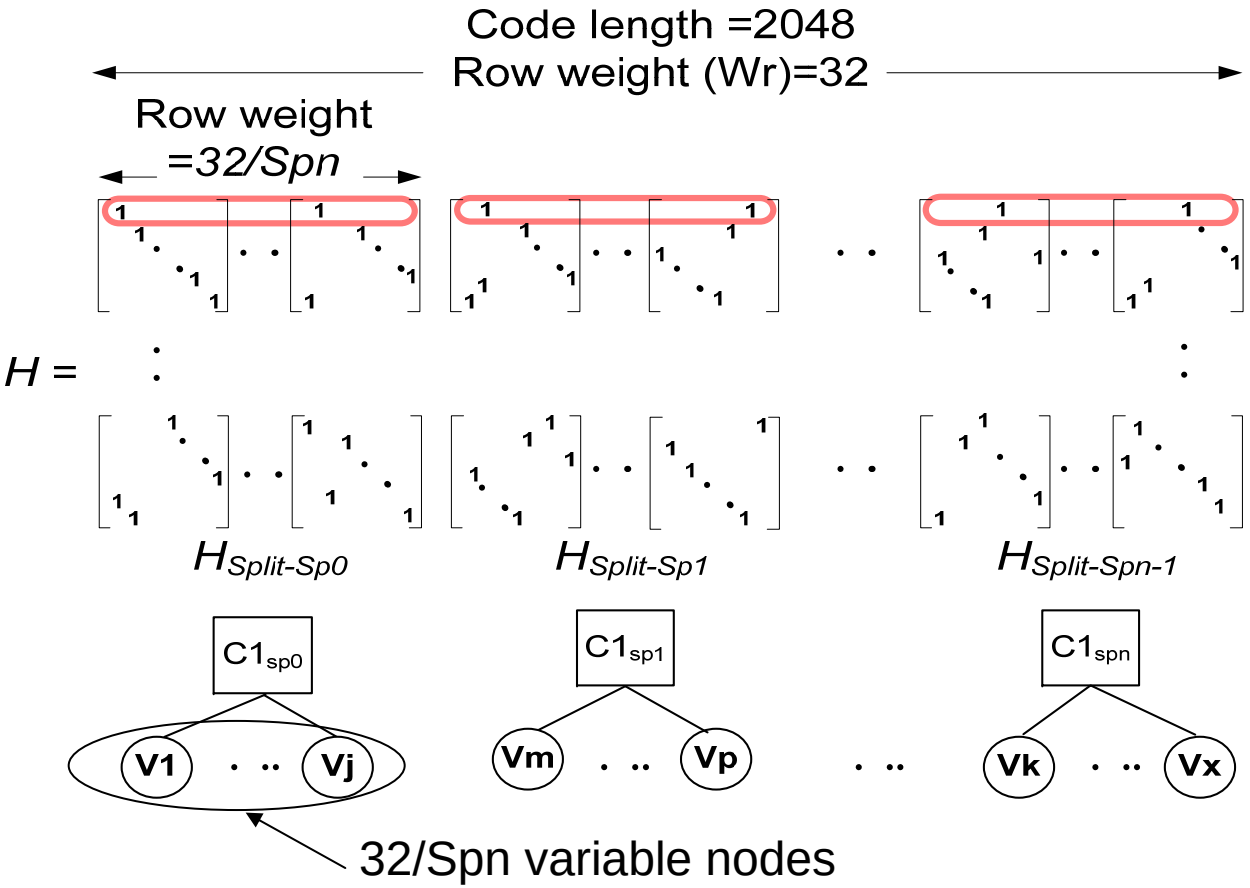
- (6,32) (2048,1723) LDPC Code
- Optimum threshold (*T*) is independent of SNR and decoding iteration

Outline

- Introduction to LDPC Codes and Iterative Decoding
- Goals and Key Features
- Split-Row Threshold Decoding Method
- **Error Performance Results**
- Multi-Split-Row Threshold Decoder Implementations and Results
- Conclusion

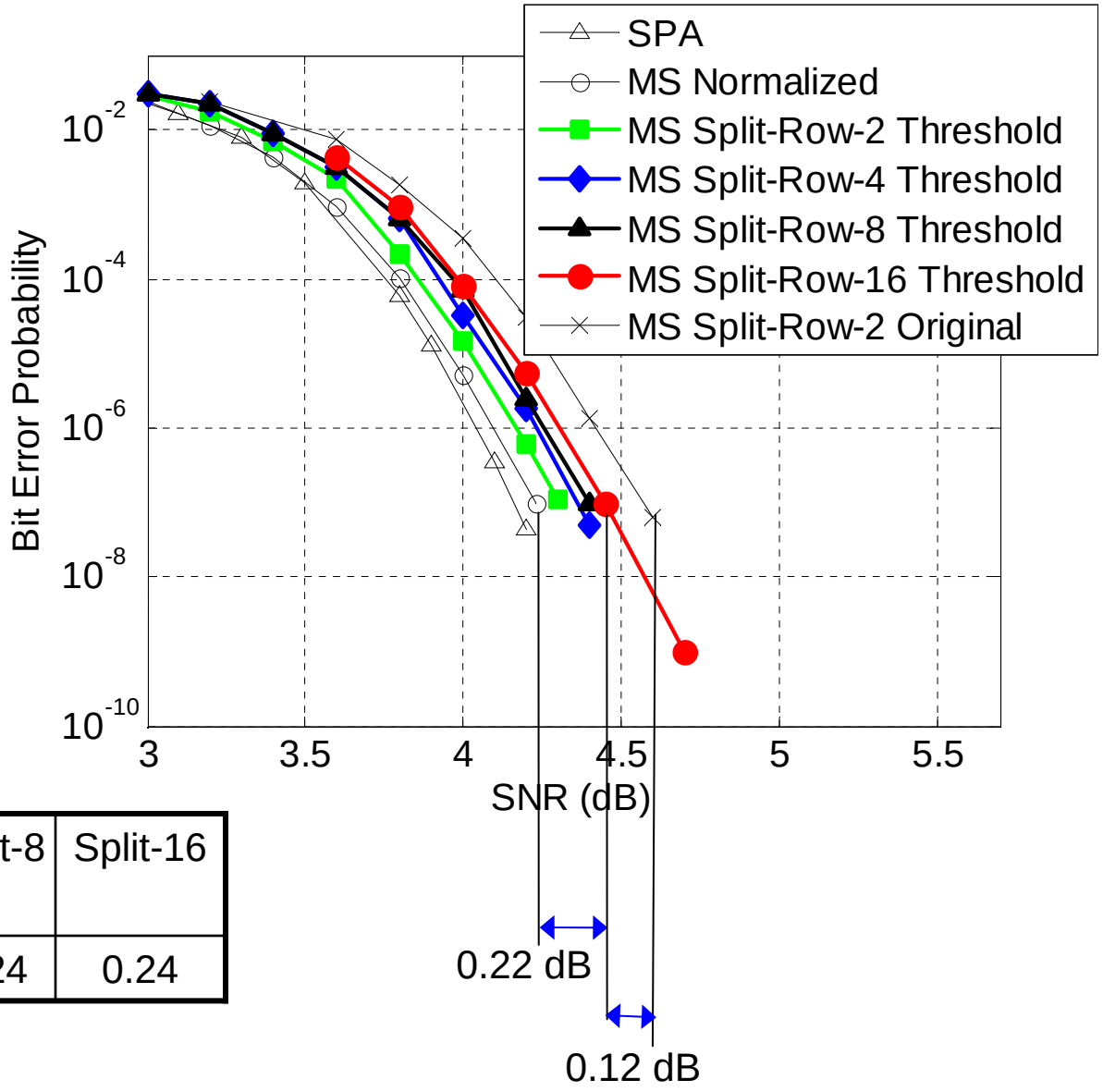
Multi-Split-Row Threshold Decoding

- Divide parity check matrix to *Spn* (*Spn*>2) partitions
- Partitioning can be arbitrary so long as there are at least two variable nodes per partition
- Example: (6,32) (2048,1723) LDPC Code



Error Performance for (2048,1723) 10GBASE-T Code

- MS Split-Row-2 Threshold is 0.07 dB away from MS
- MS Split-Row-16 Threshold is 0.22 dB away from MS and is 0.12 dB better than Split-Row-2 Original.



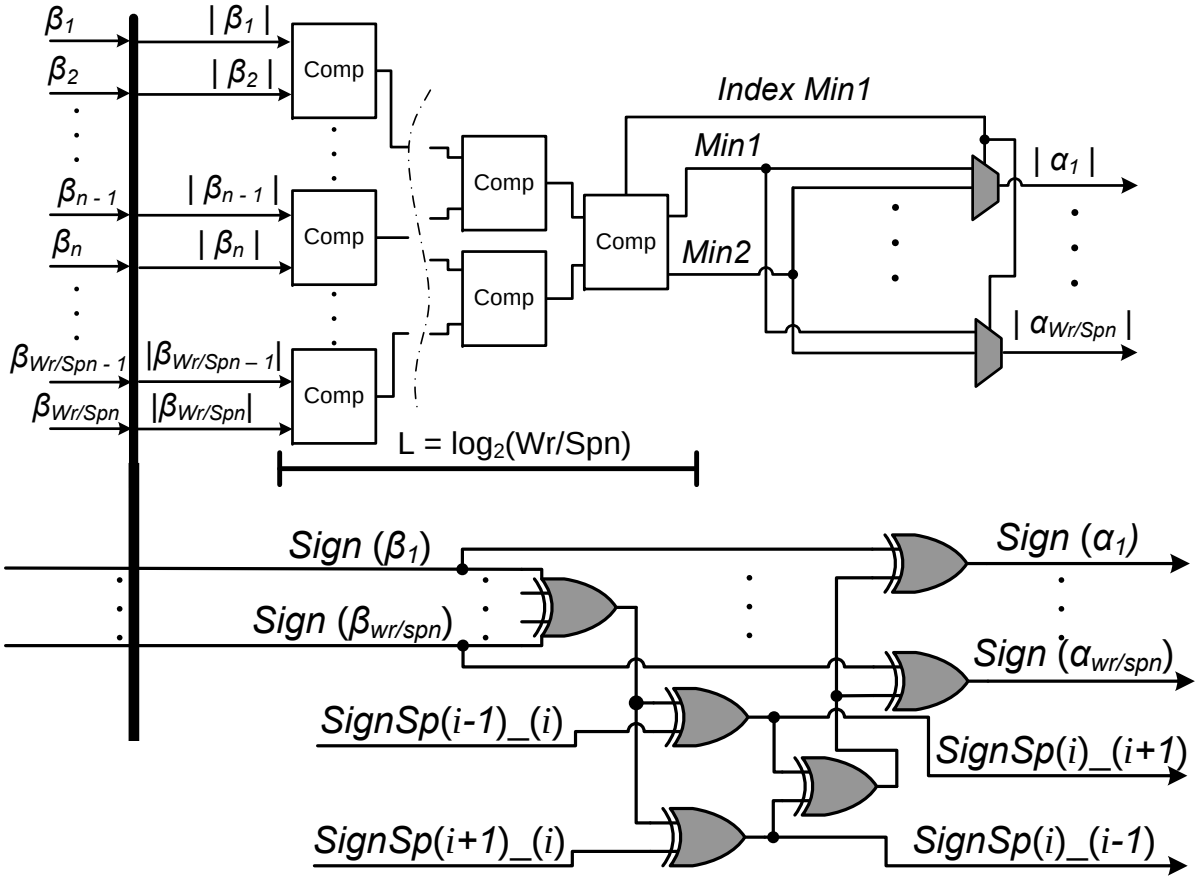
Decoder	Split-2	Split-4	Split-8	Split-16
Optimum *T*	0.2	0.23	0.24	0.24

Outline

- Introduction to LDPC Codes and Iterative Decoding
- Goals and Key Features
- Split-Row Threshold Decoding Method
- Error Performance Results
- **Multi-Split-Row Threshold Decoder Implementations and Results**
- Conclusion

Check Node Processor: Split-Row (original)

- The check node computes the row update equation
- Split-Row takes the MinSum check node processor and breaks it into two or more simpler row processors
  - Simplification of comparator tree
  - Number of check node I/Os reduced



cost of at most 3 XOR gates and a couple of sign wires while significantly reducing interconnections

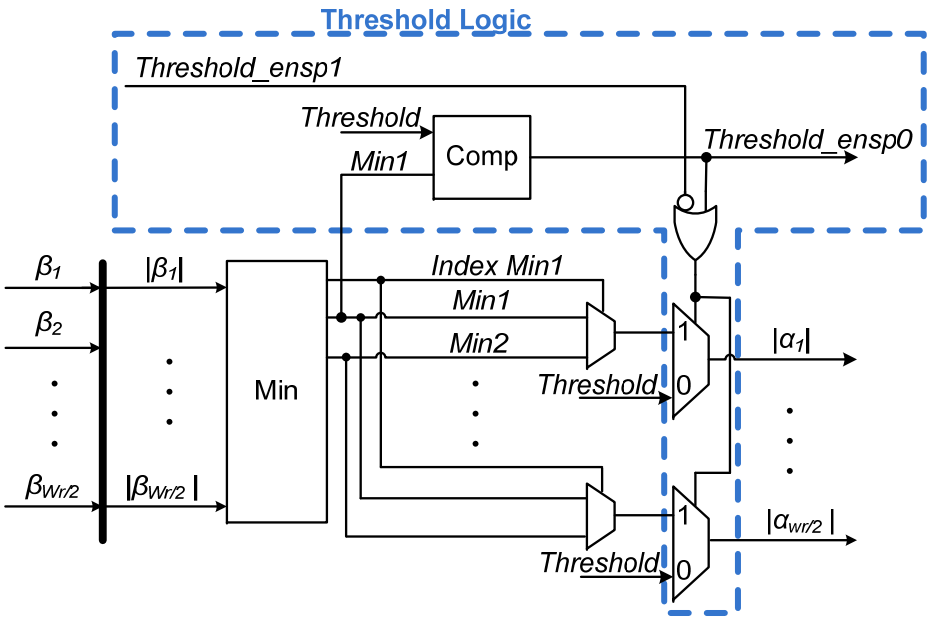
$$\alpha_{ij\,MSSplit\text{-}Row} = S_{MSSplit\text{-}Row} \times \left(\prod_{j', h_{ij'}=1, j' \neq j} sign\beta_{ij'} \right) \times \min_{j', h_{ij'\,Split\text{-}Row}=1, j' \neq j} \left(\left| \beta_{ij'} \right| \right)$$

Check Node Proc.: Split-Row Threshold

- Split-Row's loss of global minima transmission causes poor BER
- This can be overcome if we compare a Split-Row partition's minima with a well chosen *Threshold*
  - Small HW overhead
    - 5% increase in area, 7% increase in gate count
    - Negligible effect on local critical path
  - Improved BER
    - 0.2 dB improvement over original Split-Row2

Pseudocode for Threshold algorithm (Split-Row2)

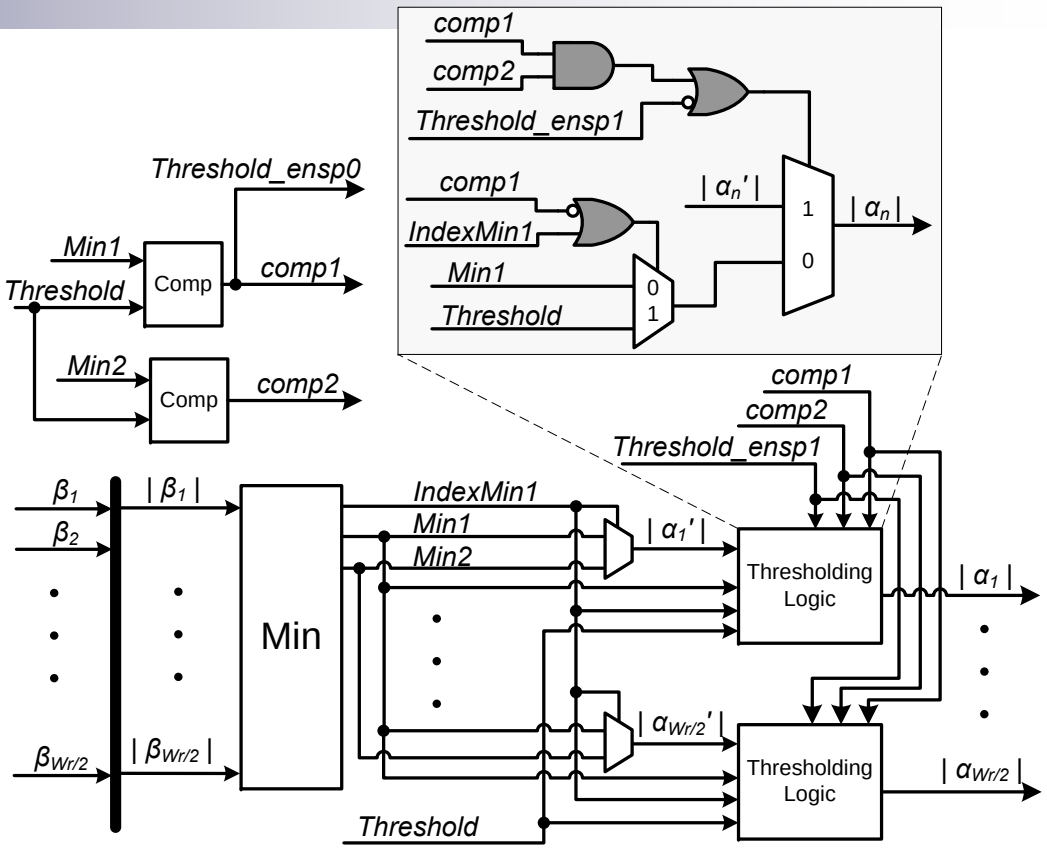
If $(Min1 \leq Threshold)$
  $Threshold\_ensp0 = 1$,
  if $(|\beta_j| = Min1)$
    $|\alpha_j| = Min2$
  else
    $|\alpha_j| = Min1$
else if $(Threshold\_ensp1 = 1)$
  $|\alpha_j| = Threshold$
else
  if $(|\beta_j| = Min1)$
    $|\alpha_j| = Min2$
  else
    $|\alpha_j| = Min1$.



T. Mohsenin, P. Urard and B. Baas, "A Thresholding Algorithm for Improved Split-Row Decoding of LDPC Codes" *Asilomar Conference on Signals, Systems and Computers (ACSSC),* October 2008.

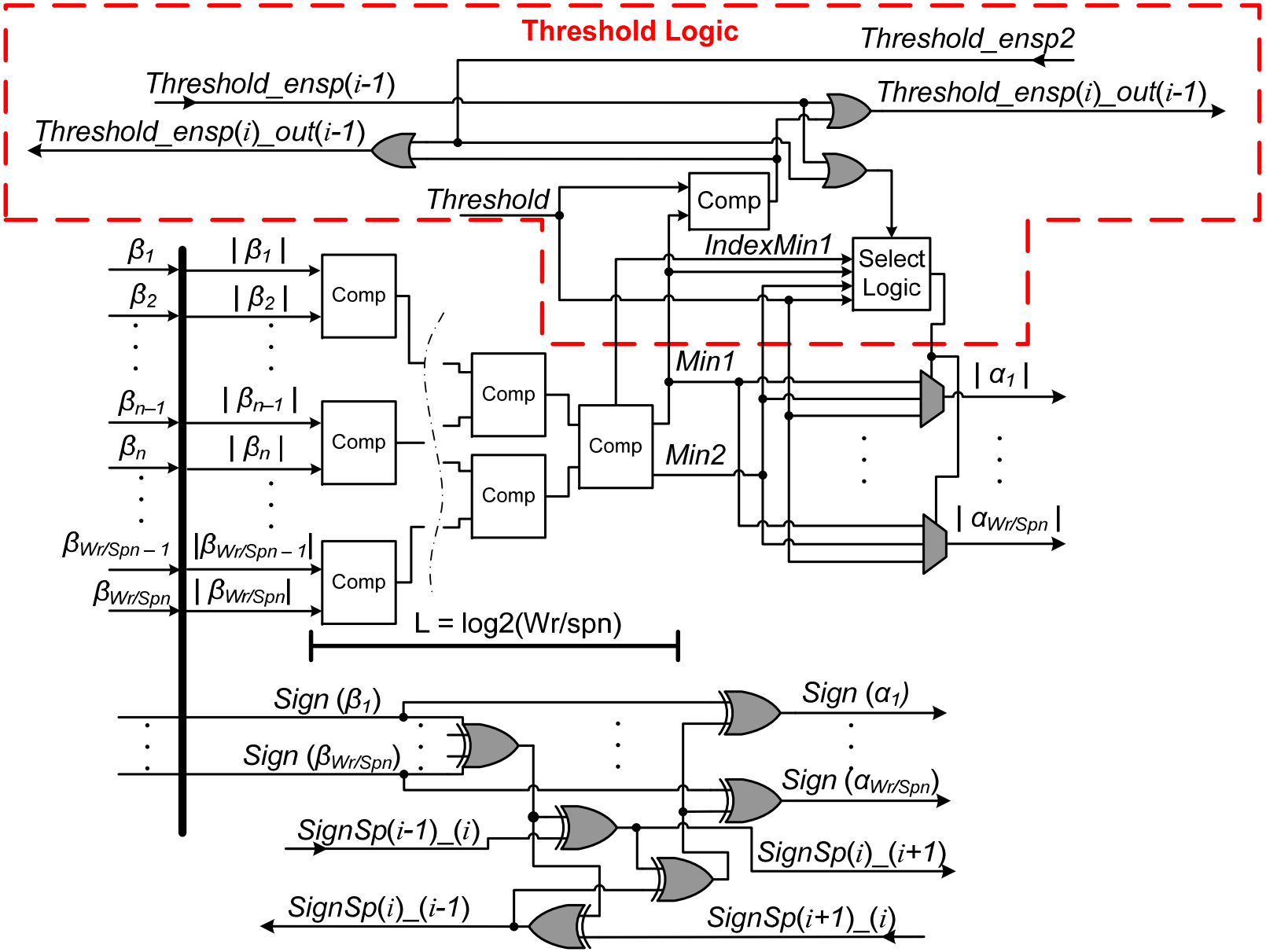
Check Node Proc.: Split-Row Threshold Improved

- Considering the 2nd minima (*Min2*) requires more complex logic
  - Additional HW includes two comparators and new select-mux logic
- Split-Row2 Threshold Improved BER is 0.07dB from original normalized MinSum
- Split-Row16 Threshold Improved Check Node Processor area is over 10x smaller than normalized MinSum at half the latency



Check Node Processor Synthesis Results (65nm)	Area (μm²)	Gate count	Delay (ns)
MinSum (MS)	3578	1018	2.0
MS Split-Row2 (original)	1767	541	1.4
MS Split-Row16 (original)	250	85	0.8
MS Split-Row16 Threshold Improved	317	95	0.9

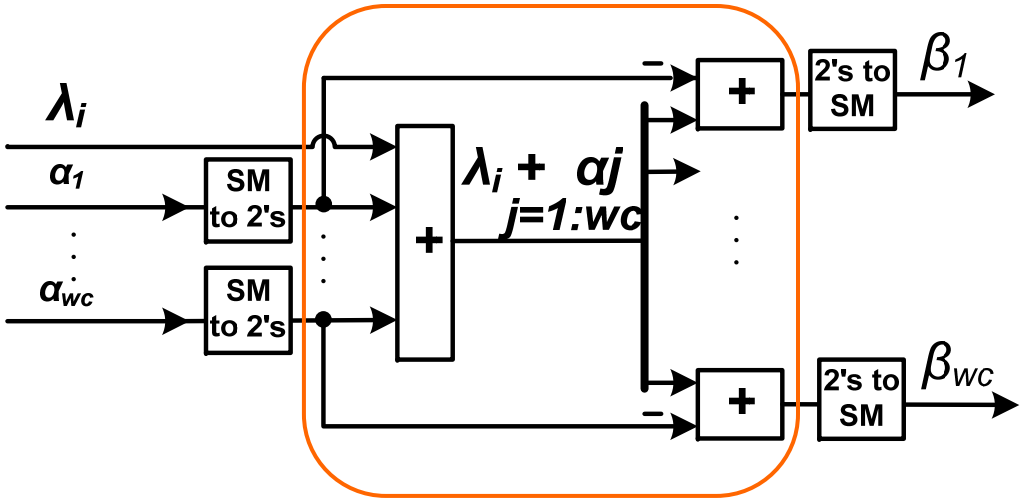
Check Node Proc.: Multi-Split-Row Threshold Improved

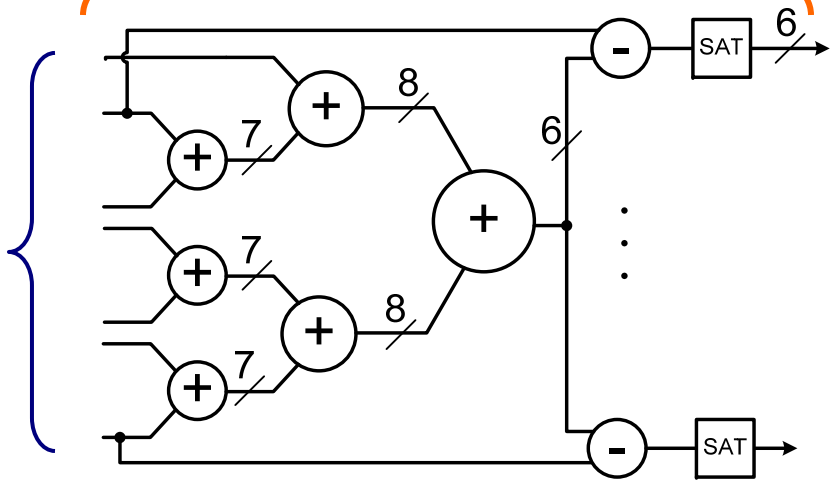


Variable Node Processor

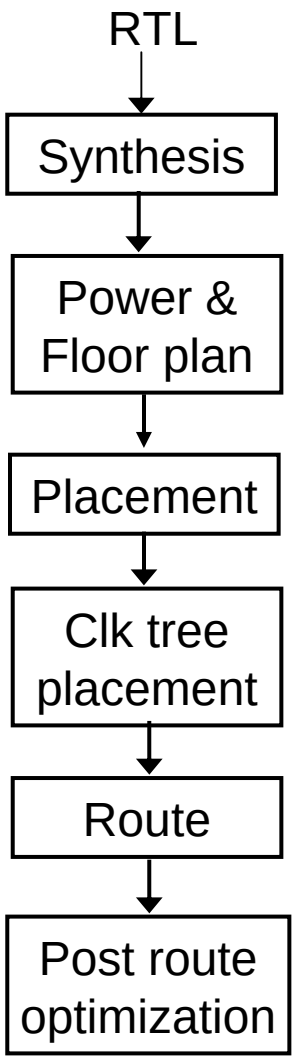
- Based on the column update equation
  - Split-Row leaves this unchanged from the original MinSum and SPA algorithms
- Variable node hardware complexity complexity is mainly reduced via wordwidth reduction

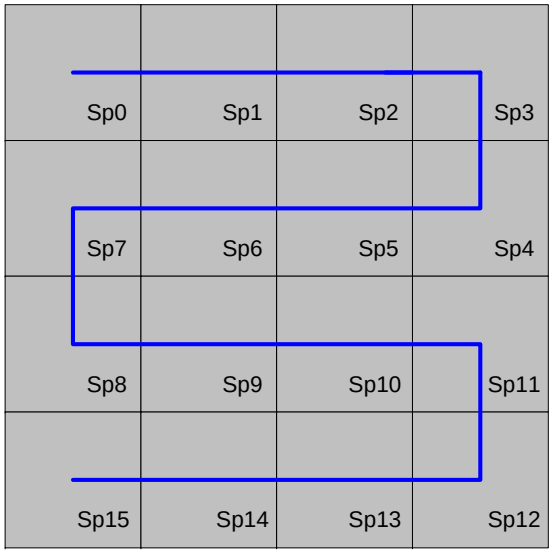
$$\beta_{ij} = \sum_{j', h_{ij'}=1} \alpha_{ij'} + \lambda_j$$

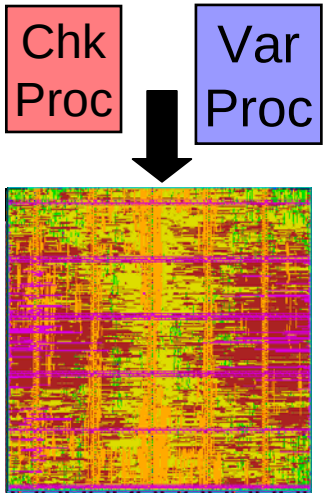


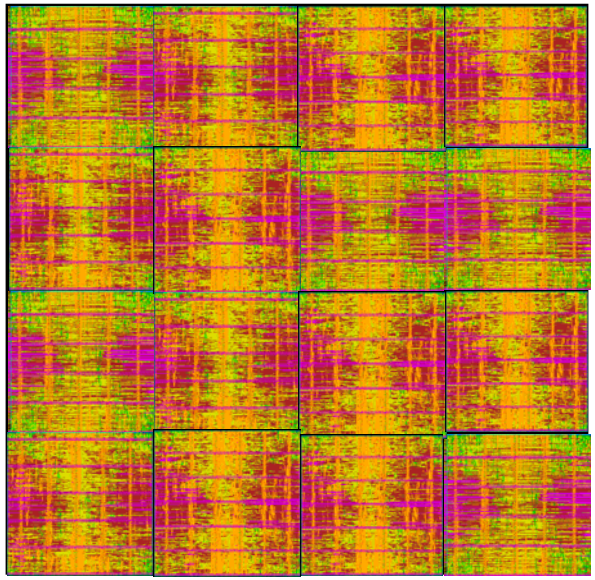


Multi-Split-Row Threshold Decoder Physical Layout

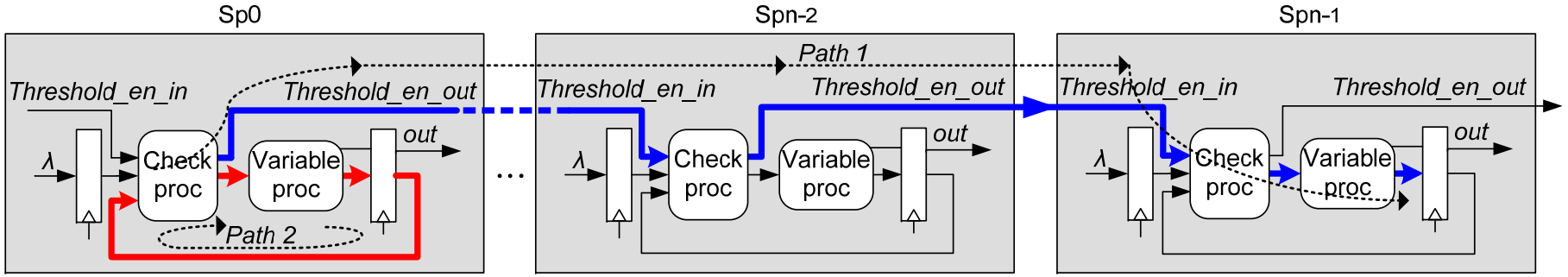




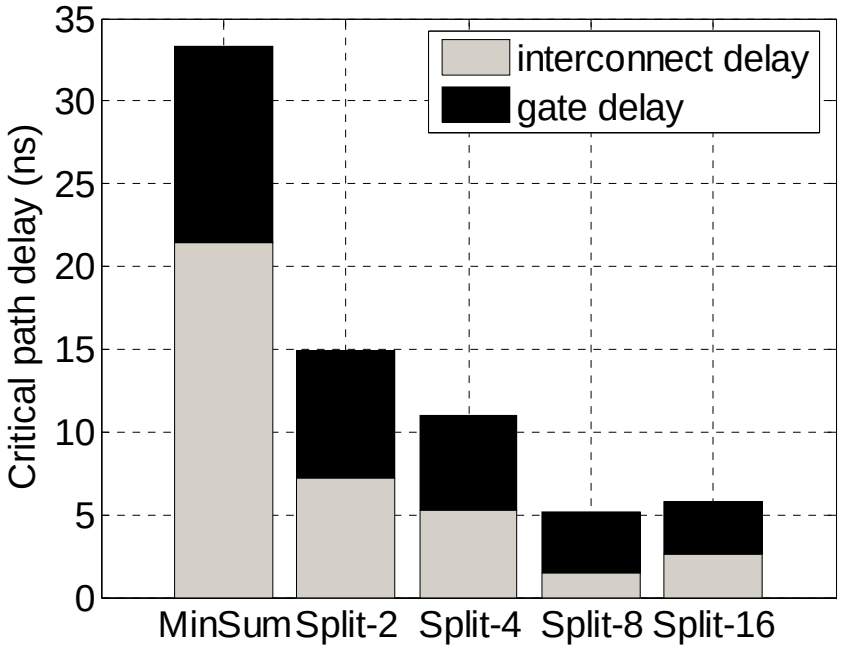




Delay Analysis for Decoders

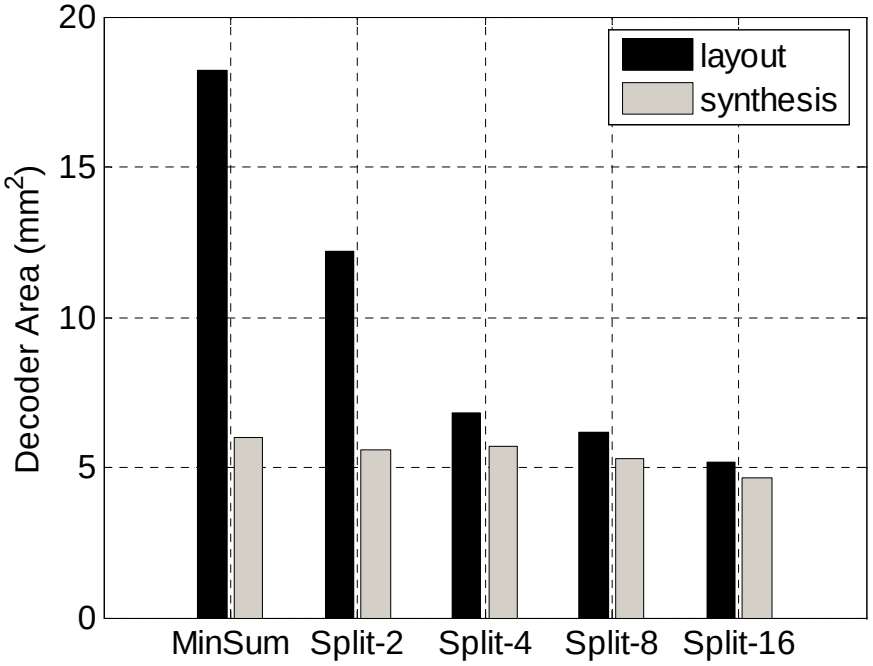


- Path1: propagation of *Threshold_en* passing through Spn-2 partitions
- Path2: delay path through check and variable procs
- For small *Spn* the interconnect delay is dominant because of wire interconnect complexity
- As the number of partitioning increases Path 1 delay increases

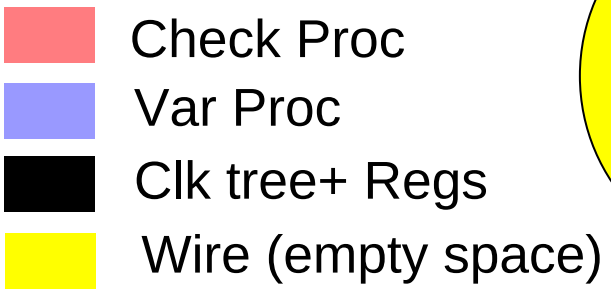


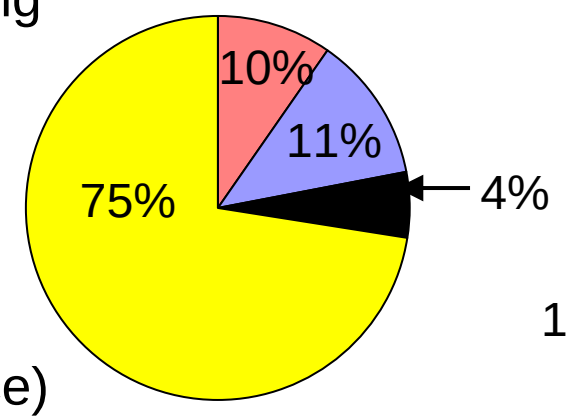
Area Analysis for Decoders

- In MinSum, the synthesis area deviates significantly from layout area due to low utilization.

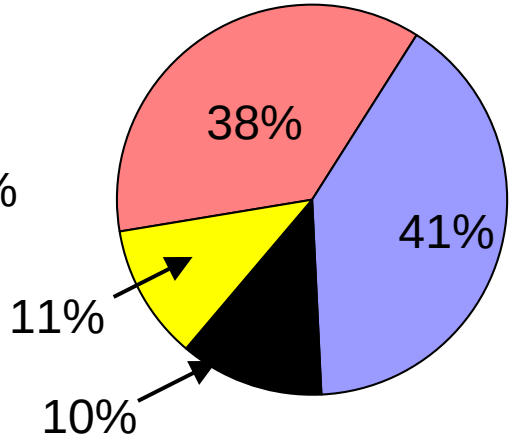


- Area break down per sub-block for MinSum and Split-16
  - 70% of MinSum decoder is empty space for wiring





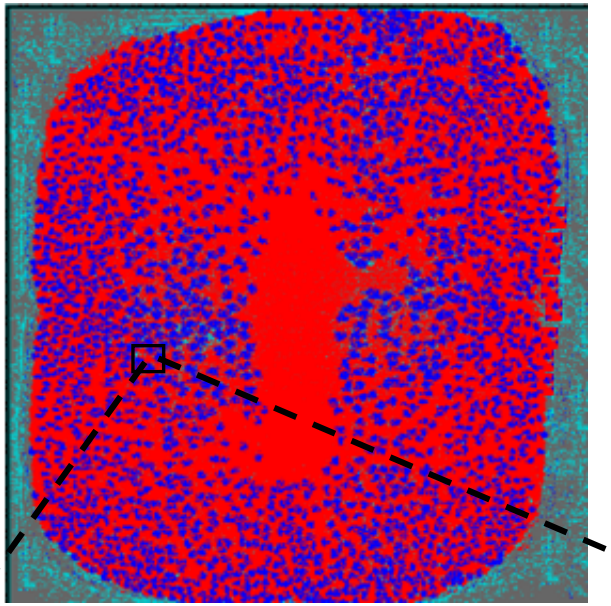
MinSum



Split-Row16 Threshold

Logic Utilization

MinSum

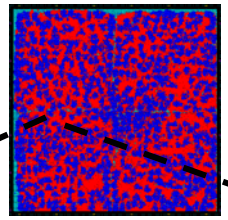


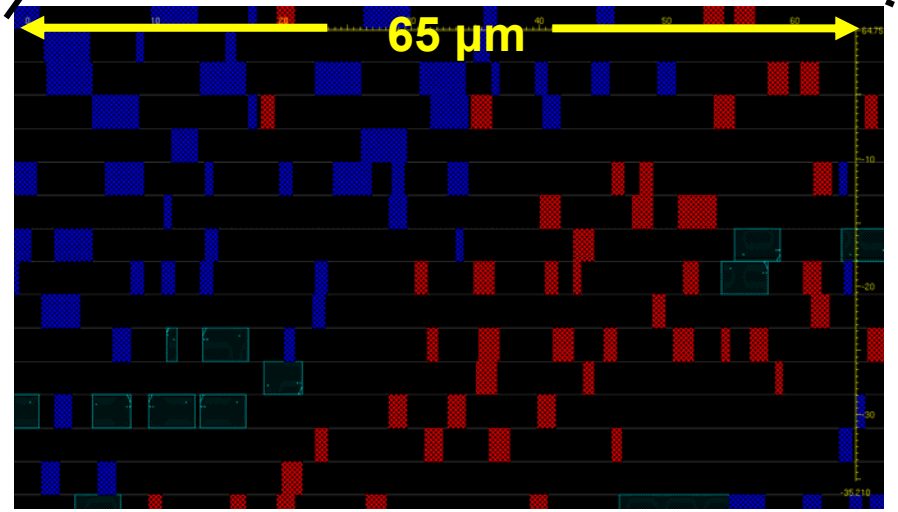
Variable processor

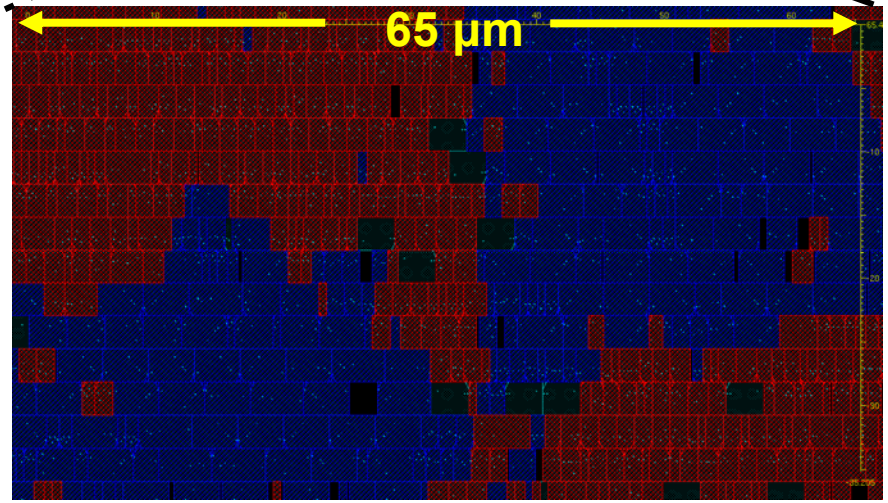
Check processor

Registers+buffers

SplitRow-16 Threshold one block, area not scaled

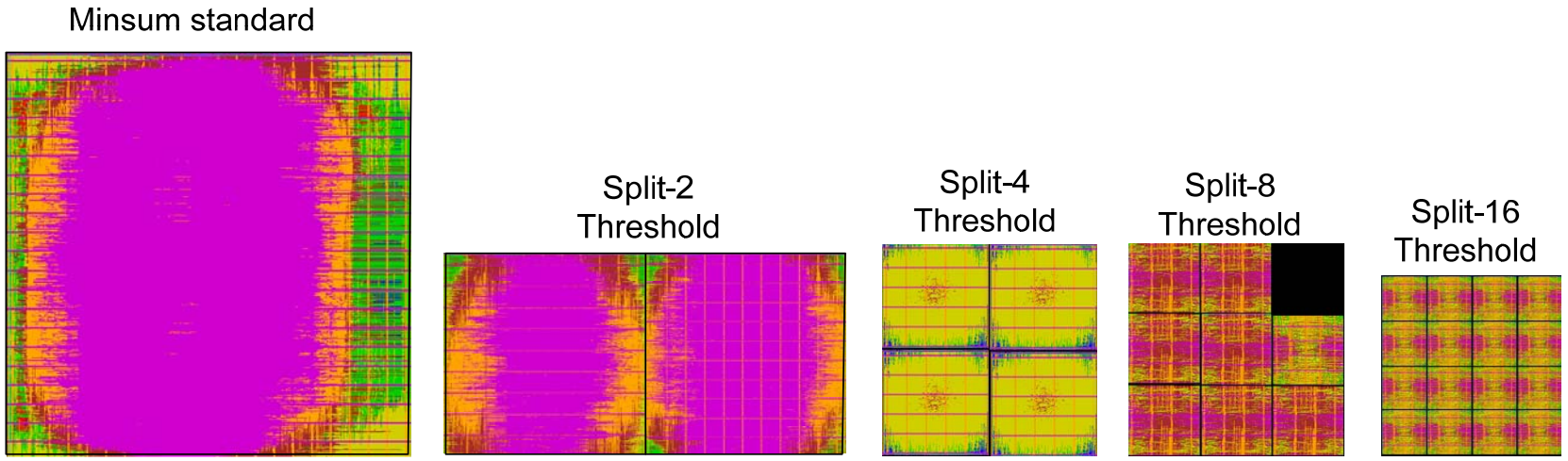






Comparison of Decoders

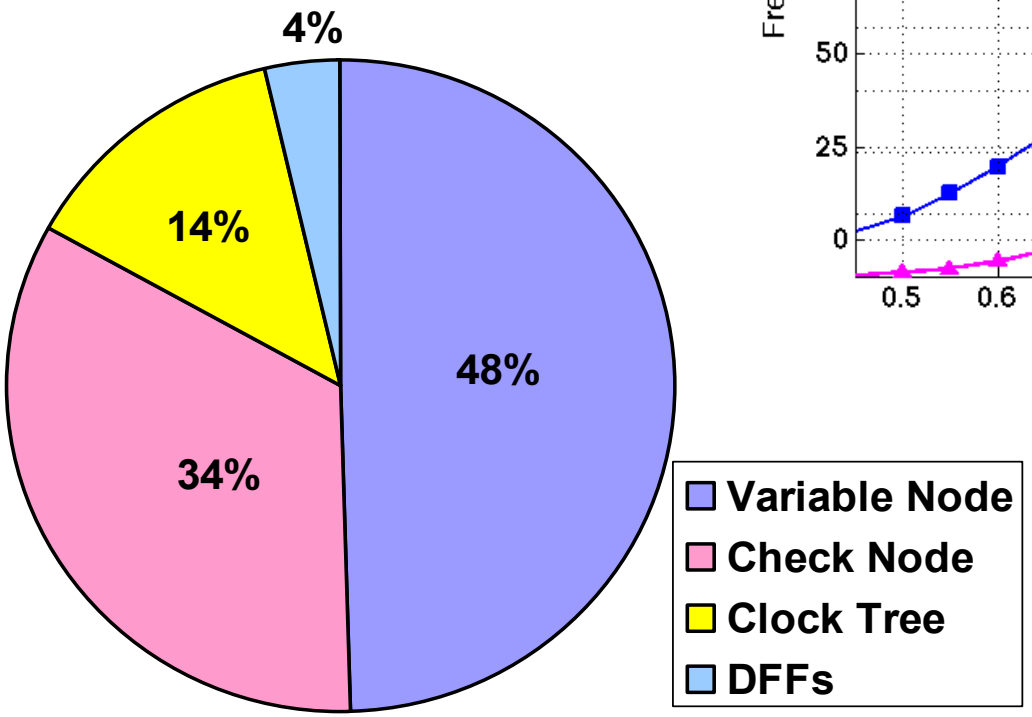
(6,32) (2048,1723) 10GBASE-T code with 11 decoding iterations.

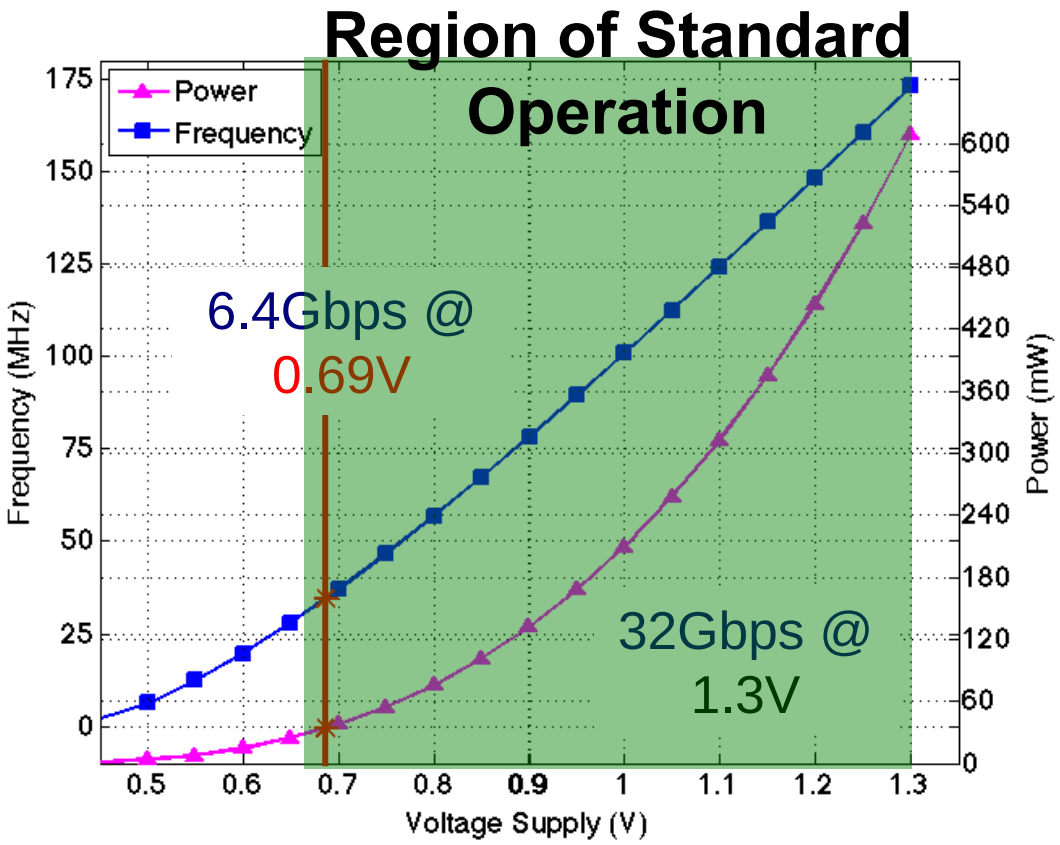


10GBASE-T Code 65 nm, 7 M, 1.3 V	MinSum standard	Split-2 Threshold	Split-4 Threshold	Split-8 Threshold	Split-16 Threshold	Split-16 vs.MinSum
Area Utilization	25%	40%	83%	86%	**89%**	**3.6x**
Area (mm^2)	18.2	12.2	6.8	6.2	**5.2**	**3.5x**
Speed (MHz)	30	67	91	**192**	**173**	**5.8x**
Throughput @ 11 iter (Gbps)	5.6	12.5	16.9	35.7	**32.2**	**5.7x**
CAD Tool CPU Time (hour)	>78	36	18	10	**5**	**>15.6x**

Power Analysis for Split-Row16 Decoder

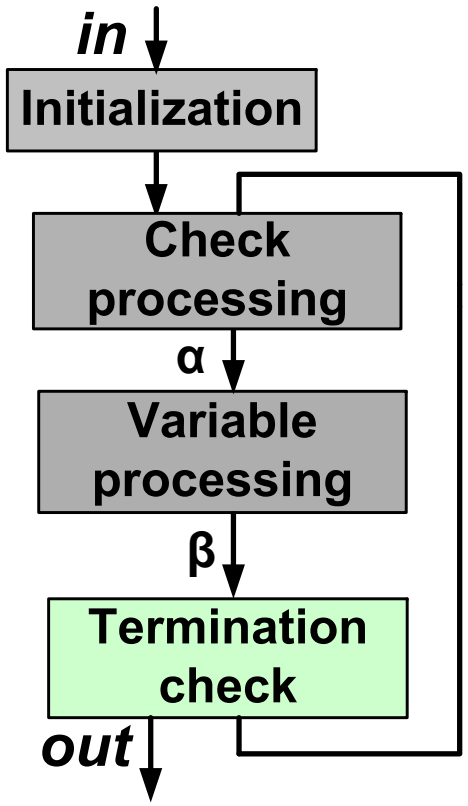
- Predicted voltage scaling on ST 65nm
- Power breakdown (under heavy activity)

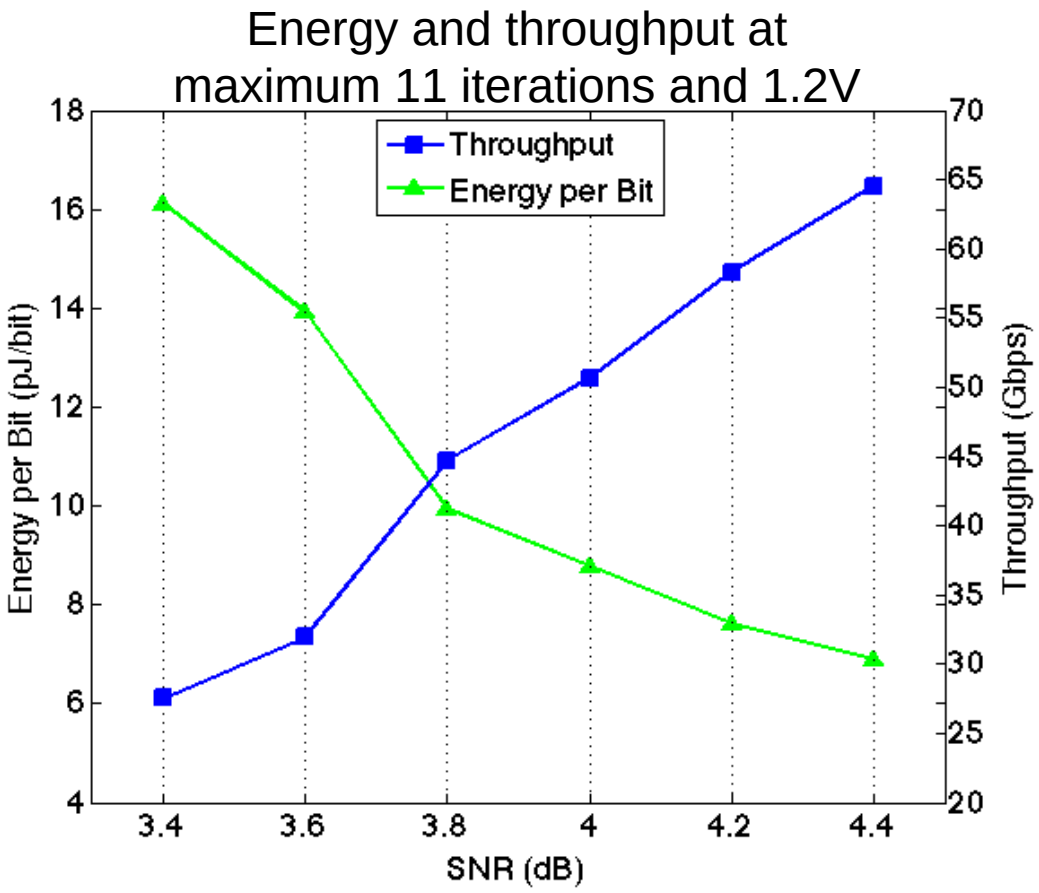




0.69V: 34MHz, 34mW
1.20V: 148MHz, 444mW
1.30V: 173MHz, 608mW

Early Termination for Split-Row16 Decoder

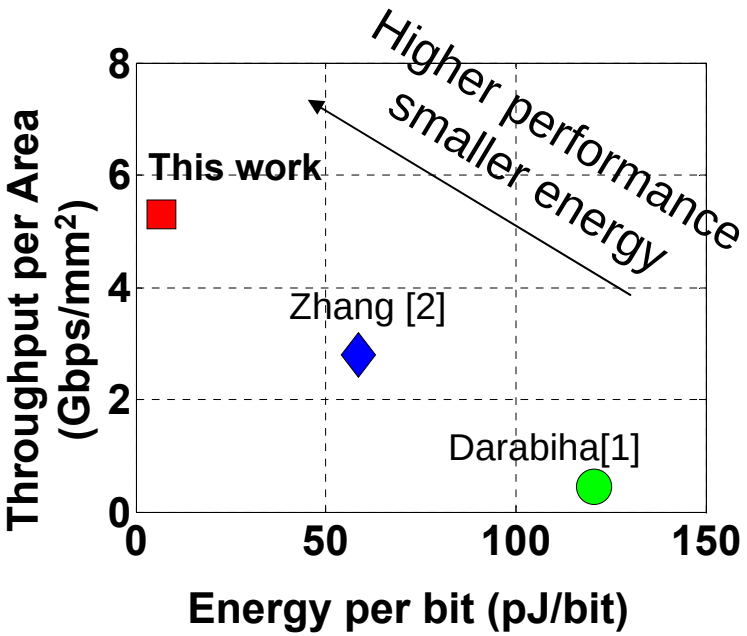




- With early termination a high energy efficiency for a variety of SNRs can be achieved
  - @ 3.4dB: 16.1pJ/bit 27.5Gbps
  - @ 4.4dB: 6.9pJ/bit 64.5Gbps

Comparison with Previous Work

	Darabiha [1]	Zhang [2]	Liu [3]	**This work (Split-16)**
LDPC Code	(4,15) (660,480)	(6,32) (2048, 1723)		
Technology	130 nm, -	65 nm, 7M	90 nm, 8M	65 nm, 7M
Voltage (V)	1.2	1.2	-	1.2
Word length (bit)	4	4	6	5
Utilization	72%	80%	50%	**89%**
Area (mm^2)	7.3	5.35	9.8	**5.2**
Speed (MHz)	300	700	207	**148**
Early Termination	Yes	Yes	No	Yes
Max Iteration (Imax)	15	8	16	**11**
Throughput (Gbps)	3.3	47.7	5.3	**64.5**
Power (mW)	398	2800	-	**444**
Energy per bit (pJ/bit)	8.0	58.7	-	**7.0**



[1] A. Darabiha et al., *JSSC.*, 2008

[2] Z. Zhang et al., *VLSI Symp.*, 2009

[3] L.Liu et al., *TCAS I*, 2008

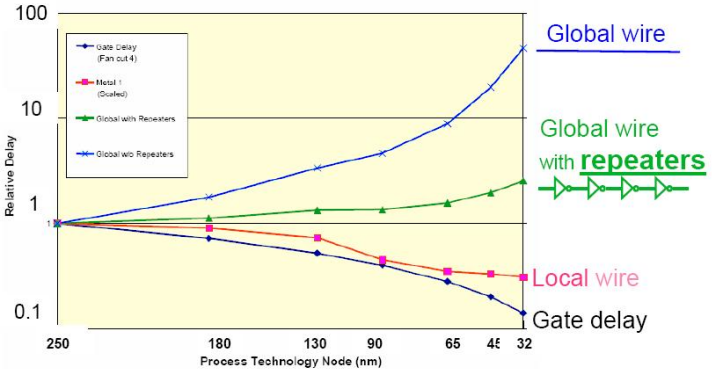
Future of LDPC in Deep Submicron CMOS

- New LDPC codes are being studied and constructed trying to balance theoretical performance and practical hardware realization
  - However, code theorists generally are not concerned with transistor power and area
- 32nm technology and below present increased restrictions on the freedom of the backend designer, while wire delay is still increasing
  - Must reduce design dependency on low-level optimizations for success
- The Split-Row technique presents an algorithmic and architectural solution that can be compatible with both future LDPC codes and submicron CMOS technology

$$H =$$

Low-density parity check matrix: N=20000 M=10000 (From: *Information Theory, Inference, Learning Algorithms*, D. MacKay)

Gate delay gets better, wire delay gets worse



Delay for Metal 1 and Global Wiring versus Feature Size

http://www.gtsav.gatech.edu/candle/research.html

Conclusion

- Split-Row reduces VLSI interconnect complexity through message passing reduction on row update
  - Partitioning reduces the number of connections between check and variable processors. This results in higher silicon utilization and smaller and efficient layouts.
- Threshold algorithm does not reduce the effectiveness of original Split-Row
  - At most two additional Threshold enable wires per row
- Improved Threshold algorithm increases error performance over original Split-Row
  - Split-Row2: 0.07 dB away from MinSum Normalized
  - Split-Row16: 0.12 dB better than Split-Row2 original
- Multi-Split Threshold allows us to use full parallel decoding for high speed applications with acceptable error performance loss, high energy efficiency and low area
  - @ 1.2 V and SNR = 4.4 dB: 64.5 Gbps, 444 mW, 7 pJ/bit

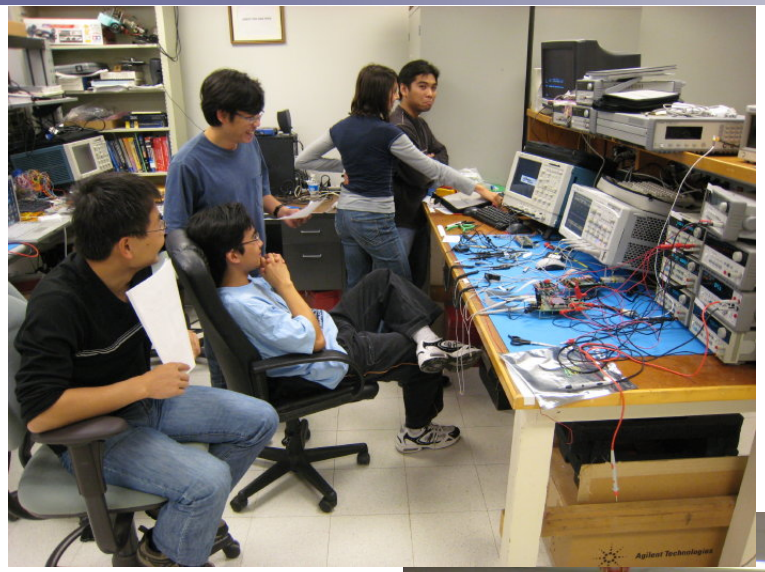
Acknowledgements

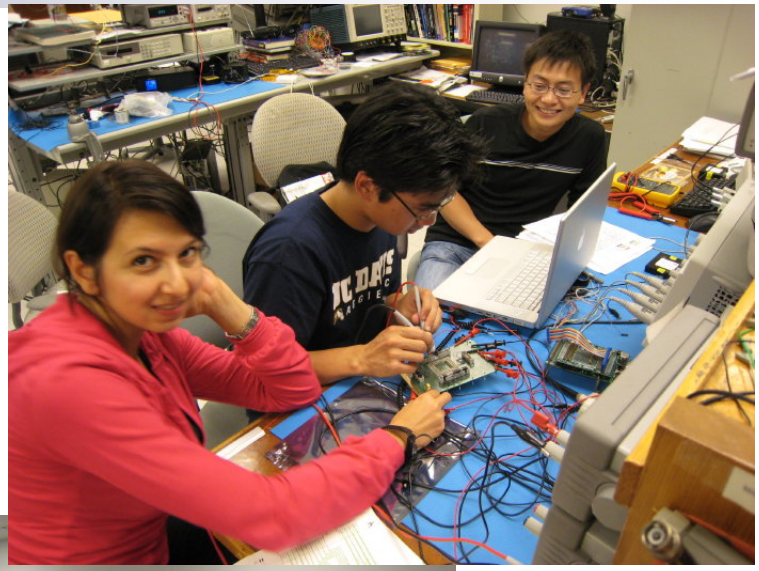
- Support
  - ST Microelectronics
  - NSF Grant 430090 and CAREER award 546907
  - Intel
  - SRC GRC Grant 1598 and CSR Grant 1659
  - Intellasys
  - UC Micro
  - SEM
- Special thanks
  - Professor Shu Lin

VLSI Computation Lab (VCL)

- Advisor: Professor Bevan Baas
- 7 PhD students
- 6 MS students
- 3 Undergraduate student
- Website: http://www.ece.ucdavis.edu/vcl/

VLSI Computation Lab (VCL)





Projects in VCL

- High performance and high energy efficiency Low Density Parity Check (LDPC) Decoders
- Programmable processors
  - Many-core DSP: AsAP 1.0 (36 processors), AsAP 2.0 (167 processors)
- Special purpose processors
  - FFT, Viterbi decoder, …
- Applications
  - H.264
  - Biomedical Applications
- Circuits
  - Dynamic frequency scaling (DVFS)
- Algorithms/Architectures
  - LDPC decoding
  - Network on chip