Information Optimized Quantized Message Passing for Near-Tbps Fully Pipelined LDPC Decoding

Alexios Balatsoukas-Stimming[1], Reza Ghanaatian[1],
Michael Meidlinger[2], Gerald Matz[2],
and **Andreas Burg**[1]

[2] Telecommunications Circuits Laboratory, EPFL
[2] Vienna University of Technology

February 27, 2019

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Telecommunications Circuits Laboratory

Low Density Parity Check (LDPC) Codes

1962: invented by **R. G. Gallager**

- Performance close to the Shannon limit
- Iterative decoding was initially **considered to complex for** economic **implementation**

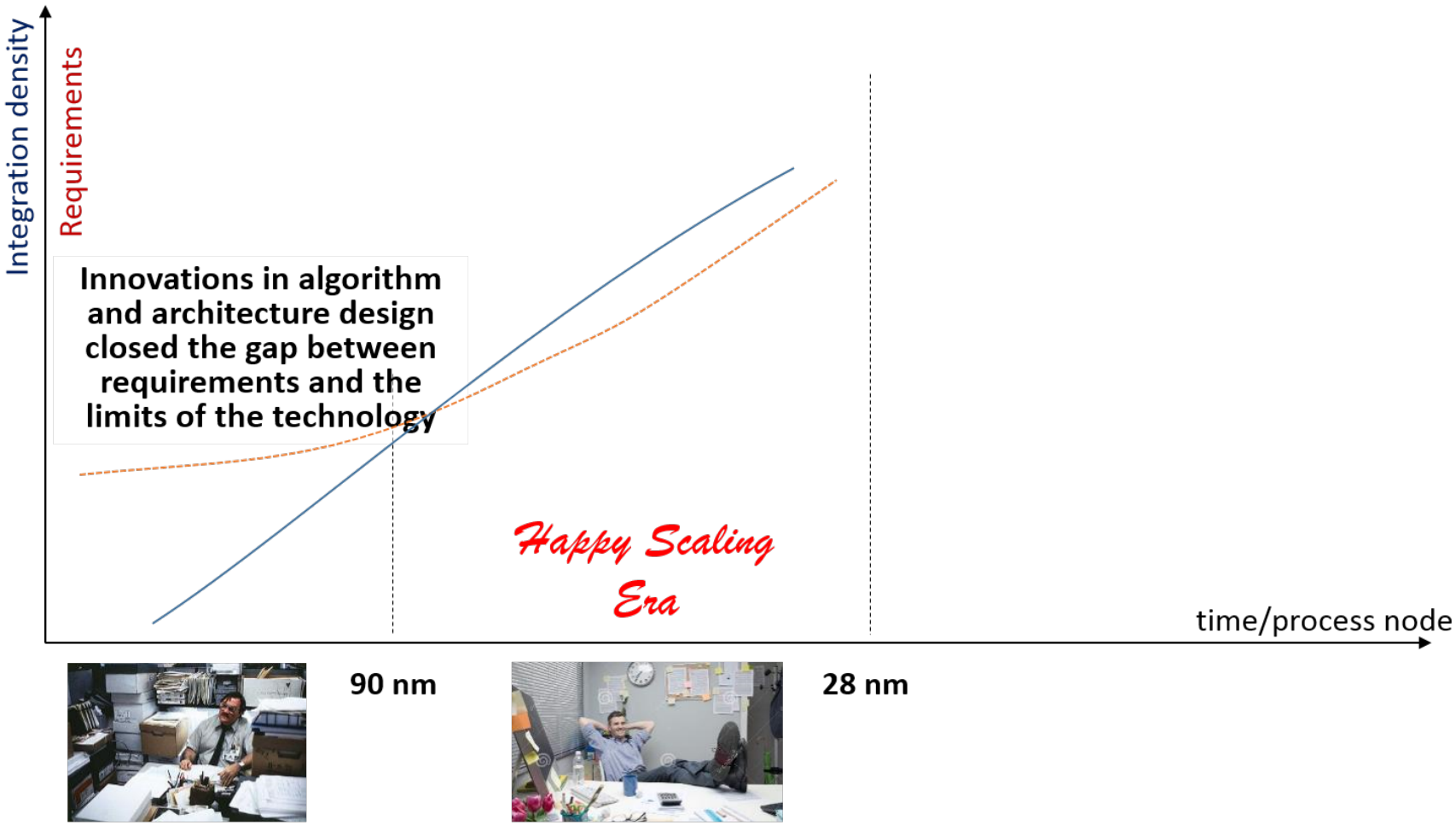
1999: re-discovered by MacKay and Neal

- VLSI technology allowed for the implementation of LDPC codes

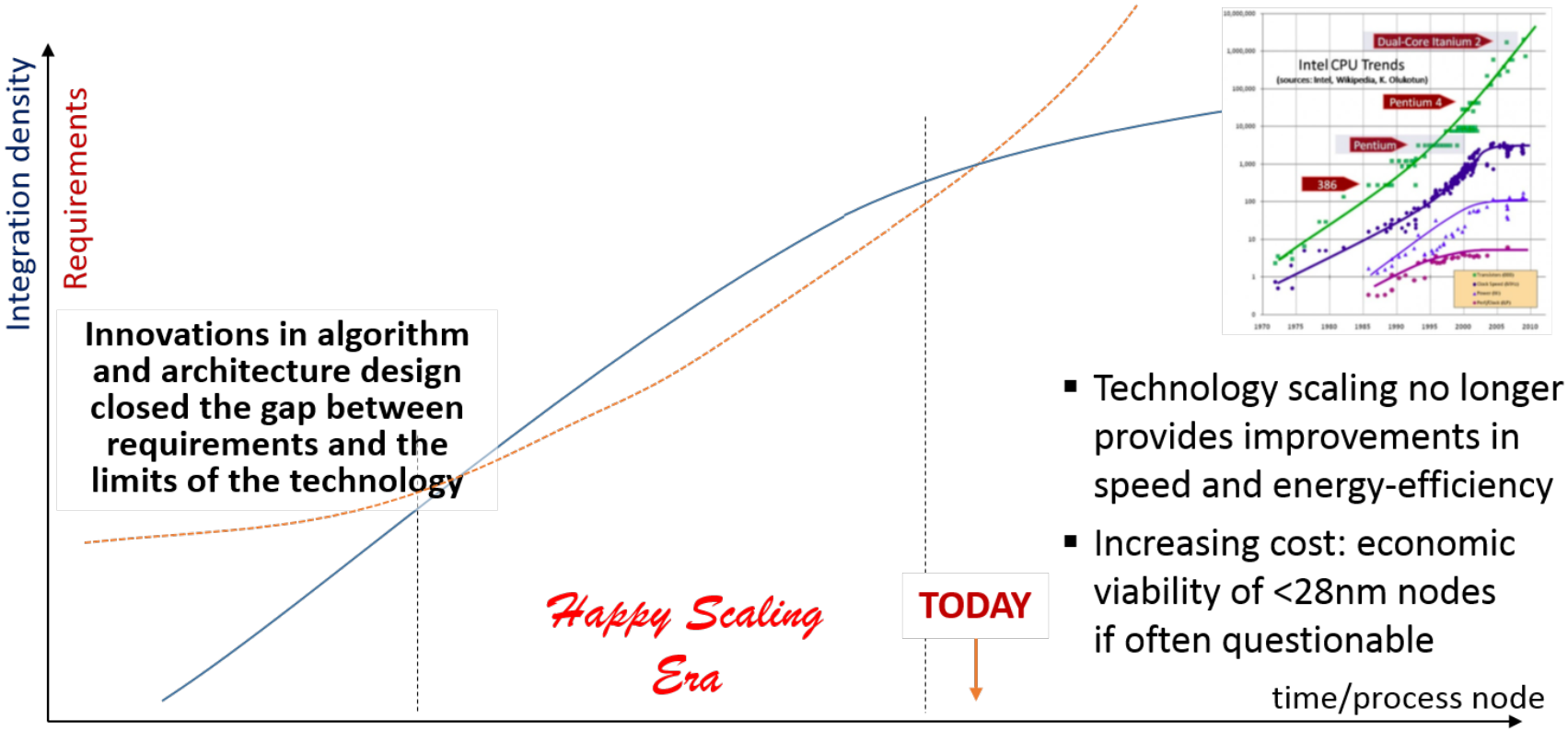
Today: LDPC codes are optional or mandatory in almost all standards

- Increasingly favored over other codes for high throughput.

The Dawn of the Happy Scaling Era



The Dawn of the Happy Scaling Era



- Technology scaling no longer provides improvements in speed and energy-efficiency
- Increasing cost: economic viability of <28nm nodes if often questionable

The Offset Min-Sum (MS) Algorithm

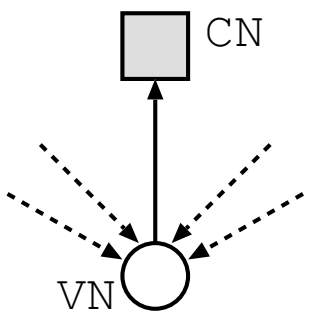
The Min-Sum algorithm and its variants are the workhorses of LDPC decoding.

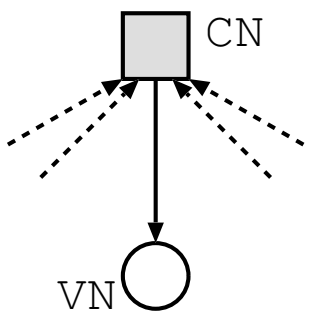
Initialization: Set L_v^0 based on LLRs from the demodulator and set $R_{c,v}^0 = 0$
Iterations: $i = 1 \dots I_{\max}$

$$\text{VN}: \quad Q_{v,c}^i = L_v^0 + \sum_{c' \in \mathcal{N}(v) \setminus c} R_{c',v}^{i-1}$$

$$\text{CN}: \quad R_{c,v}^i = \max\left(\min_{v' \in \mathcal{M}(c) \setminus v} \left(|Q_{v',c}| \right) - \beta, 0 \right) \prod_{v' \in \mathcal{M}(c) \setminus v} \operatorname{sign}\left(Q_{v',c}^i \right)$$

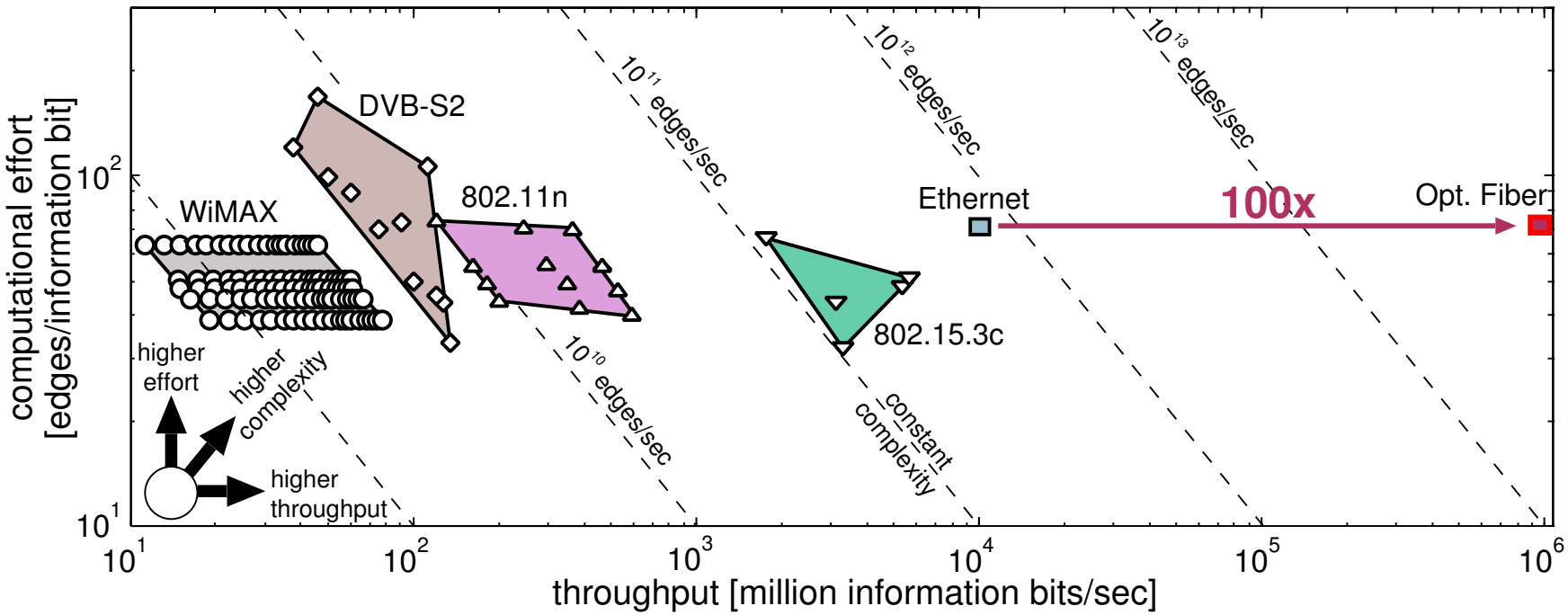
$$\text{VN}: \quad L_v^i = L_v^0 + \sum_{c' \in \mathcal{N}(v)} R_{c',v}^i$$





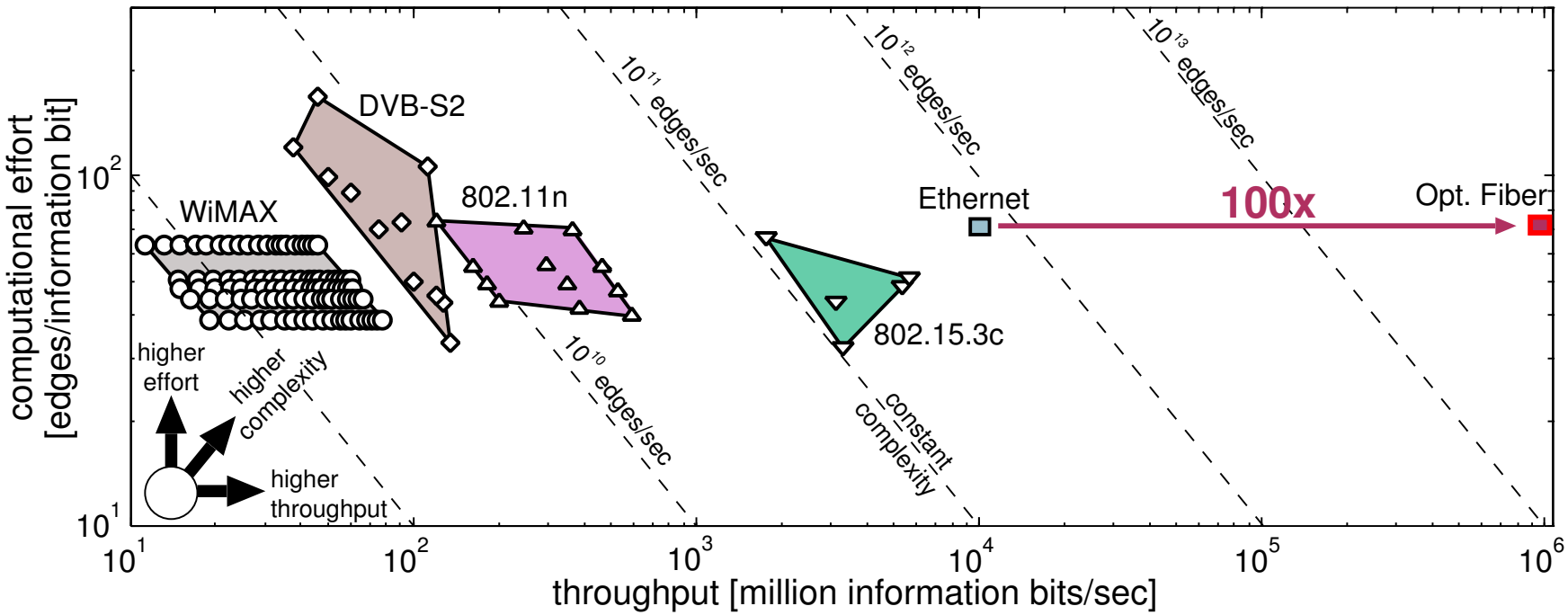
Computational Complexity of LDPC Decoding

Consider the **computational effort per information bit** and the **required throughput** for different standards and for their different operating modes



Computational Complexity of LDPC Decoding

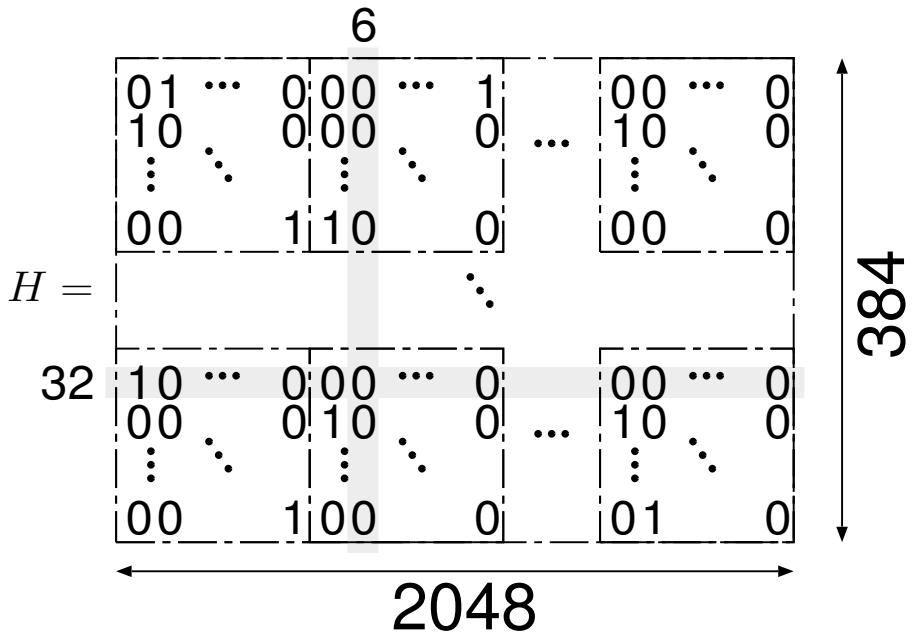
Consider the **computational effort per information bit** and the **required throughput** for different standards and for their different operating modes



VLSI architectures for LDPC decoding must **cover** more than **6 orders of magnitude in throughput** and **different degrees of reconfigurability**

10GBASE-T 10 Gbps Ethernet

10GBASE-T employs a (6,32)-regular (2048,1723) code with rate $R = 0.84$

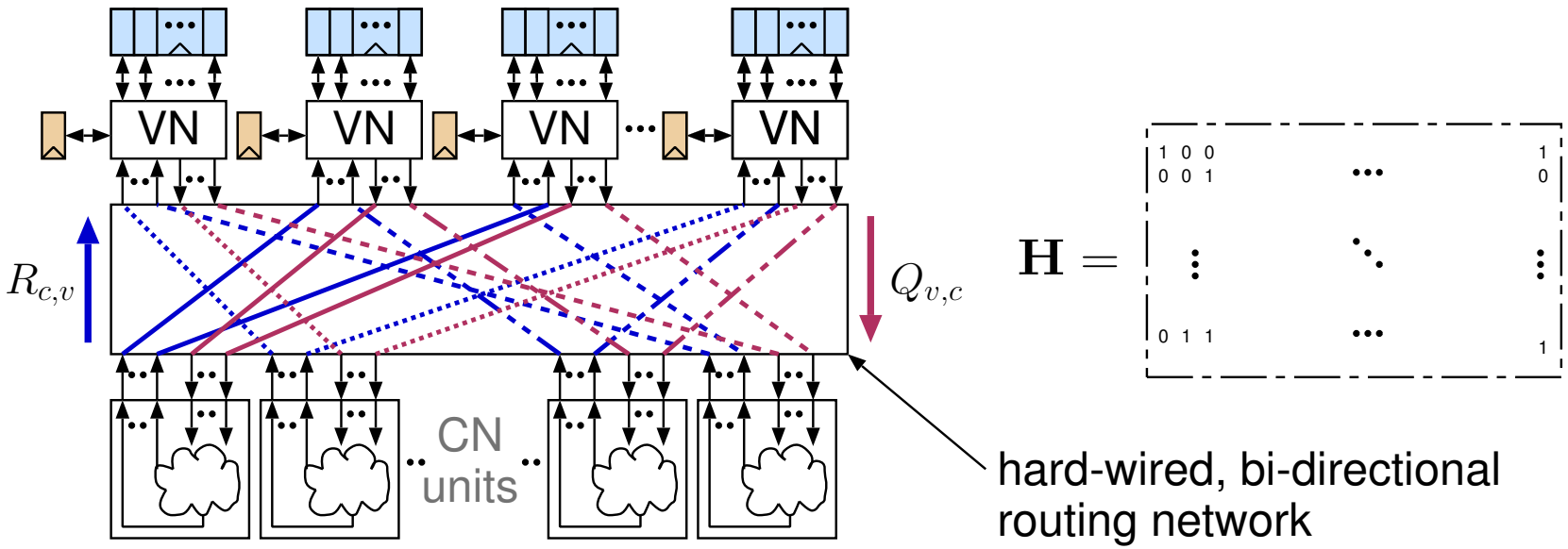


- 384 Check Nodes, 2048 Variable Nodes
- Organized in 6 layers
- 12'288 edges in the corresponding Tanner graph

Fully Parallel Implementation

Isomorphic architecture: direct mapping of Tanner graph onto silicon

- Instantiate **2048 VNs** and **384 check nodes**
- **Edges** are implemented through a global routing network
- **Each iteration** is carried out in **one cycle**



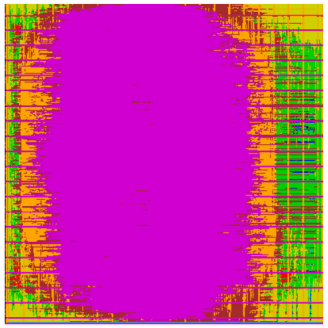
Fully Parallel Implementation

Isomorphic architecture: direct mapping of Tanner graph onto silicon

- Instantiate **2048 VNs** and **384 check nodes**
- **Edges** are implemented through a global routing network
- **Each iteration** is carried out in **one cycle**

Straightforward reference implementation in 65nm CMOS illustrates the main implementation issue

- Throughput: 1.7 Gbps
- Silicon area: 18.2 mm^2
- **Core utilization: 25%**



[Mohsenin et al., ISSCC 2008]

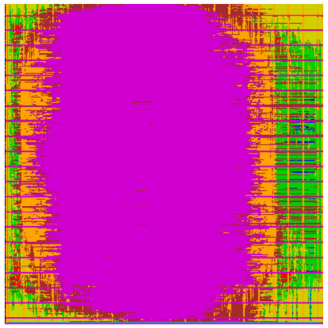
Fully Parallel Implementation

Isomorphic architecture: direct mapping of Tanner graph onto silicon

- Instantiate **2048 VNs** and **384 check nodes**
- **Edges** are implemented through a global routing network
- **Each iteration** is carried out in **one cycle**

Straightforward reference implementation in 65nm CMOS illustrates the main implementation issue

- Throughput: 1.7 Gbps
- Silicon area: 18.2 mm^2
- **Core utilization: 25%**



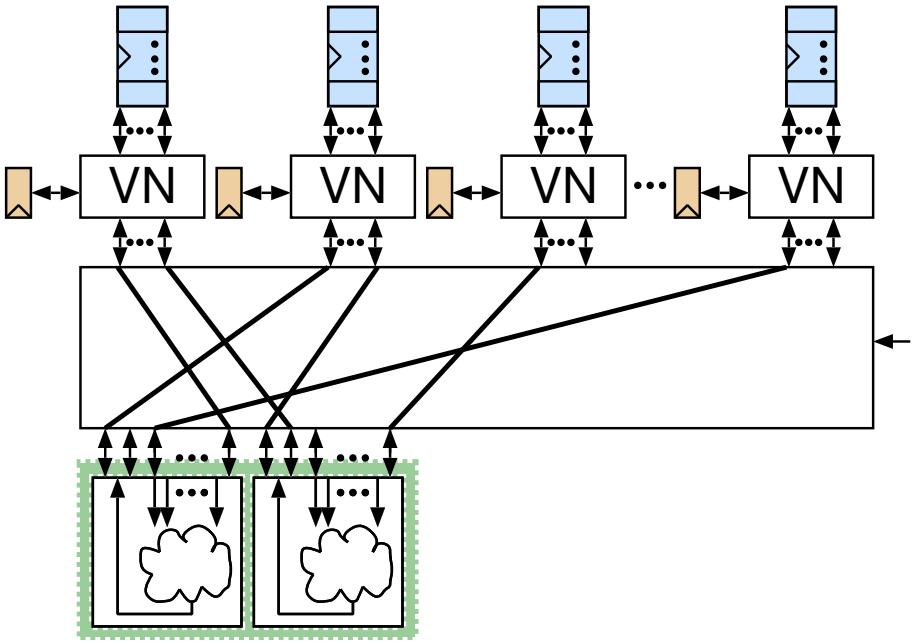
[Mohsenin et al., ISSCC 2008]

The **exchange of messages** between VNs and CNs requires **more than 100'000 global point-to-point connections**

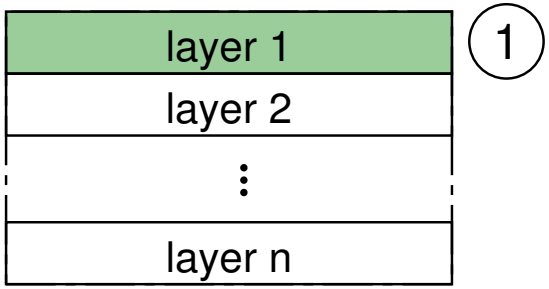
Layered Message Passing for Time Sharing

Layered decoding: Modify the schedule of VN and CN operations

- Process **one layer at a time**, but **update VNs after each layer**



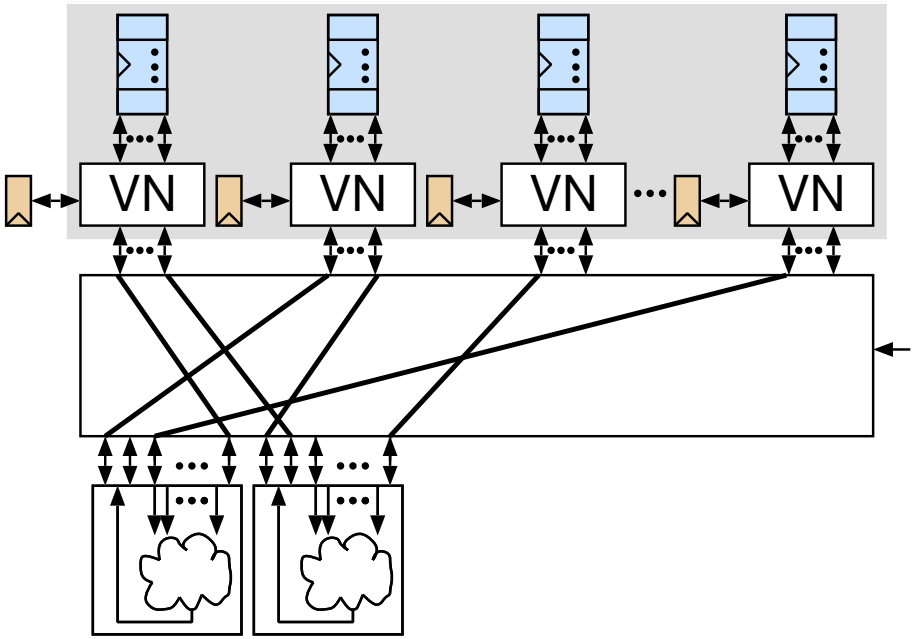
$\mathbf{H} =$



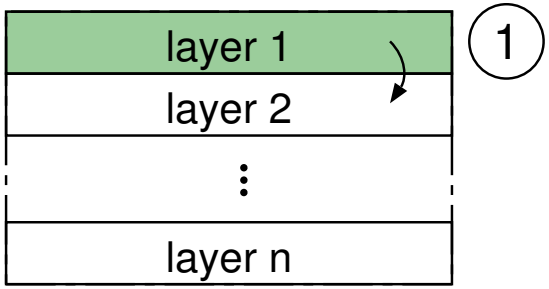
Layered Message Passing for Time Sharing

Layered decoding: Modify the schedule of VN and CN operations

- Process **one layer at a time**, but **update VNs after each layer**



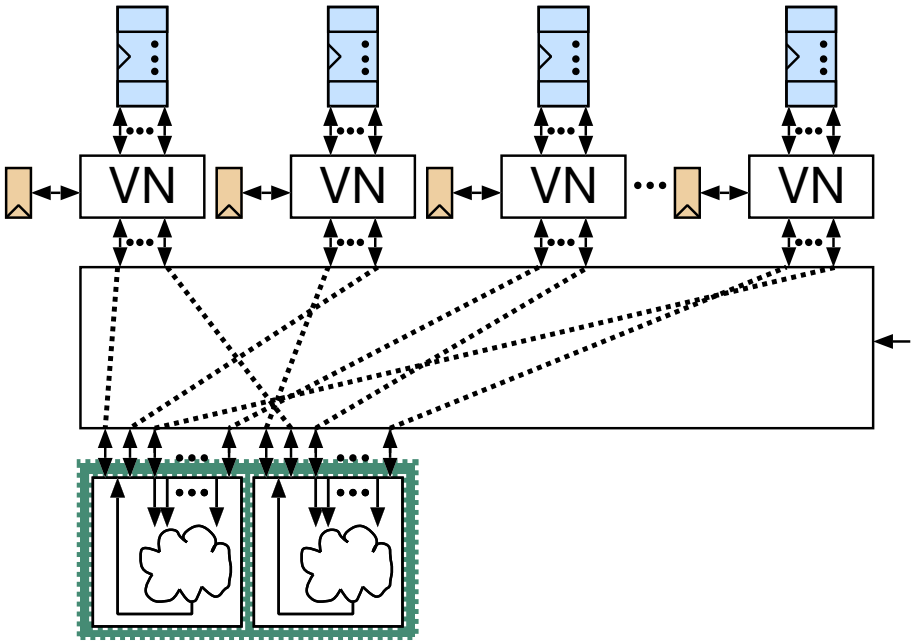
$\mathbf{H} =$



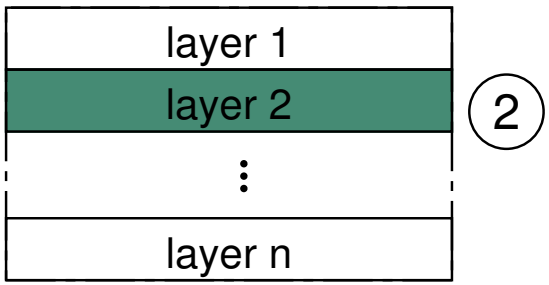
Layered Message Passing for Time Sharing

Layered decoding: Modify the schedule of VN and CN operations

- Process **one layer at a time**, but **update VNs after each layer**



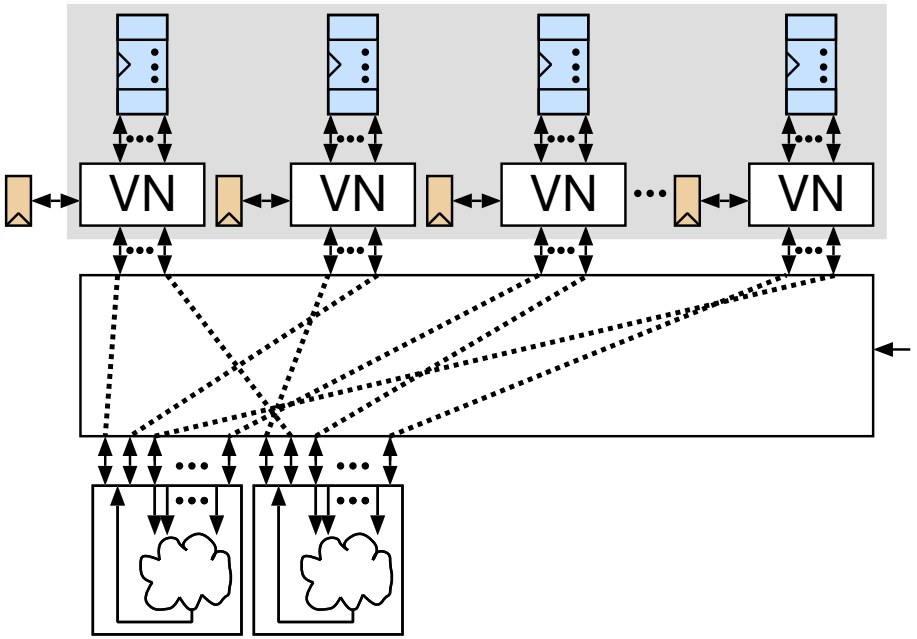
$$\mathbf{H} =$$



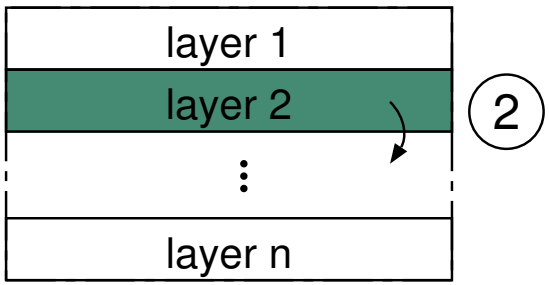
Layered Message Passing for Time Sharing

Layered decoding: Modify the schedule of VN and CN operations

- Process **one layer at a time**, but **update VNs after each layer**



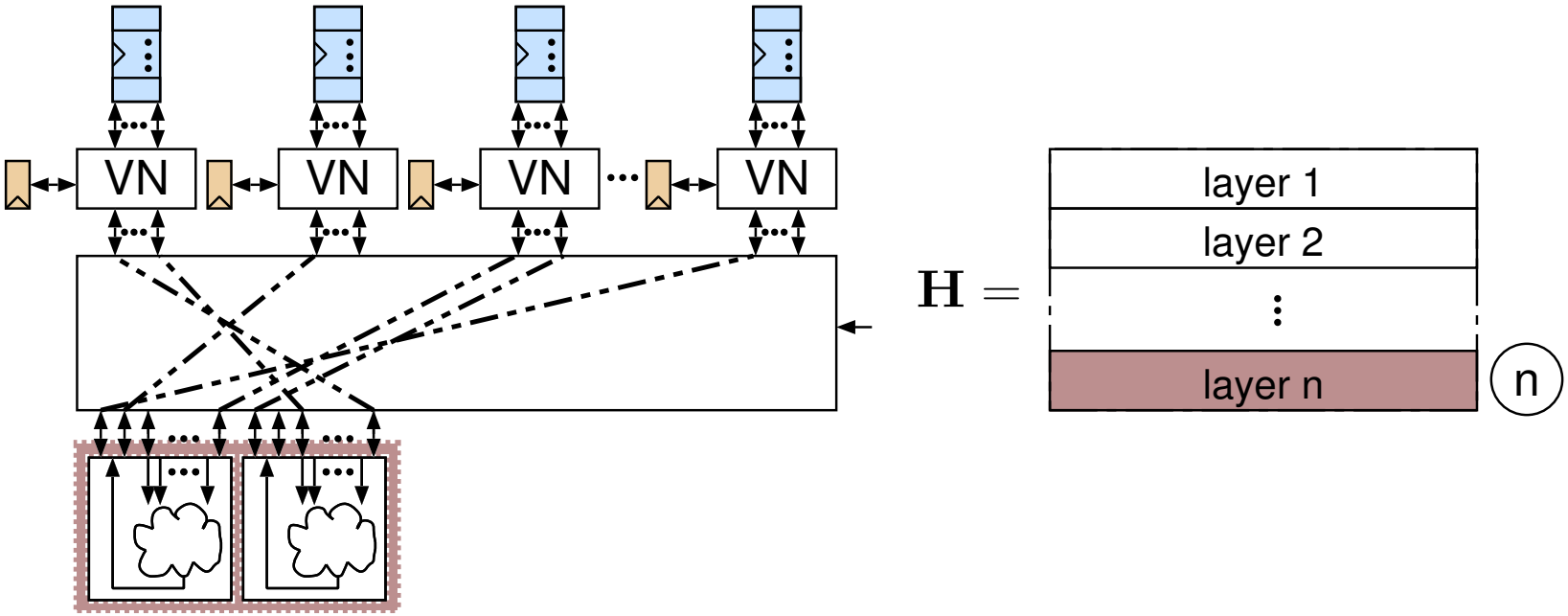
$$\mathbf{H} =$$



Layered Message Passing for Time Sharing

Layered decoding: Modify the schedule of VN and CN operations

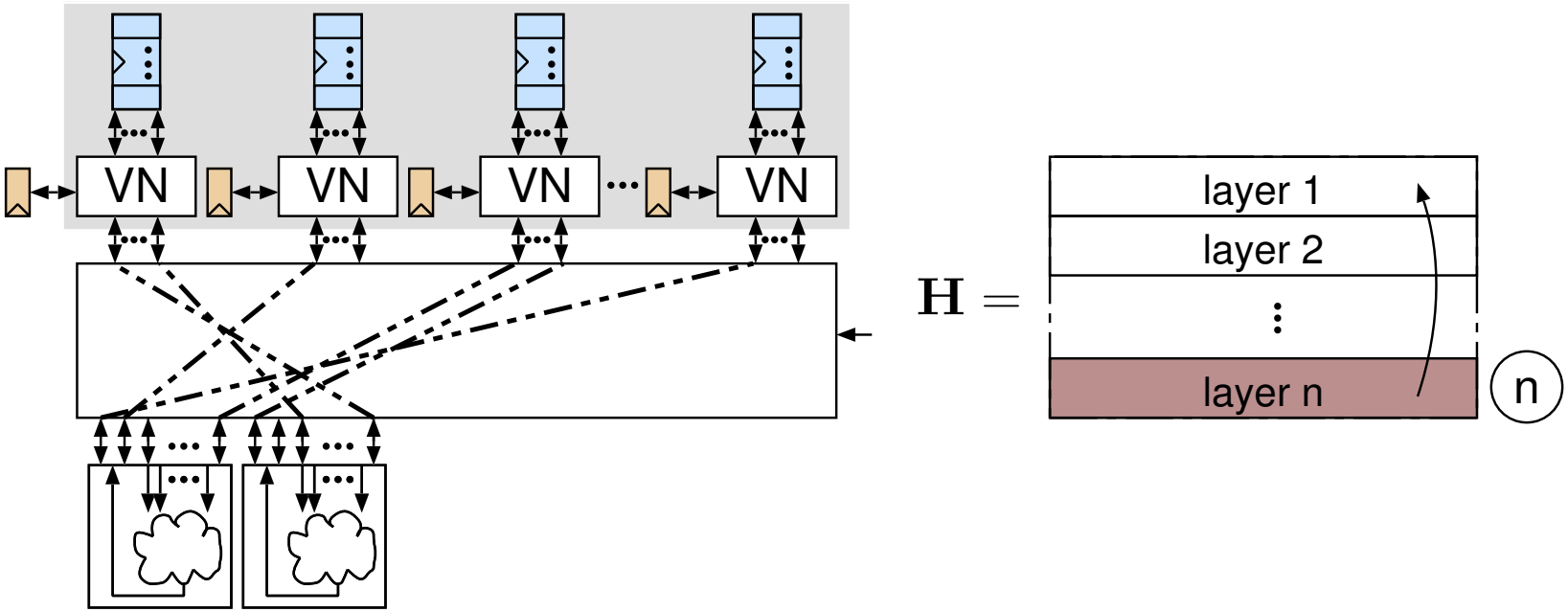
- Process **one layer at a time**, but **update VNs after each layer**



Layered Message Passing for Time Sharing

Layered decoding: Modify the schedule of VN and CN operations

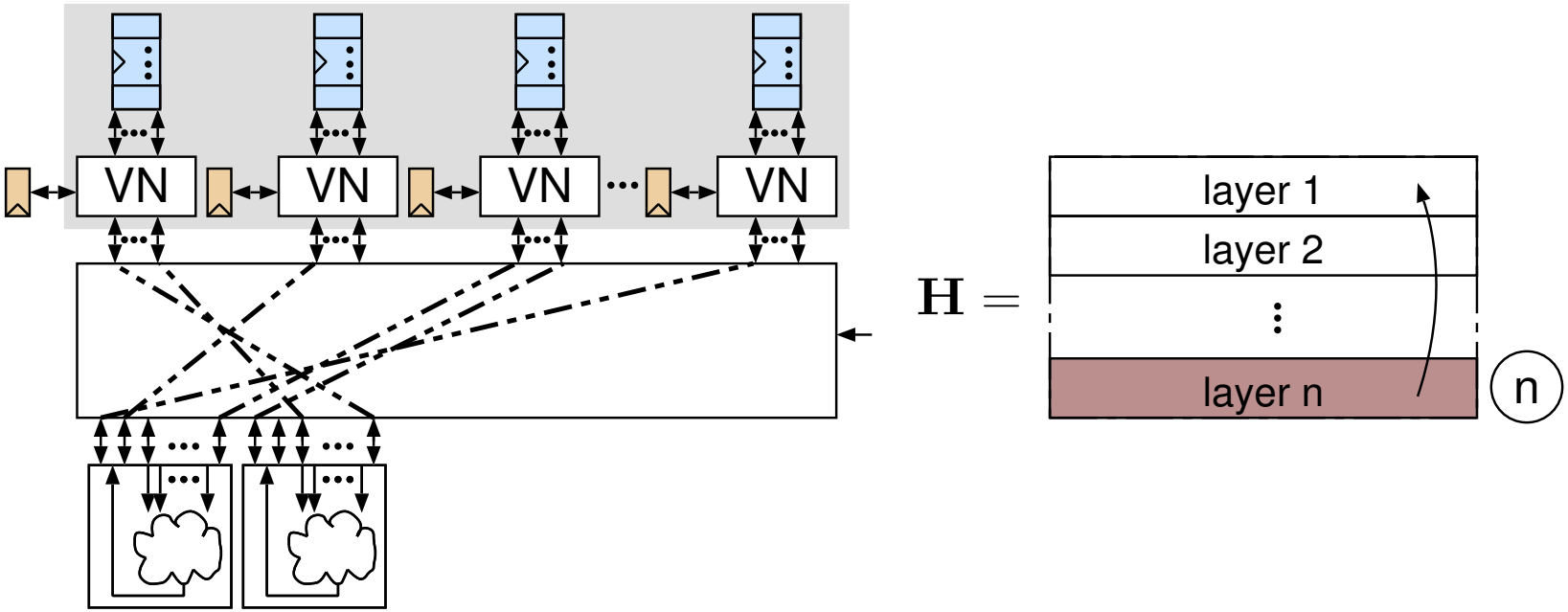
- Process **one layer at a time**, but **update VNs after each layer**



Layered Message Passing for Time Sharing

Layered decoding: Modify the schedule of VN and CN operations

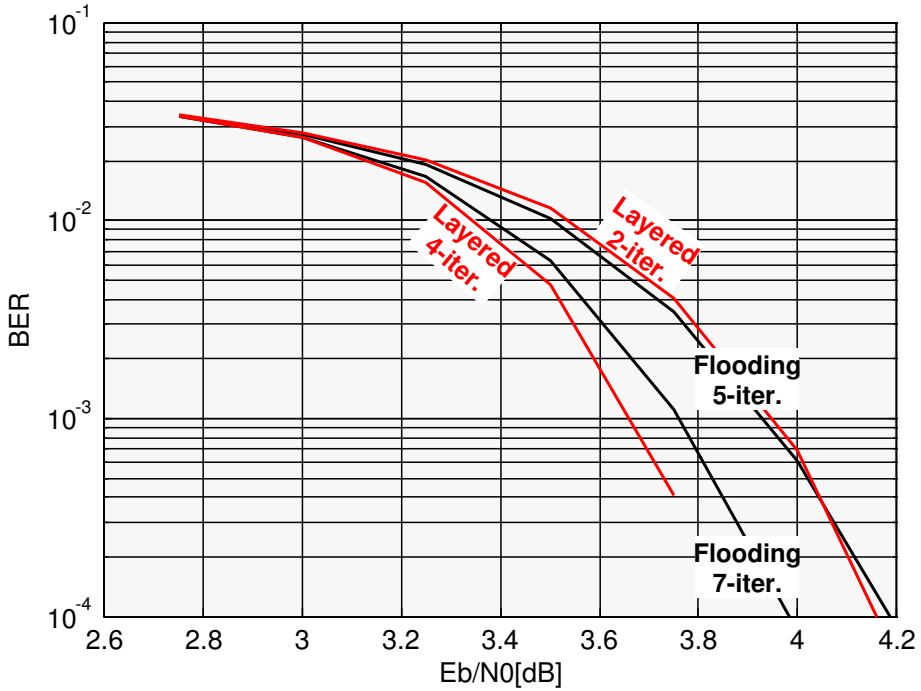
- Process **one layer at a time**, but **update VNs after each layer**



Layered decoding enables efficient time sharing of resources

Impact of Layered Decoding on Performance

Using a **layered schedule** results in a behavior that is **different from** message passing with a **flooding schedule**



BPSK, AWGN, (2048,1723) LDPC code for 10GBASE-T with OMS decoding, $\beta = 1.0$

The **layered schedule improves convergence**

↓

Reduces throughput loss from resource sharing.

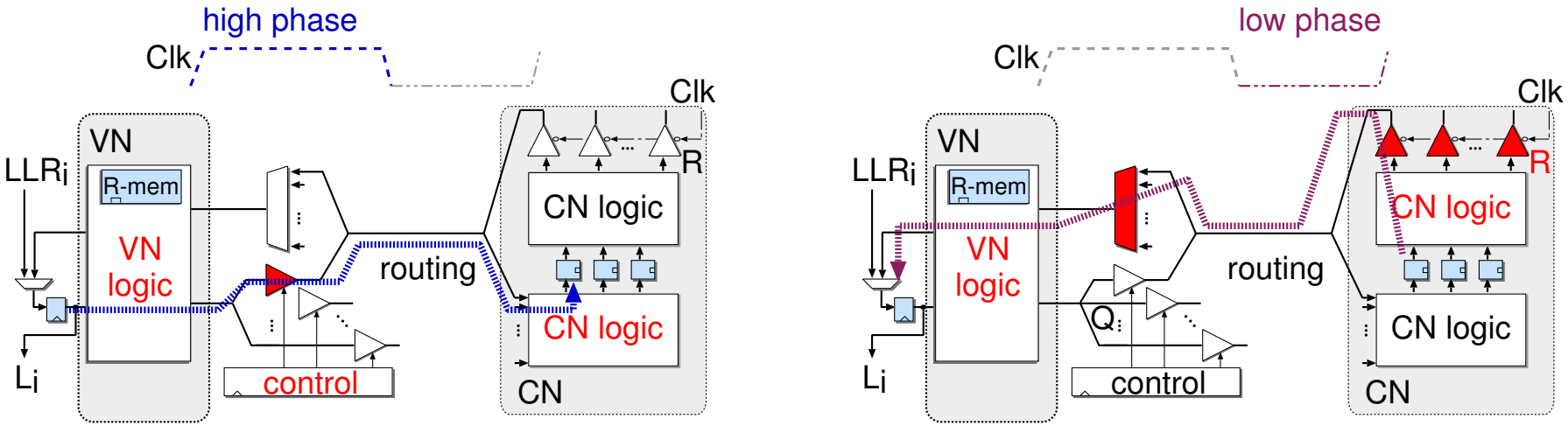
Solving the Routing Issue with Circuit Techniques

Main issue: Routing overhead is one of the main limitations (density & frequency)

Solving the Routing Issue with Circuit Techniques

Main issue: Routing overhead is one of the main limitations (density & frequency)

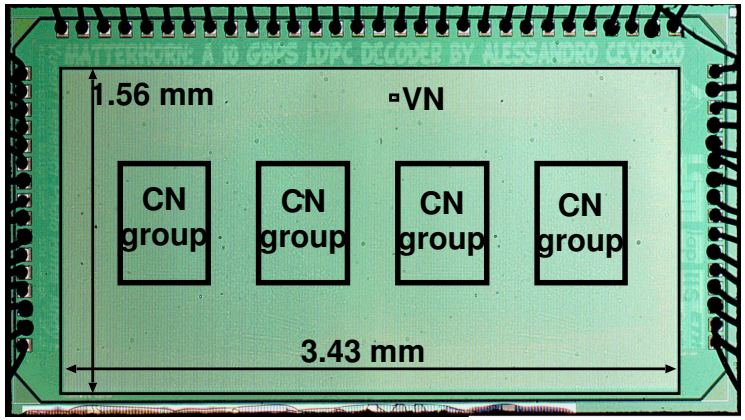
Solution: Time share routing wired for VN→CN and CN→VN routing



Full-duplex routing

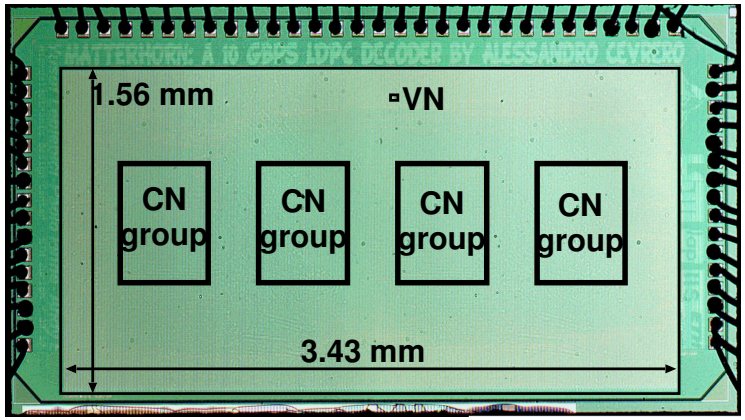
Enables 50% less routing wires and enables 84% area utilization

10 Gbps LDPC Implementation Results



Design	this work	Zhang JSSC'10	Mohsenin TCAS'10	Liu TCAS'08
Tech. [nm]	90	65	65	90
Algorithm	OMS	OMS	split-row-16	SPA
Scheduling	layered	flooding	flooding	flooding
Area [mm^2]	5.35	5.35	3.8	14.5
Speed [Gb/s]	**11.69**	**13.3**	**13.8**	5.30
Energy [pJ/bit]	133.37	210.52		

10 Gbps LDPC Implementation Results



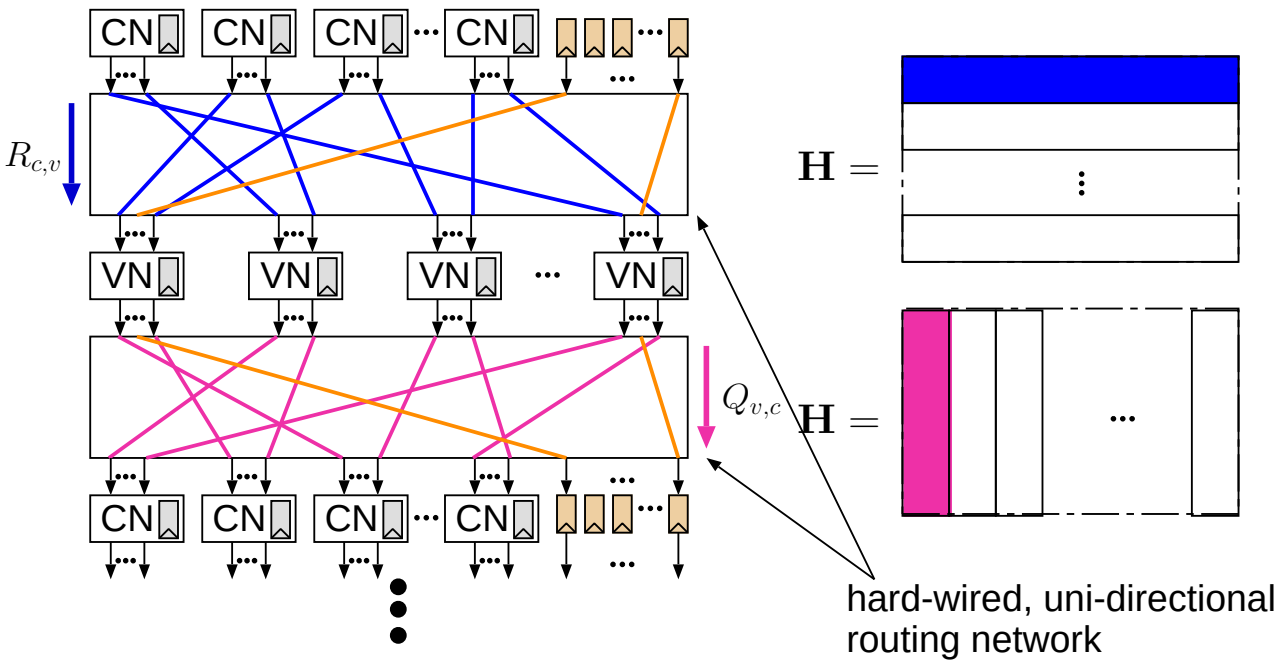
Design	this work	Zhang JSSC'10	Mohsenin TCAS'10	Liu TCAS'08
Tech. [nm]	90	65	65	90
Algorithm	OMS	OMS	split-row-16	SPA
Scheduling	layered	flooding	flooding	flooding
Area [mm^2]	5.35	5.35	3.8	14.5
Speed [Gb/s]	**11.69**	**13.3**	**13.8**	5.30
Energy [pJ/bit]	133.37	210.52		

Throughput scaling beyond 10 Gbps limited by sequential processing and routing overhead which limits frequency.

Unrolled Fully Parallel Implementation

Unrolled architecture: mapping of all decoding iterations onto silicon

- **Each iteration** is instantiation of **2048 VNs** and **384 CNs** (two stages)
- Decoder architecture consists of **distinct sets** of VN and CN stages **for each iteration**
- **Connections** are realized through routing networks between CN/VN stages
- **One decoded codeword per cycle**



Unrolled Fully Parallel Implementation

Bottlenecks:

- Very large area
- Each routing network still requires **more than 50'000** interconnects (12'288 messages of 4-5 bits) $\rightarrow$ **severe routing congestion**

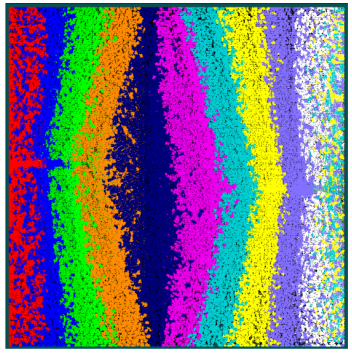
Unrolled Fully Parallel Implementation

Bottlenecks:

- Very large area
- Each routing network still requires **more than 50'000** interconnects (12'288 messages of 4-5 bits) $\rightarrow$ **severe routing congestion**

The only available implementation in 65nm CMOS:

- Throughput: 160 Gbps
- Silicon area: $13.6\,\text{mm}^2$
- **Code specification:** $N = 672$, $d_C = 6$, $d_V = 3$



[Schlafer et al., SiPS 2013]

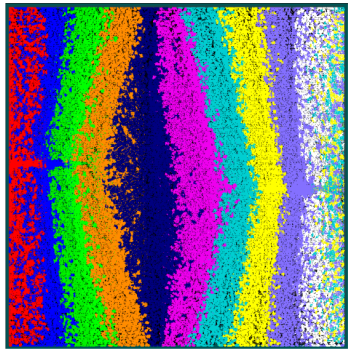
Unrolled Fully Parallel Implementation

Bottlenecks:

- Very large area
- Each routing network still requires **more than 50'000** interconnects (12'288 messages of 4-5 bits) $\rightarrow$ **severe routing congestion**

The only available implementation in 65nm CMOS:

- Throughput: 160 Gbps
- Silicon area: $13.6\,\text{mm}^2$
- **Code specification:** $N = 672$, $d_C = 6$, $d_V = 3$



[Schlafer et al., SiPS 2013]

Implementation for longer codes with larger CN/VN degree is **NOT trivial**.

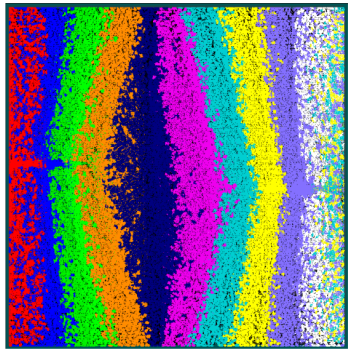
Unrolled Fully Parallel Implementation

Bottlenecks:

- Very large area
- Each routing network still requires **more than 50'000** interconnects (12'288 messages of 4-5 bits) $\rightarrow$ **severe routing congestion**

The only available implementation in 65nm CMOS:

- Throughput: 160 Gbps
- Silicon area: $13.6\,\text{mm}^2$
- **Code specification:** $N = 672$, $d_C = 6$, $d_V = 3$



[Schlafer et al., SiPS 2013]

Implementation for longer codes with larger CN/VN degree is **NOT trivial**.

More complex codes require further reduction of the routing congestion.

Unrolled Architecture: Layout Considerations

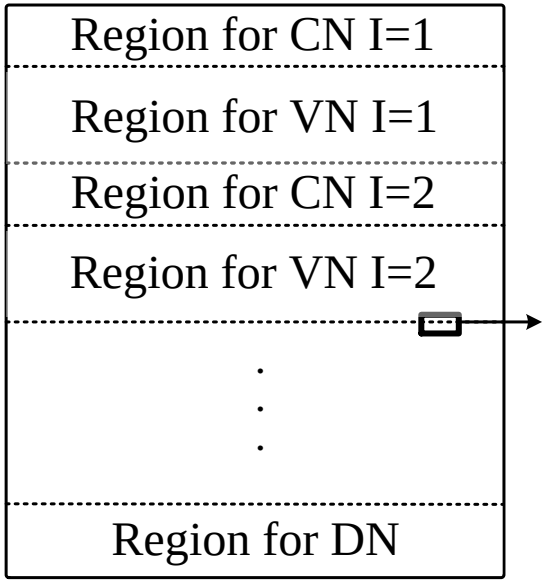
Flat layout beyond the capabilities of automatic P&R tools.

Structured hierarchical layout required for acceptable results.

Unrolled Architecture: Layout Considerations

Flat layout beyond the capabilities of automatic P&R tools.

Structured hierarchical layout required for acceptable results.

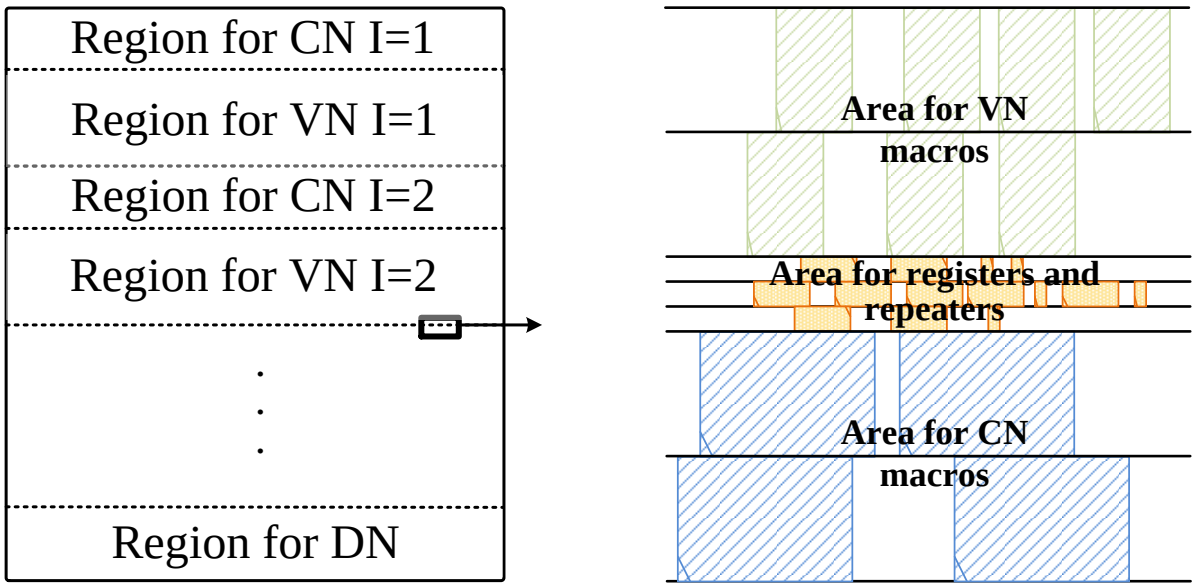


- Structured floorplan based on message-passing data flow

Unrolled Architecture: Layout Considerations

Flat layout beyond the capabilities of automatic P&R tools.

Structured hierarchical layout required for acceptable results.

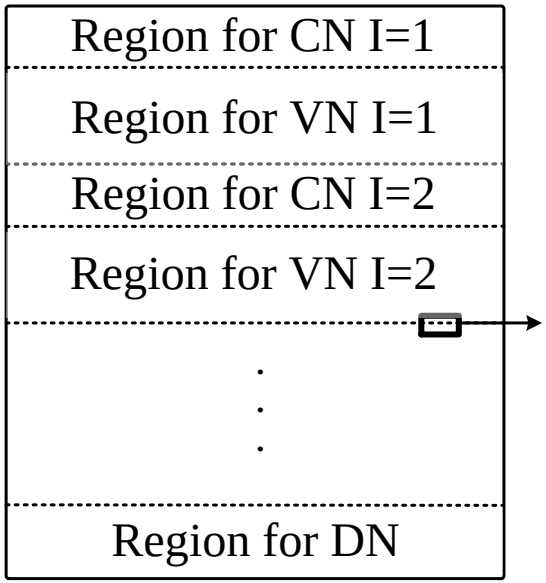


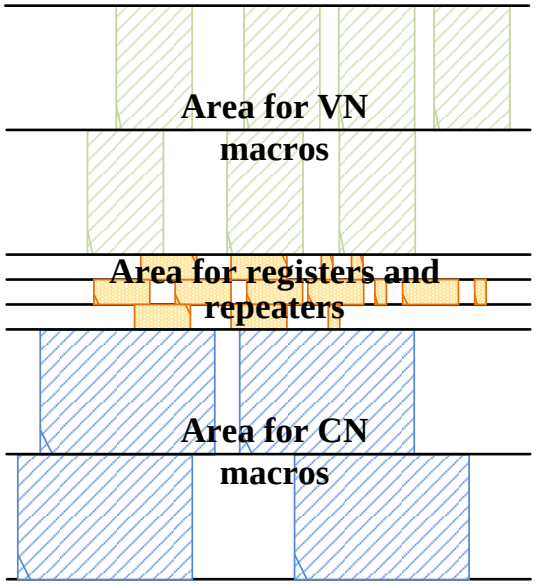
- Structured floorplan based on message-passing data flow
- Package VN/CN macros as standard-cells → allows using the more capable standard-cell (instead of macro) APR tool for P&R

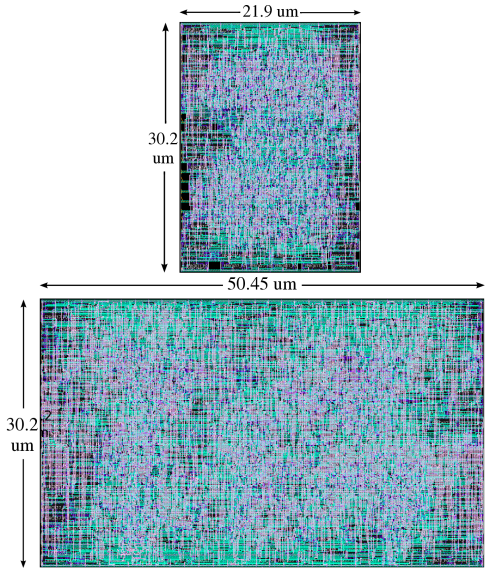
Unrolled Architecture: Layout Considerations

Flat layout beyond the capabilities of automatic P&R tools.

Structured hierarchical layout required for acceptable results.





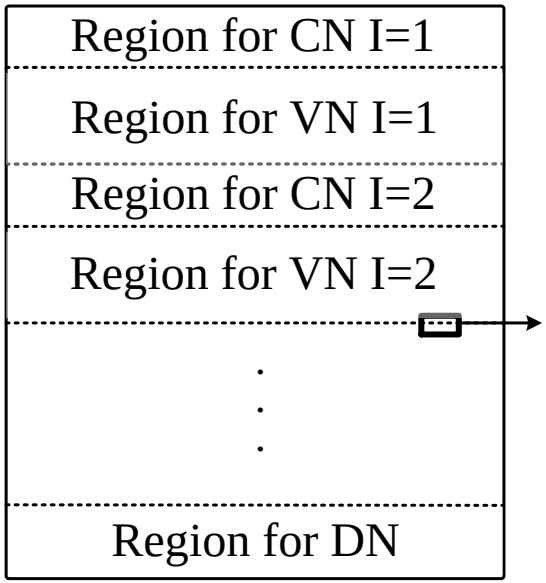


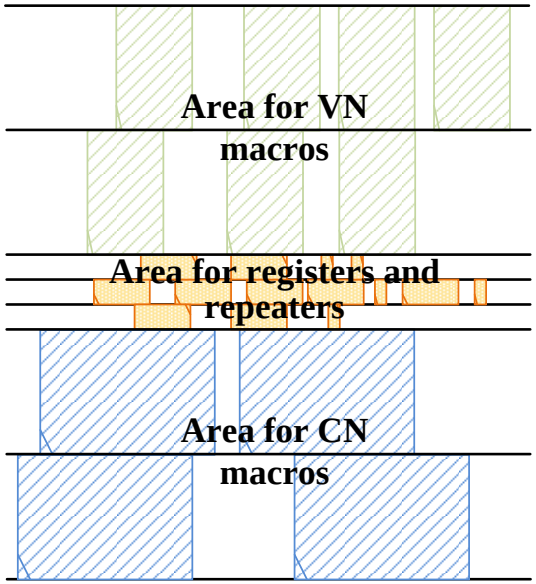
- Structured floorplan based on message-passing data flow
- Package VN/CN macros as standard-cells → allows using the more capable standard-cell (instead of macro) APR tool for P&R
- Optimize VN/CN macro size & pins and limit in-cell routing to 3 layers

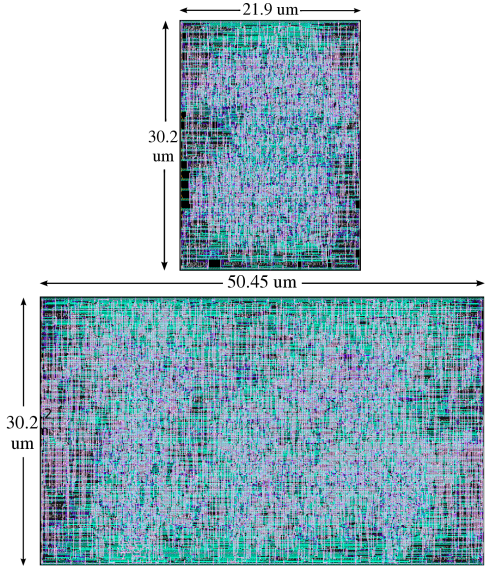
Unrolled Architecture: Layout Considerations

Flat layout beyond the capabilities of automatic P&R tools.

Structured hierarchical layout required for acceptable results.







- Structured floorplan based on message-passing data flow
- Package VN/CN macros as standard-cells → allows using the more capable standard-cell (instead of macro) APR tool for P&R
- Optimize VN/CN macro size & pins and limit in-cell routing to 3 layers

Even with structured layout, **routing overhead remains prohibitive**

Serial Message-Transfer Architecture

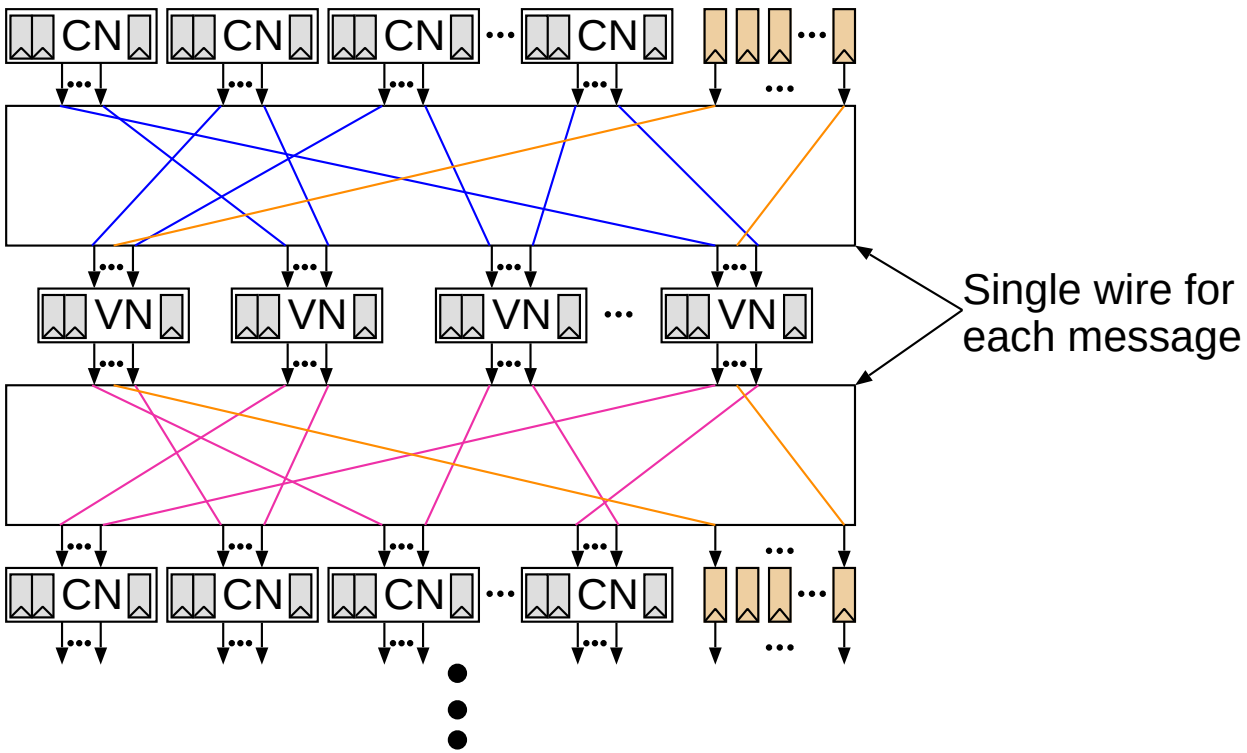
Main idea: Send/receive messages serially.

- Transfer each message bit-by-bit through a single wire
- Overlap (pipeline) message transfer with processing
- Process two codewords interleaved
- **Two clocks**: **slow clock** for logic, **fast clock** for bit-transfer (routing)

Serial Message-Transfer Architecture

Main idea: Send/receive messages serially.

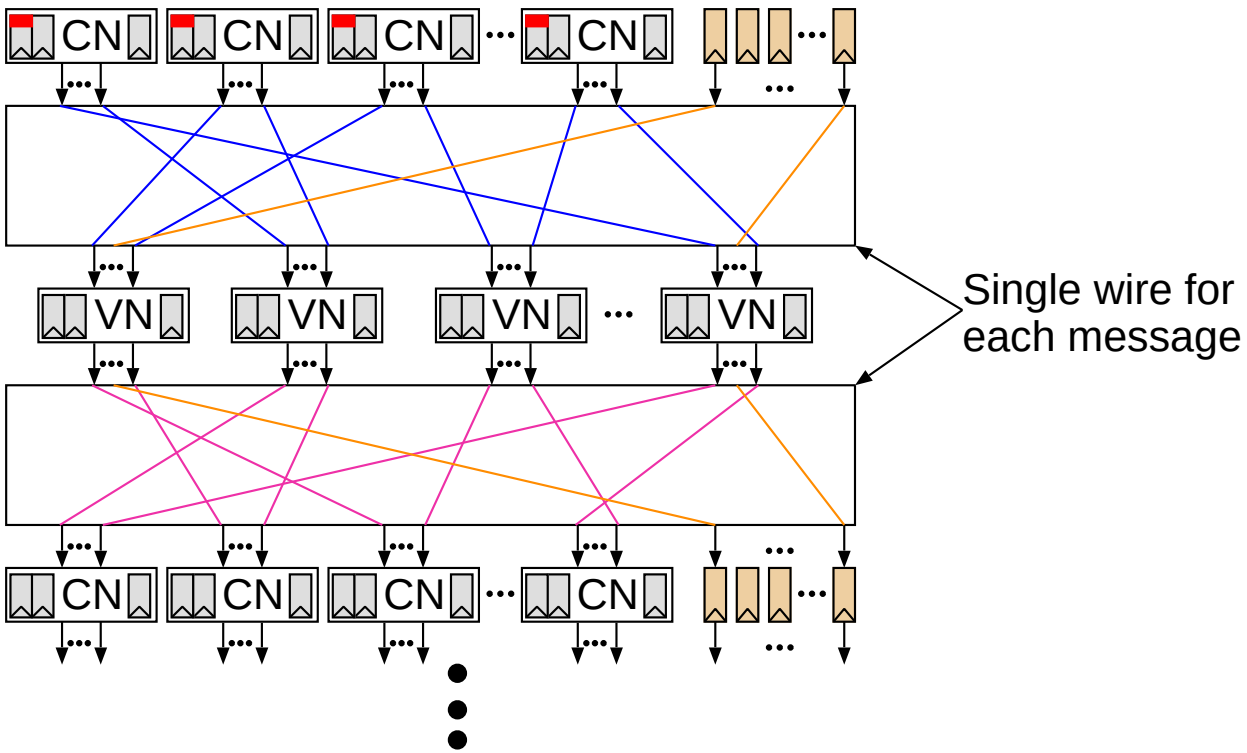
- Transfer each message bit-by-bit through a single wire
- Overlap (pipeline) message transfer with processing
- Process two codewords interleaved
- **Two clocks**: **slow clock** for logic, **fast clock** for bit-transfer (routing)



Serial Message-Transfer Architecture

Main idea: Send/receive messages serially.

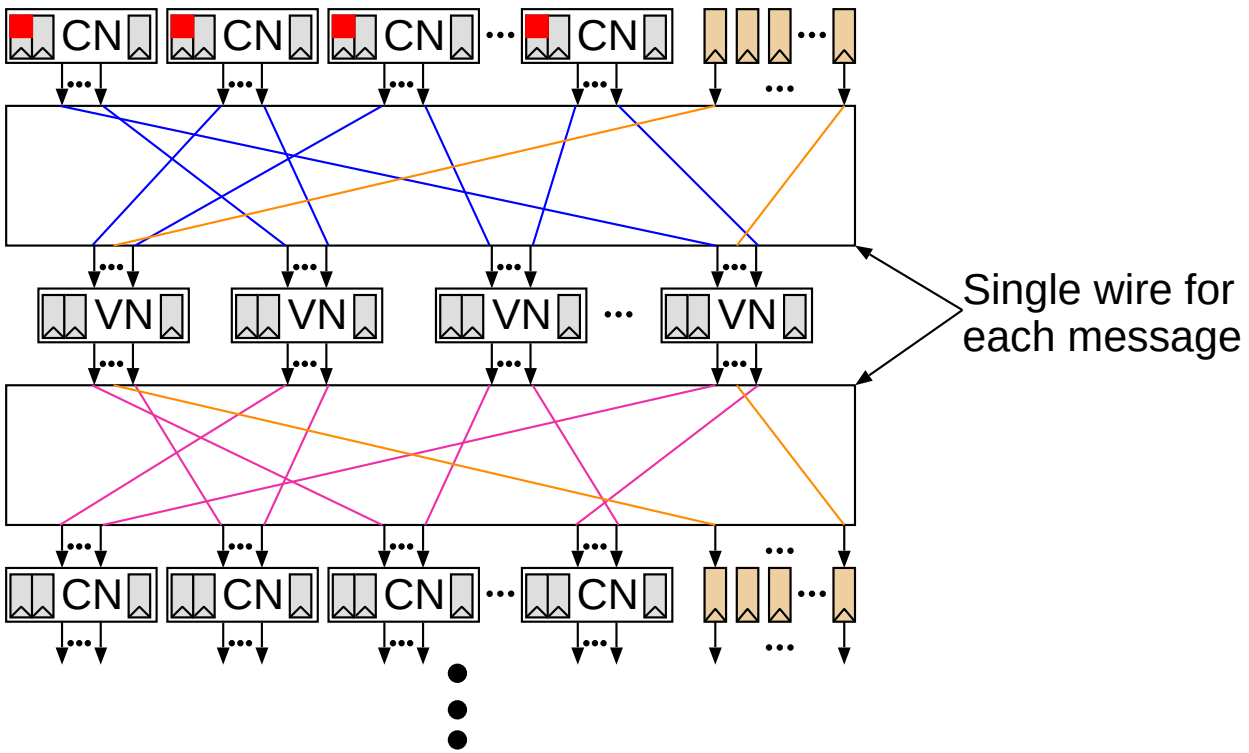
- Transfer each message bit-by-bit through a single wire
- Overlap (pipeline) message transfer with processing
- Process two codewords interleaved
- **Two clocks**: **slow clock** for logic, **fast clock** for bit-transfer (routing)



Serial Message-Transfer Architecture

Main idea: Send/receive messages serially.

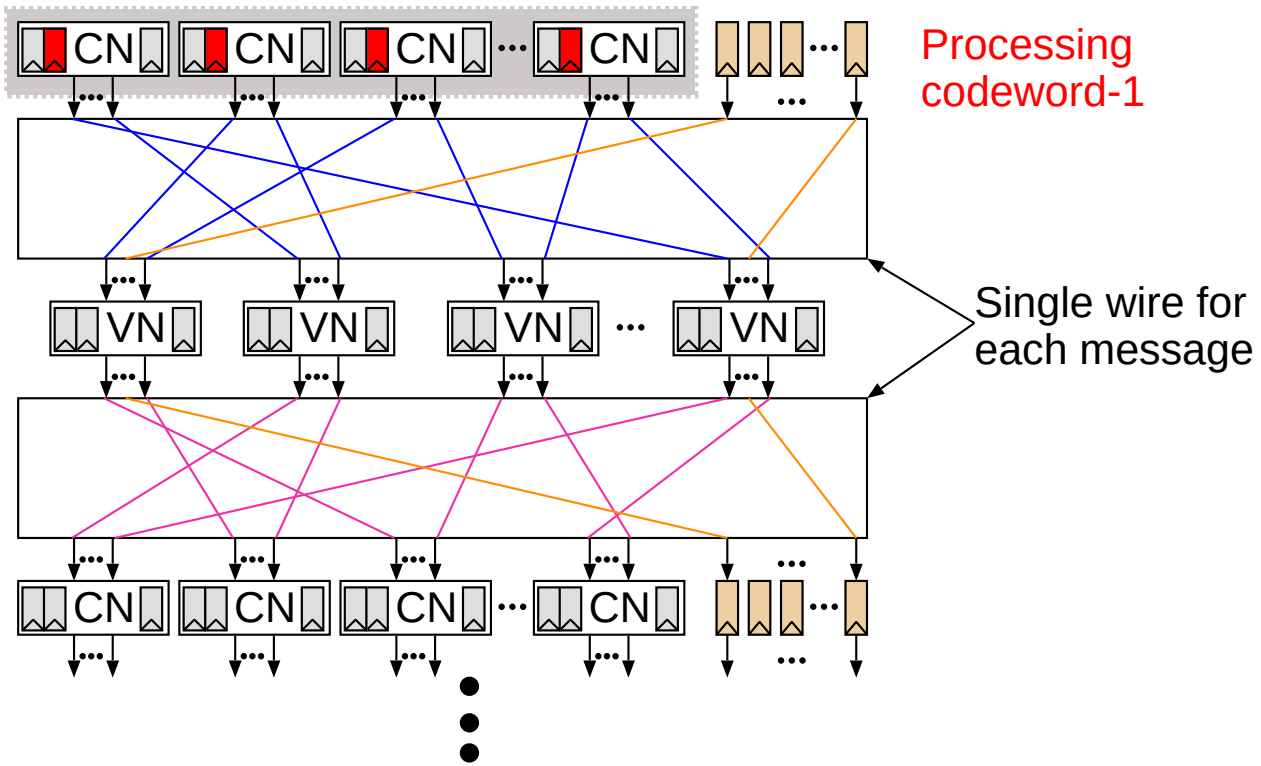
- Transfer each message bit-by-bit through a single wire
- Overlap (pipeline) message transfer with processing
- Process two codewords interleaved
- **Two clocks**: **slow clock** for logic, **fast clock** for bit-transfer (routing)



Serial Message-Transfer Architecture

Main idea: Send/receive messages serially.

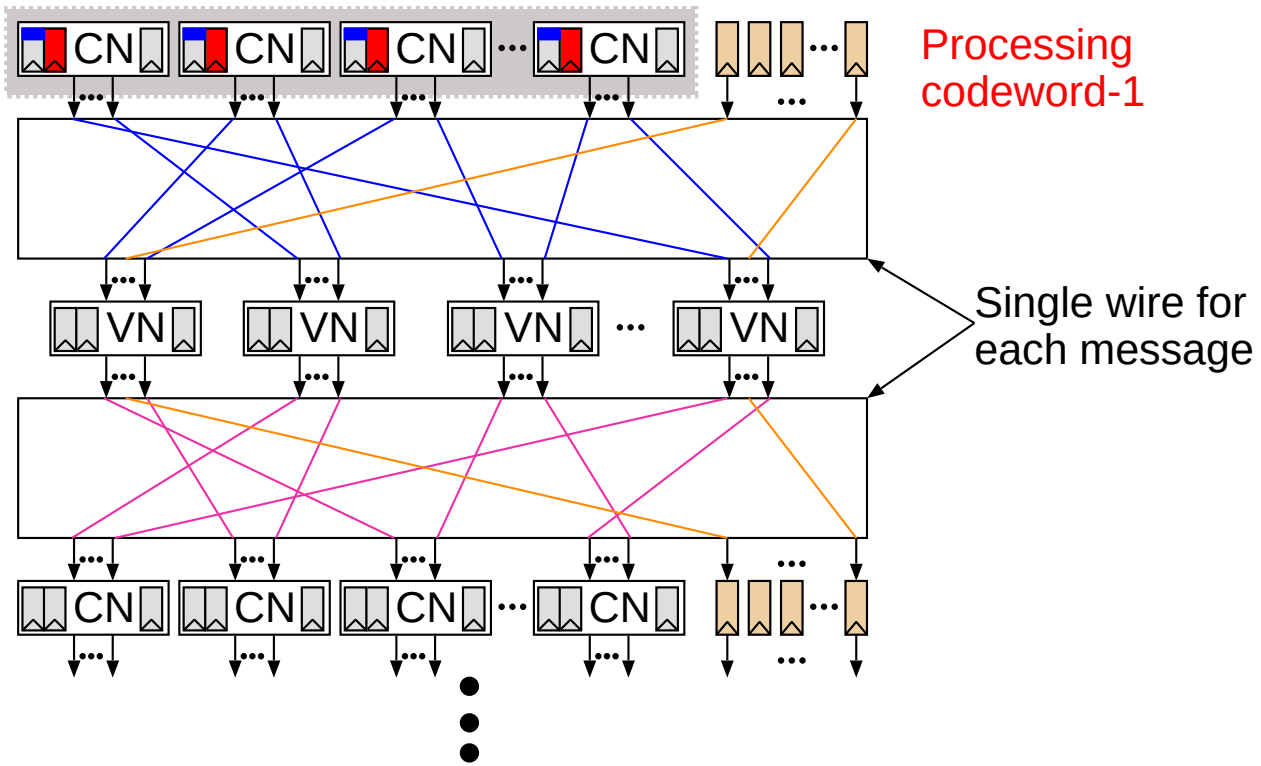
- Transfer each message bit-by-bit through a single wire
- Overlap (pipeline) message transfer with processing
- Process two codewords interleaved
- **Two clocks**: **slow clock** for logic, **fast clock** for bit-transfer (routing)



Serial Message-Transfer Architecture

Main idea: Send/receive messages serially.

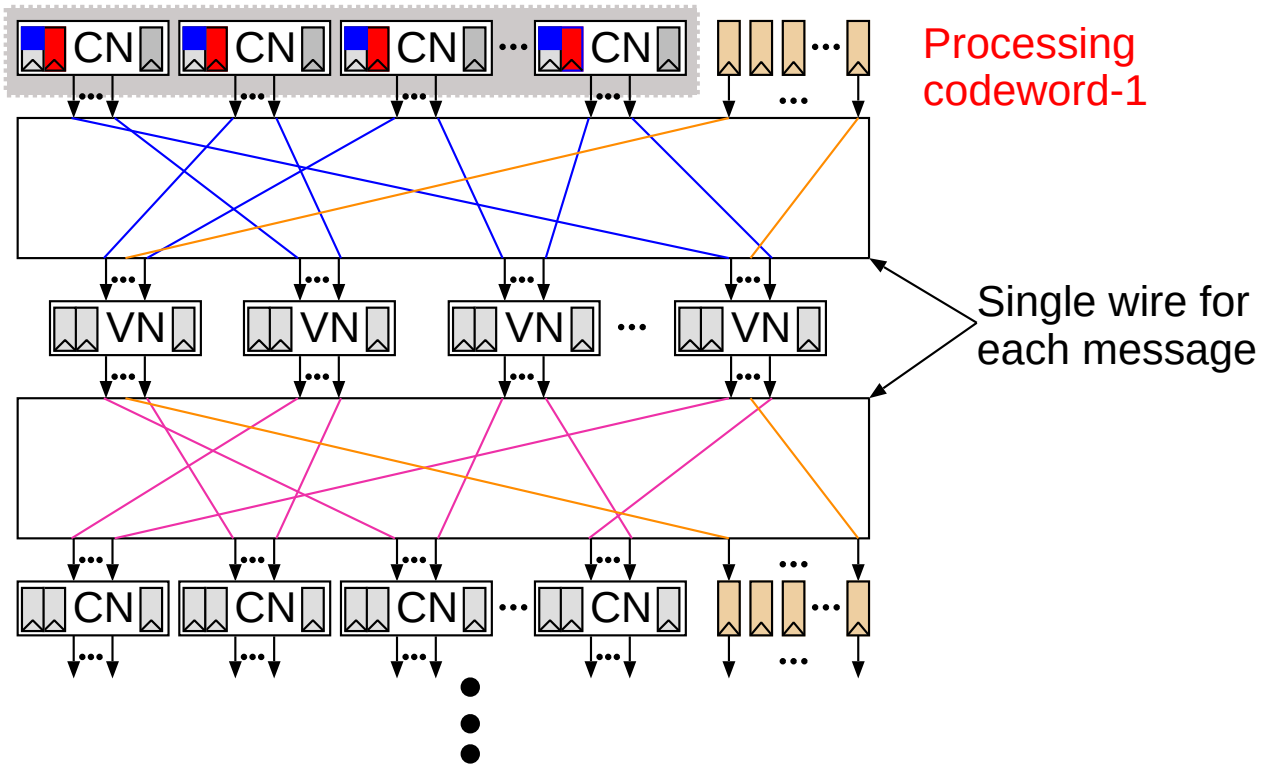
- Transfer each message bit-by-bit through a single wire
- Overlap (pipeline) message transfer with processing
- Process two codewords interleaved
- **Two clocks**: **slow clock** for logic, **fast clock** for bit-transfer (routing)



Serial Message-Transfer Architecture

Main idea: Send/receive messages serially.

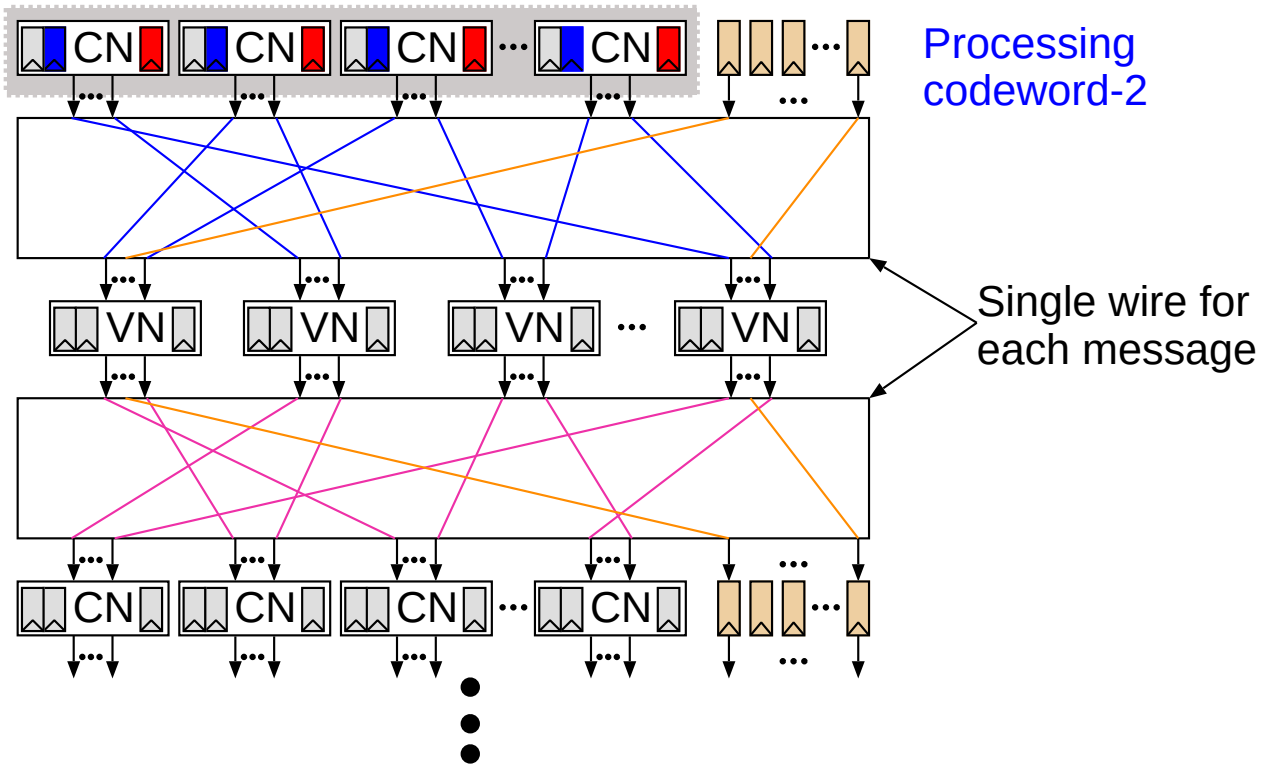
- Transfer each message bit-by-bit through a single wire
- Overlap (pipeline) message transfer with processing
- Process two codewords interleaved
- **Two clocks**: **slow clock** for logic, **fast clock** for bit-transfer (routing)



Serial Message-Transfer Architecture

Main idea: Send/receive messages serially.

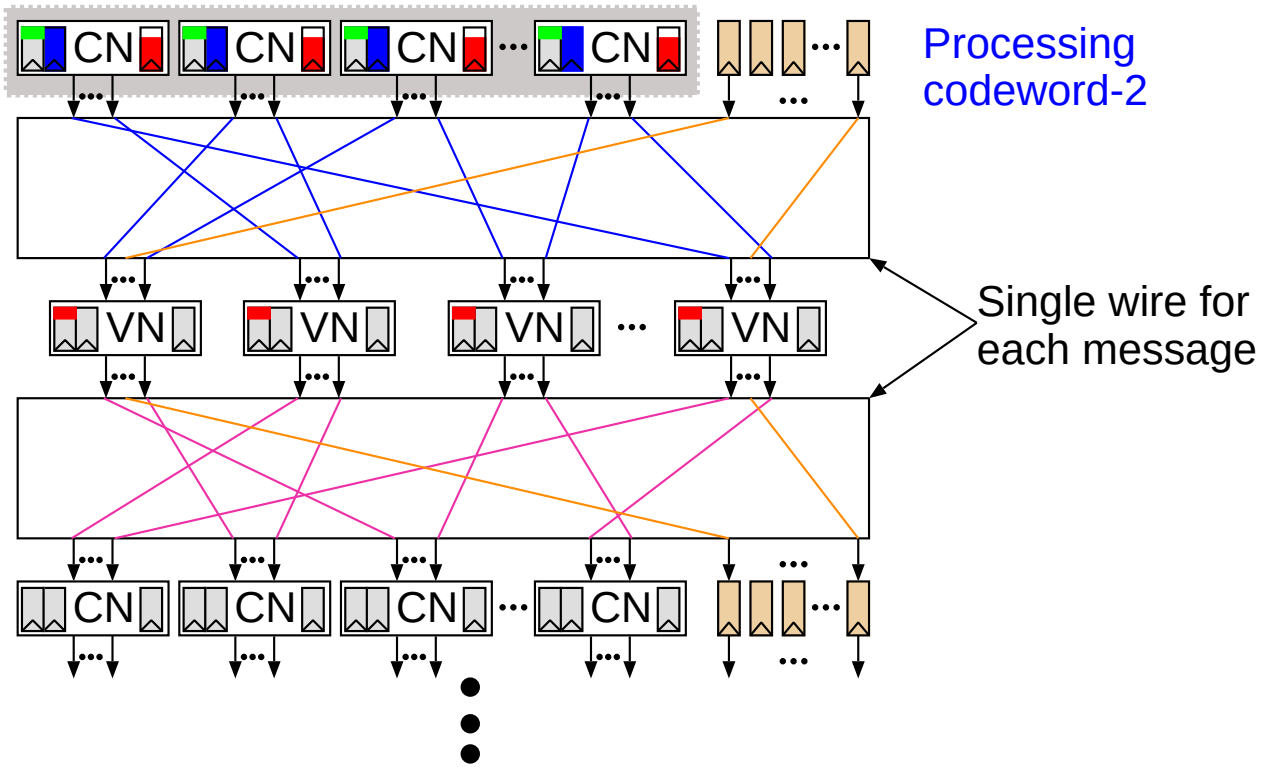
- Transfer each message bit-by-bit through a single wire
- Overlap (pipeline) message transfer with processing
- Process two codewords interleaved
- **Two clocks**: **slow clock** for logic, **fast clock** for bit-transfer (routing)



Serial Message-Transfer Architecture

Main idea: Send/receive messages serially.

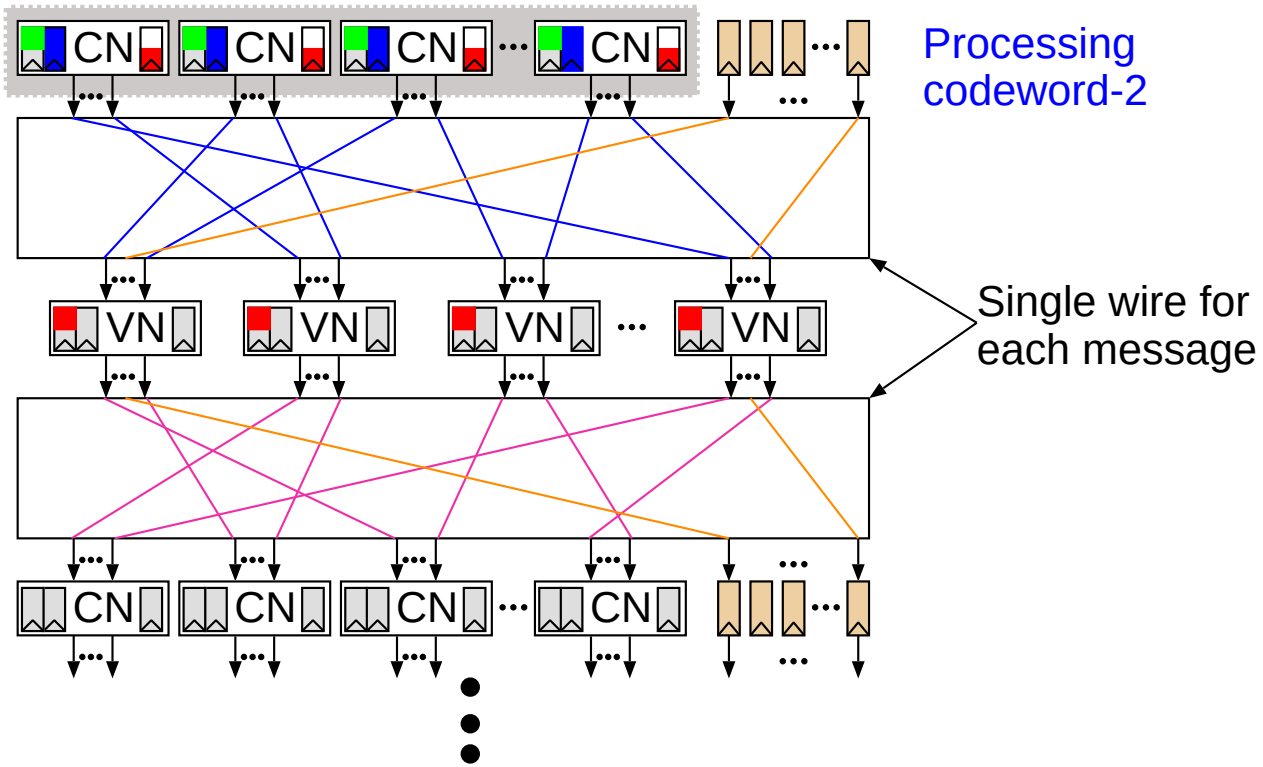
- Transfer each message bit-by-bit through a single wire
- Overlap (pipeline) message transfer with processing
- Process two codewords interleaved
- **Two clocks**: **slow clock** for logic, **fast clock** for bit-transfer (routing)



Serial Message-Transfer Architecture

Main idea: Send/receive messages serially.

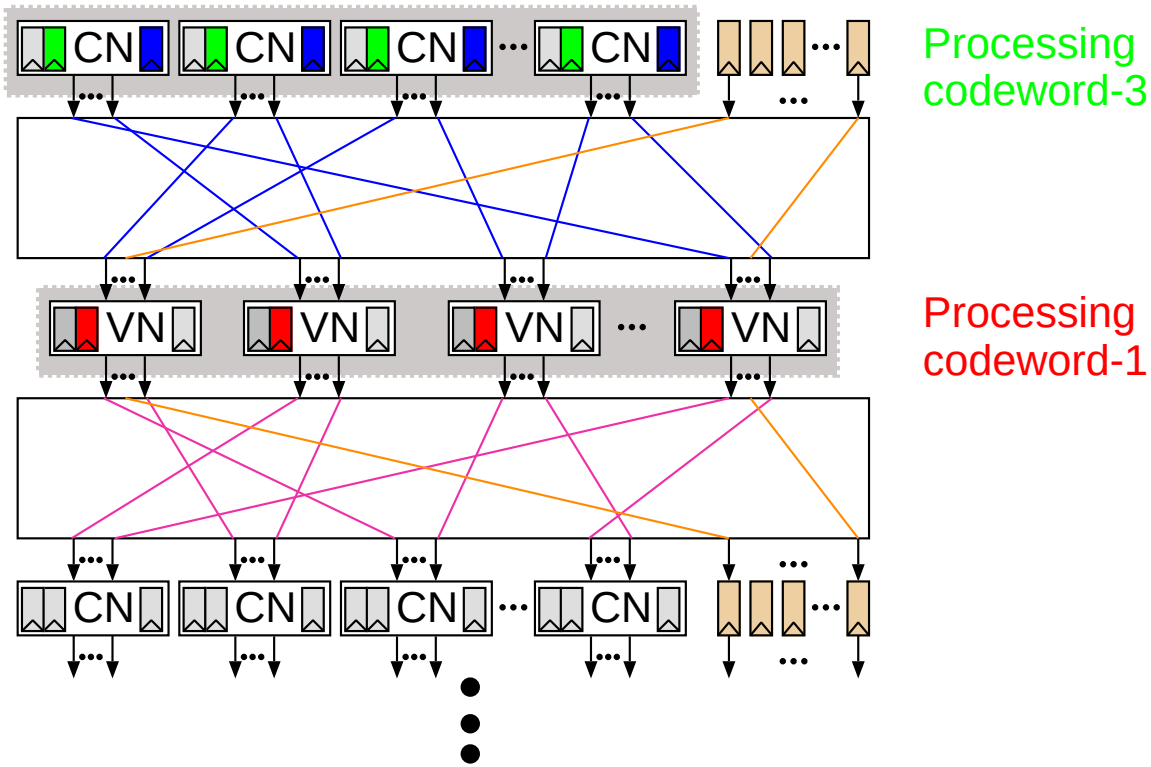
- Transfer each message bit-by-bit through a single wire
- Overlap (pipeline) message transfer with processing
- Process two codewords interleaved
- **Two clocks**: **slow clock** for logic, **fast clock** for bit-transfer (routing)



Serial Message-Transfer Architecture

Main idea: Send/receive messages serially.

- Transfer each message bit-by-bit through a single wire
- Overlap (pipeline) message transfer with processing
- Process two codewords interleaved
- **Two clocks**: **slow clock** for logic, **fast clock** for bit-transfer (routing)



Serial Message-Transfer: Pros & Cons

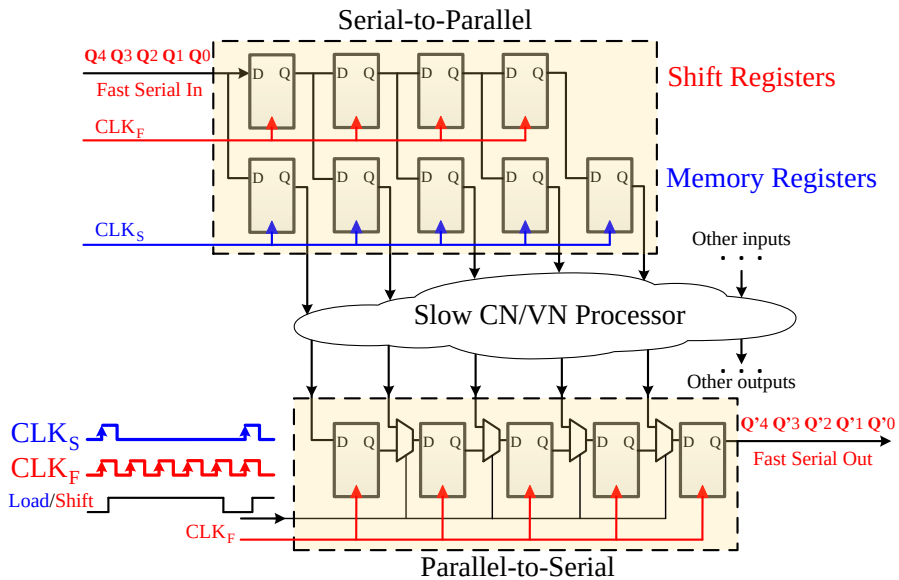
Advantages:

- **Routing congestion is significantly reduced**
- **Overlapped processing** of two codewords **hides message transfer** delay
- Signal **routing in a separate pipeline stage** (no increase in logic delay)

Disadvantages:

- Number of registers increases by 3x
- Decoder latency increases by 2x
- Minimum clock period limited by

$$T_{clk} > \min\left(T_{clk_{logic}}, Q_{msg} \times T_{clk_{routing}}\right)$$

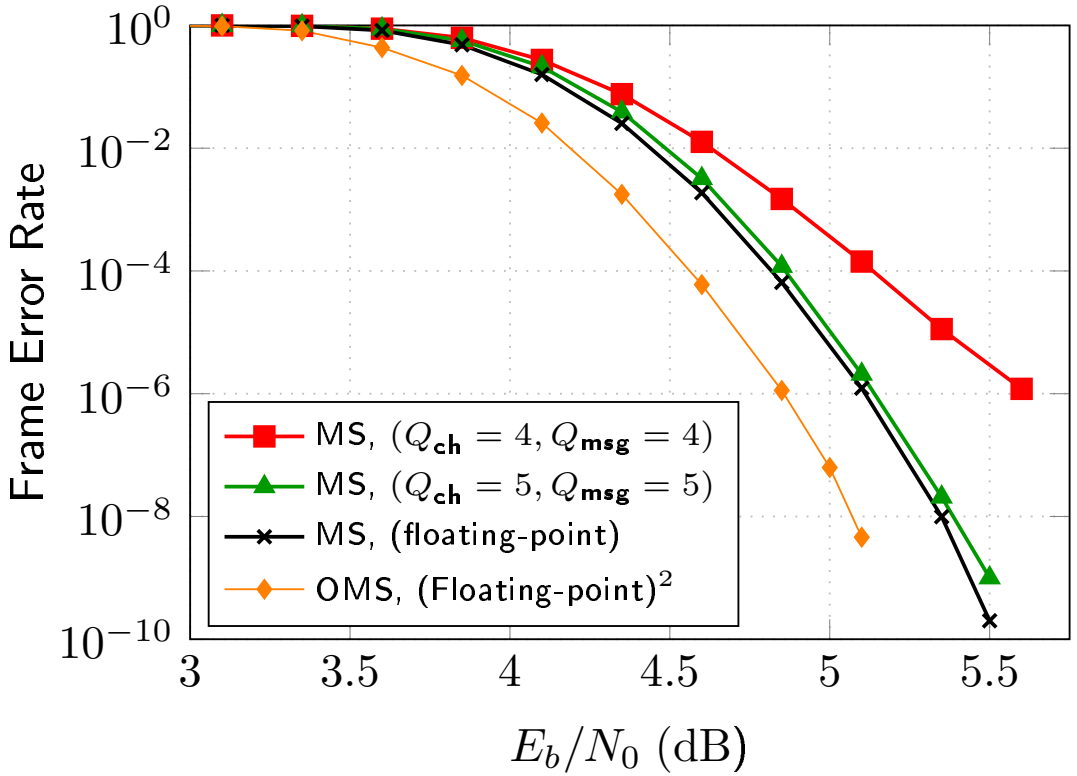


Routing delay must be significantly shorter than logic delay.

Min-Sum Message Quantization

Message quantization (wordlength) Q_{msg} has a critical impact on

- Complexity (area and delay) of VNs and CNs
- Message-routing overhead between stages
- Decoder performance (FER)

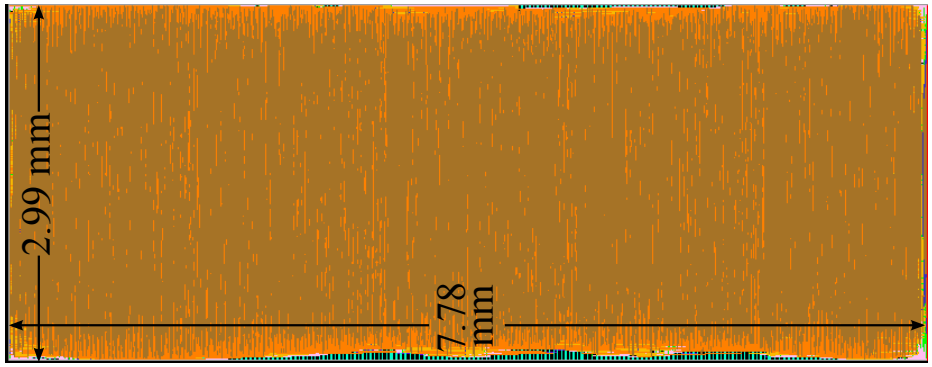


Message quantization with $Q_{msg} \geq 5$ bit required.

Unrolled Serial Message-Transfer Results

Example: Quantization of messages with $Q_{msg} = 5$ bit

- Automatic P&R is finally feasible with **66.4% layout density**

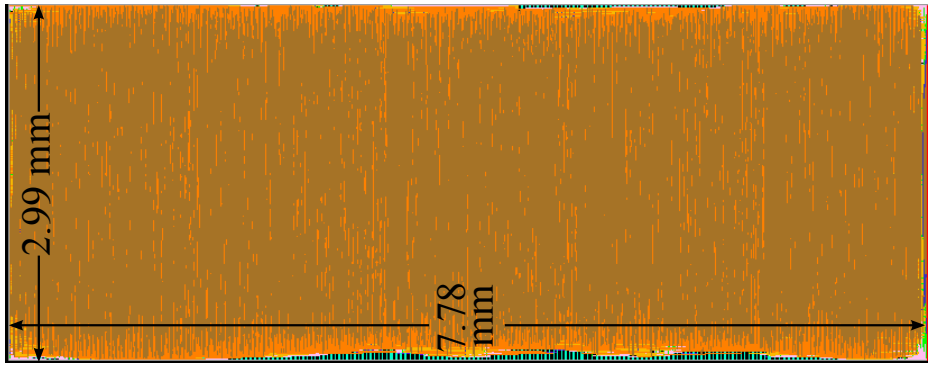


Layout in 28 nm FD-SOI technology

Unrolled Serial Message-Transfer Results

Example: Quantization of messages with $Q_{msg} = 5$ bit

- Automatic P&R is finally feasible with **66.4% layout density**



Layout in 28 nm FD-SOI technology

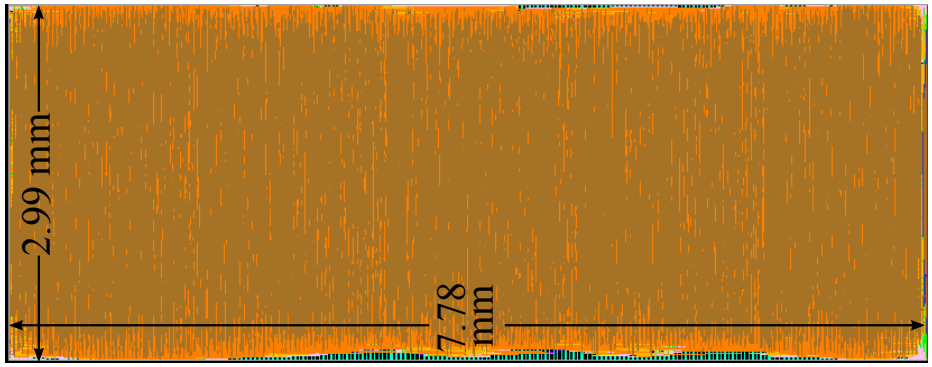
Throughput: 271 Gbps

- Required time for processing: $T_{logic} = 2.38\,ns$
- Required time for transferring one bit: $T_{routing} = 1.51\,ns$
- **Critical path**: $T_{clk} > \min\left(T_{clk_{logic}}, Q_{msg} \times T_{clk_{routing}}\right) = \mathbf{7.55\,ns}$

Unrolled Serial Message-Transfer Results

Example: Quantization of messages with $Q_{msg} = 5$ bit

- Automatic P&R is finally feasible with **66.4% layout density**



Layout in 28 nm FD-SOI technology

Throughput: 271 Gbps

- Required time for processing: $T_{logic} = 2.38\,ns$
- Required time for transferring one bit: $T_{routing} = 1.51\,ns$
- **Critical path**: $T_{clk} > \min\left(T_{clk_{logic}}, Q_{msg} \times T_{clk_{routing}}\right) = \mathbf{7.55\,ns}$

Decoding throughput is **limited by serial message transfer** of 5 bit messages

Quantized Message Passing

Motivation: Message **wordlength has significant (linear) impact on throughput** with serial message transfer and influences logic area and delay.

Quantized Message Passing

Motivation: Message **wordlength has significant (linear) impact on throughput** with serial message transfer and influences logic area and delay.

Conventional Message-Passing

Conventional Arithmetic Update Rules $\xrightarrow{\text{use}}$ Uniform Quantization

Quantized Message Passing

Motivation: Message **wordlength has significant (linear) impact on throughput** with serial message transfer and influences logic area and delay.

Conventional Message-Passing

Conventional Arithmetic Update Rules $\xrightarrow{\text{use}}$ Uniform Quantization

- **Efficient arithmetic** circuits, but **large wordlengths** for good error-correcting performance due to large dynamic range.

Quantized Message Passing

Motivation: Message **wordlength has significant (linear) impact on throughput** with serial message transfer and influences logic area and delay.

Conventional Message-Passing

Conventional Arithmetic Update Rules $\xrightarrow{\text{use}}$ Uniform Quantization

- **Efficient arithmetic** circuits, but **large wordlengths** for good error-correcting performance due to large dynamic range.

Quantized Message-Passing

Non-Uniform Quantization $\xrightarrow{\text{defines}}$ Update Rules

Quantized Message Passing

Motivation: Message **wordlength has significant (linear) impact on throughput** with serial message transfer and influences logic area and delay.

Conventional Message-Passing

Conventional Arithmetic Update Rules $\xrightarrow{\text{use}}$ Uniform Quantization

- **Efficient arithmetic** circuits, but **large wordlengths** for good error-correcting performance due to large dynamic range.

Quantized Message-Passing

Non-Uniform Quantization $\xrightarrow{\text{defines}}$ Update Rules

- Potential for **significant wordlength reduction and performance improvement**.

Quantized Message Passing

Motivation: Message **wordlength has significant (linear) impact on throughput** with serial message transfer and influences logic area and delay.

Conventional Message-Passing

Conventional Arithmetic Update Rules $\xrightarrow{\text{use}}$ Uniform Quantization

- **Efficient arithmetic** circuits, but **large wordlengths** for good error-correcting performance due to large dynamic range.

Quantized Message-Passing

Non-Uniform Quantization $\xrightarrow{\text{defines}}$ Update Rules

- Potential for **significant wordlength reduction and performance improvement**.
- Update rules must be implemented as general **look-up tables**, which can require **significant area**.

Look-Up Table Design

Numerous LUT design methods [Planjery'13, Declercq'13, Cai'14, Kurkorski'14].

- Our method is similar to Kurkorski'14 and is based on an **information theoretic** criterion (Information Bottleneck (IB)).

Look-Up Table Design

Numerous LUT design methods [Planjery'13, Declercq'13, Cai'14, Kurkorski'14].

- Our method is similar to Kurkorski'14 and is based on an **information theoretic** criterion (Information Bottleneck (IB)).

LUT Design Principle

Maximization of mutual information between messages and codeword bits.

Look-Up Table Design

Numerous LUT design methods [Planjery'13, Declercq'13, Cai'14, Kurkorski'14].

- Our method is similar to Kurkorski'14 and is based on an **information theoretic** criterion (Information Bottleneck (IB)).

LUT Design Principle

Maximization of mutual information between messages and codeword bits.

- Mutual information between two RVs M and X:

Look-Up Table Design

Numerous LUT design methods [Planjery'13, Declercq'13, Cai'14, Kurkorski'14].

- Our method is similar to Kurkorski'14 and is based on an **information theoretic** criterion (Information Bottleneck (IB)).

LUT Design Principle

Maximization of mutual information between messages and codeword bits.

- Mutual information between two RVs M and X:
  - Denoted by $I(M;X)$.

Look-Up Table Design

Numerous LUT design methods [Planjery'13, Declercq'13, Cai'14, Kurkorski'14].

- Our method is similar to Kurkorski'14 and is based on an **information theoretic** criterion (Information Bottleneck (IB)).

LUT Design Principle

Maximization of mutual information between messages and codeword bits.

- Mutual information between two RVs M and X:
  - Denoted by $I(M;X)$.
  - Quantifies the information about X contained in M (and vice-versa).

Look-Up Table Design

Numerous LUT design methods [Planjery'13, Declercq'13, Cai'14, Kurkorski'14].

- Our method is similar to Kurkorski'14 and is based on an **information theoretic** criterion (Information Bottleneck (IB)).

LUT Design Principle

Maximization of mutual information between messages and codeword bits.

- Mutual information between two RVs M and X:
  - Denoted by $I(M;X)$.
  - Quantifies the information about X contained in M (and vice-versa).
  - Depends on the joint distribution of M and X:

$$p_{M,X}(m,x) = p_X(x)p_{M|X}(m|x)$$

Look-Up Table Design

Numerous LUT design methods [Planjery'13, Declercq'13, Cai'14, Kurkorski'14].

- Our method is similar to Kurkorski'14 and is based on an **information theoretic** criterion (Information Bottleneck (IB)).

LUT Design Principle

Maximization of mutual information between messages and codeword bits.

- Mutual information between two RVs M and X:
  - Denoted by $I(M;X)$.
  - Quantifies the information about X contained in M (and vice-versa).
  - Depends on the joint distribution of M and X:

$$p_{M,X}(m,x) = p_X(x)p_{M|X}(m|x)$$

$p_X(x)$ is usually known, but we **need to calculate** $p_{M|X}(m|x)$.

Look-Up Table Design: Variable Node

Use Density Evolution to compute message probability-mass function:

- CN output messages:

$$p^{(i)}_{\mathsf{m}|\mathsf{x}}(\bar{\mu}|x) = \sum_{\boldsymbol{\mu}\in\mathcal{M}_{\bar{\mu}}} \left(\frac{1}{2}\right)^{dc-2} \sum_{\boldsymbol{x}:\bigoplus \boldsymbol{x}=x} \prod_{j=1}^{d_c-1} p^{(i)}_{\mathsf{m}|\mathsf{x}}(\mu_j|x_j),$$

Look-Up Table Design: Variable Node

Use Density Evolution to compute message probability-mass function:

- CN output messages:

$$p^{(i)}_{\mathsf{m}|\mathsf{x}}(\bar{\mu}|x) = \sum_{\boldsymbol{\mu}\in\mathcal{M}_{\bar{\mu}}} \left(\frac{1}{2}\right)^{dc-2} \sum_{\boldsymbol{x}:\,\bigoplus \boldsymbol{x}=x} \prod_{j=1}^{d_c-1} p^{(i)}_{\mathsf{m}|\mathsf{x}}(\mu_j|x_j),$$

- VN input messages:

$$p^{(i)}_{\mathsf{L},\overline{\mathsf{m}}|\mathsf{x}}(L,\bar{\boldsymbol{\mu}}|x) = \sum_{\boldsymbol{x}:\,x_0=\cdots=x_{d_v-1}=x} p_{\mathsf{L}|\mathsf{x}}(L|x_0) \prod_{j=1}^{d_v-1} p^{(i)}_{\overline{\mathsf{m}}|\mathsf{x}}(\bar{\mu}_j|x_j).$$

Look-Up Table Design: Variable Node

Use Density Evolution to compute message probability-mass function:

- CN output messages:

$$p^{(i)}_{\overline{\mathsf{m}}|\mathsf{x}}(\bar{\mu}|x) = \sum_{\boldsymbol{\mu}\in\mathcal{M}_{\bar{\mu}}} \left(\frac{1}{2}\right)^{dc-2} \sum_{\boldsymbol{x}:\,\bigoplus \boldsymbol{x}=x} \prod_{j=1}^{d_c-1} p^{(i)}_{\mathsf{m}|\mathsf{x}}(\mu_j|x_j),$$

- VN input messages:

$$p^{(i)}_{\mathsf{L},\overline{\mathsf{m}}|\mathsf{x}}(L,\bar{\boldsymbol{\mu}}|x) = \sum_{\boldsymbol{x}:\,x_0=\cdots=x_{d_v-1}=x} p_{\mathsf{L}|\mathsf{x}}(L|x_0) \prod_{j=1}^{d_v-1} p^{(i)}_{\overline{\mathsf{m}}|\mathsf{x}}(\bar{\mu}_j|x_j).$$

Variable Node LUT Design: Optimization Problem

$$\Phi_v^{(i)\,\mathrm{MI}} = \arg\max_{Q\in\mathcal{Q}} I\big(Q(\mathsf{L},\overline{\mathbf{m}}^{(i)});\mathsf{x}\big).$$

Look-Up Table Design: Variable Node

Use Density Evolution to compute message probability-mass function:

- CN output messages:

$$p^{(i)}_{\mathsf{m}|\mathsf{x}}(\bar{\mu}|x) = \sum_{\boldsymbol{\mu}\in\mathcal{M}_{\bar{\mu}}} \left(\frac{1}{2}\right)^{dc-2} \sum_{\boldsymbol{x}:\,\bigoplus \boldsymbol{x}=x} \prod_{j=1}^{d_c-1} p^{(i)}_{\mathsf{m}|\mathsf{x}}(\mu_j|x_j),$$

- VN input messages:

$$p^{(i)}_{\mathsf{L},\overline{\mathsf{m}}|\mathsf{x}}(L,\bar{\boldsymbol{\mu}}|x) = \sum_{\boldsymbol{x}:\,x_0=\cdots=x_{d_v-1}=x} p_{\mathsf{L}|\mathsf{x}}(L|x_0) \prod_{j=1}^{d_v-1} p^{(i)}_{\overline{\mathsf{m}}|\mathsf{x}}(\bar{\mu}_j|x_j).$$

Variable Node LUT Design: Optimization Problem

$$\Phi_v^{(i)\,\mathrm{MI}} = \arg\max_{Q\in\mathcal{Q}} I\big(Q(\mathsf{L},\overline{\mathbf{m}}^{(i)});\mathsf{x}\big).$$

- Can be solved with complexity $O\big(|\mathcal{L}|^3|\mathcal{M}|^{3(d_v-1)}\big)$ [Kurkorski'14].

Look-Up Table Design: Variable Node

Use Density Evolution to compute message probability-mass function:

- CN output messages:

$$p^{(i)}_{\mathsf{m}|\mathsf{x}}(\bar{\mu}|x) = \sum_{\boldsymbol{\mu}\in\mathcal{M}_{\bar{\mu}}} \left(\frac{1}{2}\right)^{dc-2} \sum_{\boldsymbol{x}:\,\bigoplus \boldsymbol{x}=x} \prod_{j=1}^{d_c-1} p^{(i)}_{\mathsf{m}|\mathsf{x}}(\mu_j|x_j),$$

- VN input messages:

$$p^{(i)}_{\mathsf{L},\overline{\mathsf{m}}|\mathsf{x}}(L,\bar{\boldsymbol{\mu}}|x) = \sum_{\boldsymbol{x}:\,x_0=\cdots=x_{d_v-1}=x} p_{\mathsf{L}|\mathsf{x}}(L|x_0) \prod_{j=1}^{d_v-1} p^{(i)}_{\overline{\mathsf{m}}|\mathsf{x}}(\bar{\mu}_j|x_j).$$

Variable Node LUT Design: Optimization Problem

$$\Phi_v^{(i)\,\mathrm{MI}} = \arg\max_{Q\in\mathcal{Q}} I\big(Q(\mathsf{L},\overline{\mathbf{m}}^{(i)});\mathsf{x}\big).$$

- Can be solved with complexity $O\big(|\mathcal{L}|^3|\mathcal{M}|^{3(d_v-1)}\big)$ [Kurkorski'14].
- Check node LUTs can be designed similarly. Due to complexity issues, in this work, **we only examine LUT-based VNs for regular LDPC codes**.

LUT Decoder Performance & Design SNR

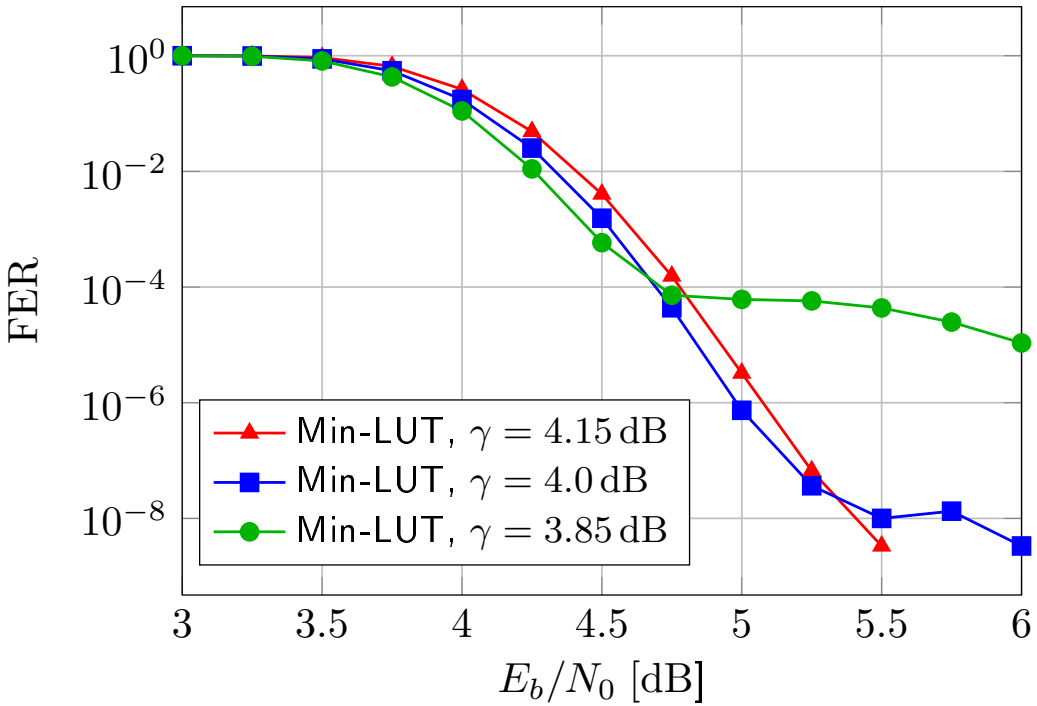
LUT design **depends on channel LLR distribution** $p_{\mathsf{L}|\mathsf{x}}(L|x_0)$.

- AWGN channel: LUT design is **SNR specific**.

LUT Decoder Performance & Design SNR

LUT design **depends on channel LLR distribution** $p_{\mathsf{L}|\mathsf{x}}(L|x_0)$.

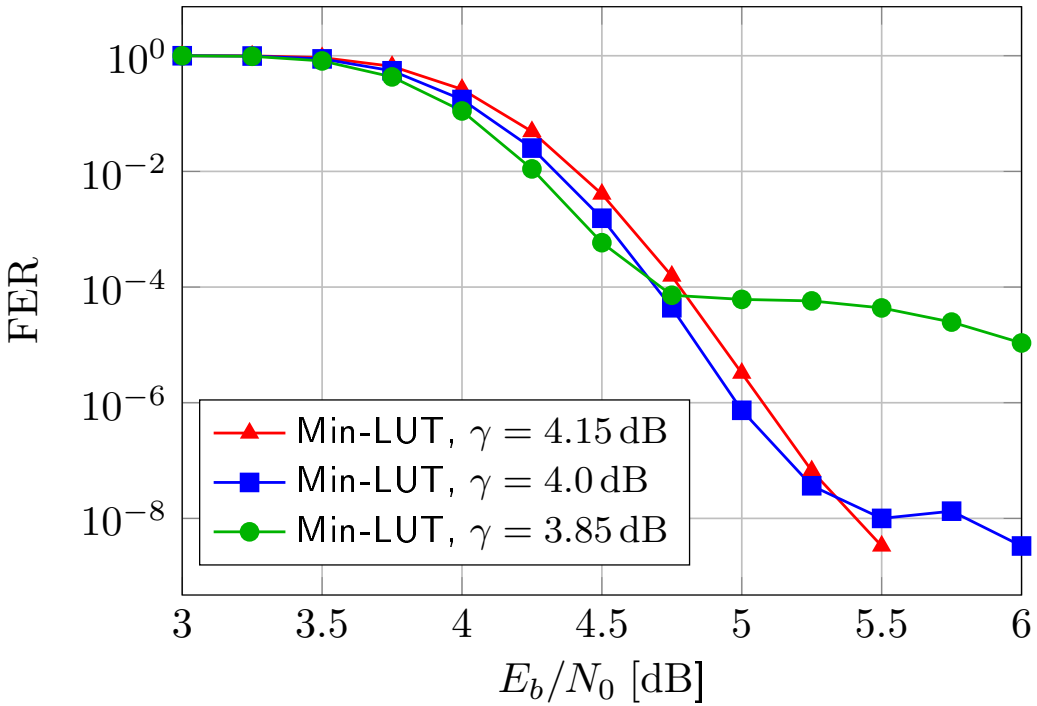
- AWGN channel: LUT design is **SNR specific**.
- Implementation constraint: same LUT used for different SNRs



LUT Decoder Performance & Design SNR

LUT design **depends on channel LLR distribution** $p_{\mathsf{L}|\mathsf{x}}(L|x_0)$.

- AWGN channel: LUT design is **SNR specific**.
- Implementation constraint: same LUT used for different SNRs

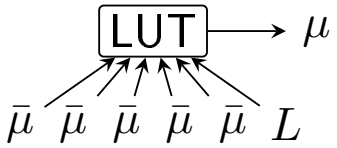


- **Lower design SNR** $\rightarrow$ better **waterfall** region performance.
- **Higher design SNR** $\rightarrow$ better **error floor** region performance.

Practical Considerations: VN LUT Size

Straightforward LUT design:

- **VN LUT size:** $d_v|\mathcal{L}||\mathcal{M}|^{d_v-1}\log|\mathcal{M}|$ bits.



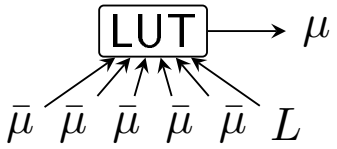
Example (Single LUT)

- $|\mathcal{L}| = |\mathcal{M}| = 32$, $d_v = 6$: 984 kbits **per VN**

Practical Considerations: VN LUT Size

Straightforward LUT design:

- **VN LUT size:** $d_v|\mathcal{L}||\mathcal{M}|^{d_v-1}\log|\mathcal{M}|$ bits.

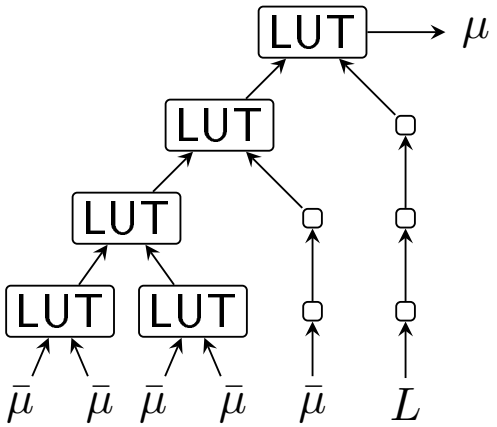


Example (Single LUT)

- $|\mathcal{L}| = |\mathcal{M}| = 32$, $d_v = 6$: 984 kbits **per VN**

Solution: Decompose large LUT into a tree of smaller LUTs.

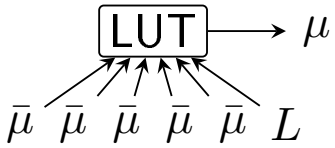
- Significant LUT-size reduction
- Small performance loss expected $rightarrow$ Complexity/performance tradeoff
- Structure and input ordering plays a role



Practical Considerations: VN LUT Size

Straightforward LUT design:

- **VN LUT size:** $d_v|\mathcal{L}||\mathcal{M}|^{d_v-1}\log|\mathcal{M}|$ bits.

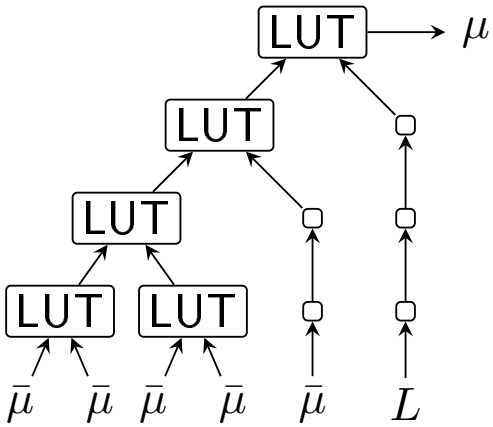


Example (Single LUT)

- $|\mathcal{L}| = |\mathcal{M}| = 32$, $d_v = 6$: 984 kbits **per VN**

Solution: Decompose large LUT into a tree of smaller LUTs.

- Significant LUT-size reduction
- Small performance loss expected $rightarrow$ Complexity/performance tradeoff
- Structure and input ordering plays a role



Example (LUT Tree)

- $|\mathcal{L}| = |\mathcal{M}| = 32$, $d_v = 6$: 26 kbits **per VN**

Practical Considerations: LUT Tree Structure

- Which trees are preferable?

Practical Considerations: LUT Tree Structure

- Which trees are preferable?
  - **Best performance** $\rightarrow$ single-node tree.
  - **Lowest complexity** $\rightarrow$ full binary tree.

Practical Considerations: LUT Tree Structure

- Which trees are preferable?
  - **Best performance** $\rightarrow$ single-node tree.
  - **Lowest complexity** $\rightarrow$ full binary tree.
- **In-between?**

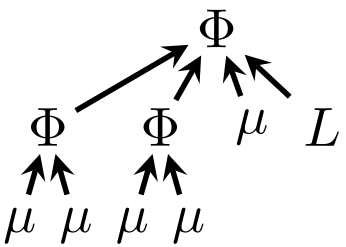
Practical Considerations: LUT Tree Structure

- Which trees are preferable?
  - **Best performance** $\rightarrow$ single-node tree.
  - **Lowest complexity** $\rightarrow$ full binary tree.
- **In-between?**
  - Ordering based on partial order $\geq_{\mathcal{T}}$.
  - Ordering based on heuristic cumulative leaf-root distance metric λ.

Practical Considerations: LUT Tree Structure

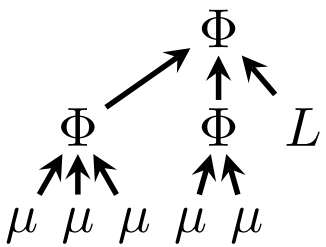
- Which trees are preferable?
  - **Best performance** $\rightarrow$ single-node tree.
  - **Lowest complexity** $\rightarrow$ full binary tree.
- **In-between?**
  - Ordering based on partial order $\geq_{\mathcal{T}}$.
  - Ordering based on heuristic cumulative leaf-root distance metric λ.

T_1: $\lambda = 10$



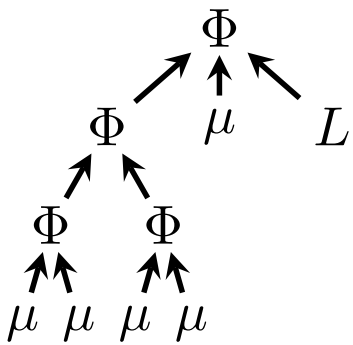
$\sigma_{\text{th}} = 0.5330$

T_2: $\lambda = 11$



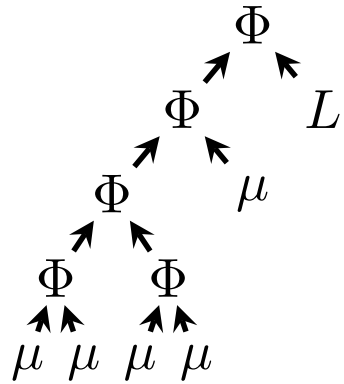
$\sigma_{\text{th}} = 0.5328$

T_3: $\lambda = 14$



$\sigma_{\text{th}} = 0.5313$

T_4: $\lambda = 19$

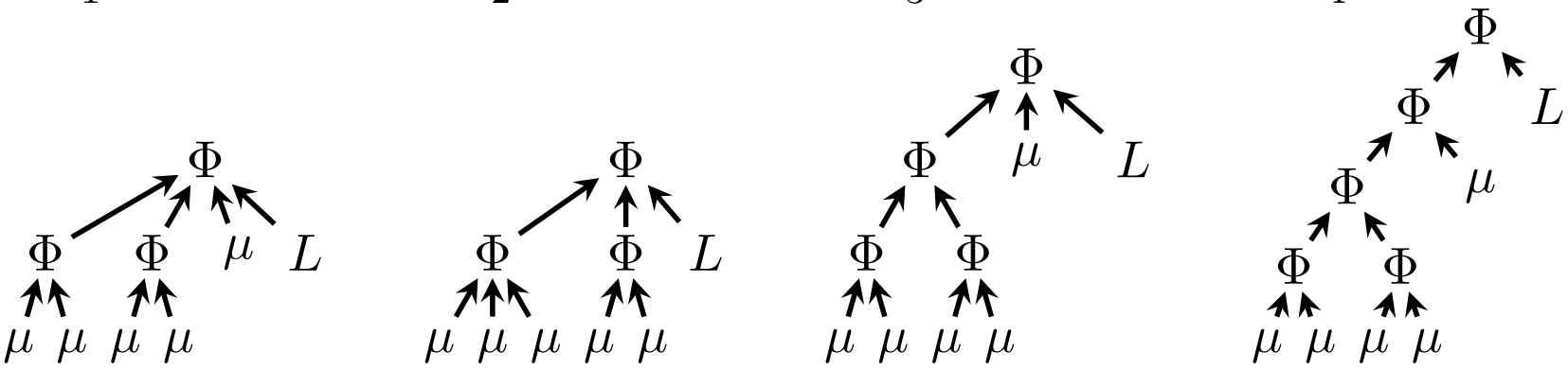


$\sigma_{\text{th}} = 0.5305$

Practical Considerations: LUT Tree Structure

- Which trees are preferable?
  - **Best performance** $\rightarrow$ single-node tree.
  - **Lowest complexity** $\rightarrow$ full binary tree.
- **In-between?**
  - Ordering based on partial order $\geq_{\mathcal{T}}$.
  - Ordering based on heuristic cumulative leaf-root distance metric λ.

T_1: $\lambda = 10$ T_2: $\lambda = 11$ T_3: $\lambda = 14$ T_4: $\lambda = 19$



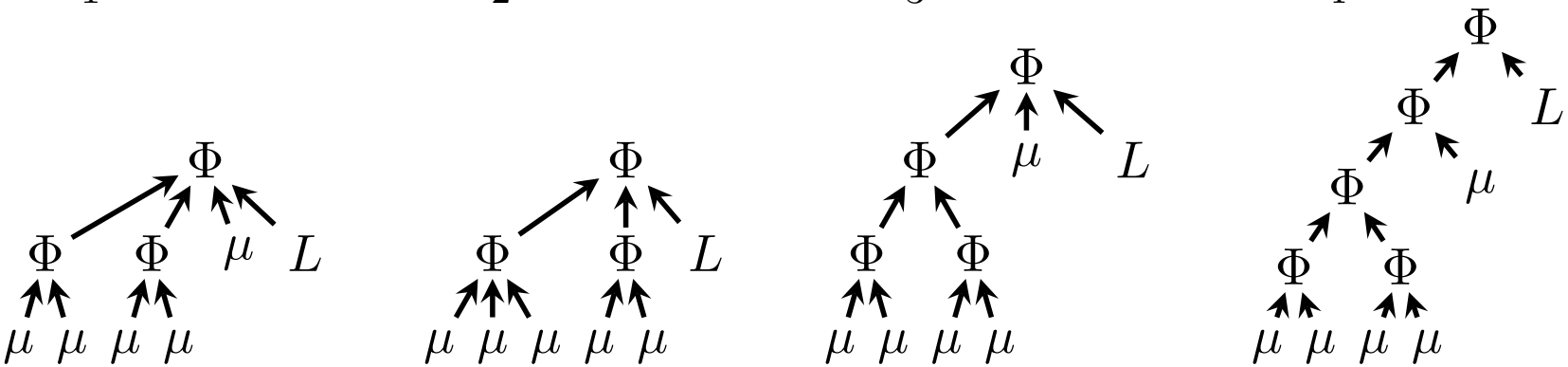
$\sigma_{\text{th}} = 0.5330$ $\sigma_{\text{th}} = 0.5328$ $\sigma_{\text{th}} = 0.5313$ $\sigma_{\text{th}} = 0.5305$

- $T_1 \geq_{\mathcal{T}} T_3 \geq_{\mathcal{T}} T_4$, but, e.g., T_2 and T_1 can not be compared with $\geq_{\mathcal{T}}$.

Practical Considerations: LUT Tree Structure

- Which trees are preferable?
  - **Best performance** $\rightarrow$ single-node tree.
  - **Lowest complexity** $\rightarrow$ full binary tree.
- **In-between?**
  - Ordering based on partial order $\geq_{\mathcal{T}}$.
  - Ordering based on heuristic cumulative leaf-root distance metric λ.

T_1: $\lambda = 10$ T_2: $\lambda = 11$ T_3: $\lambda = 14$ T_4: $\lambda = 19$

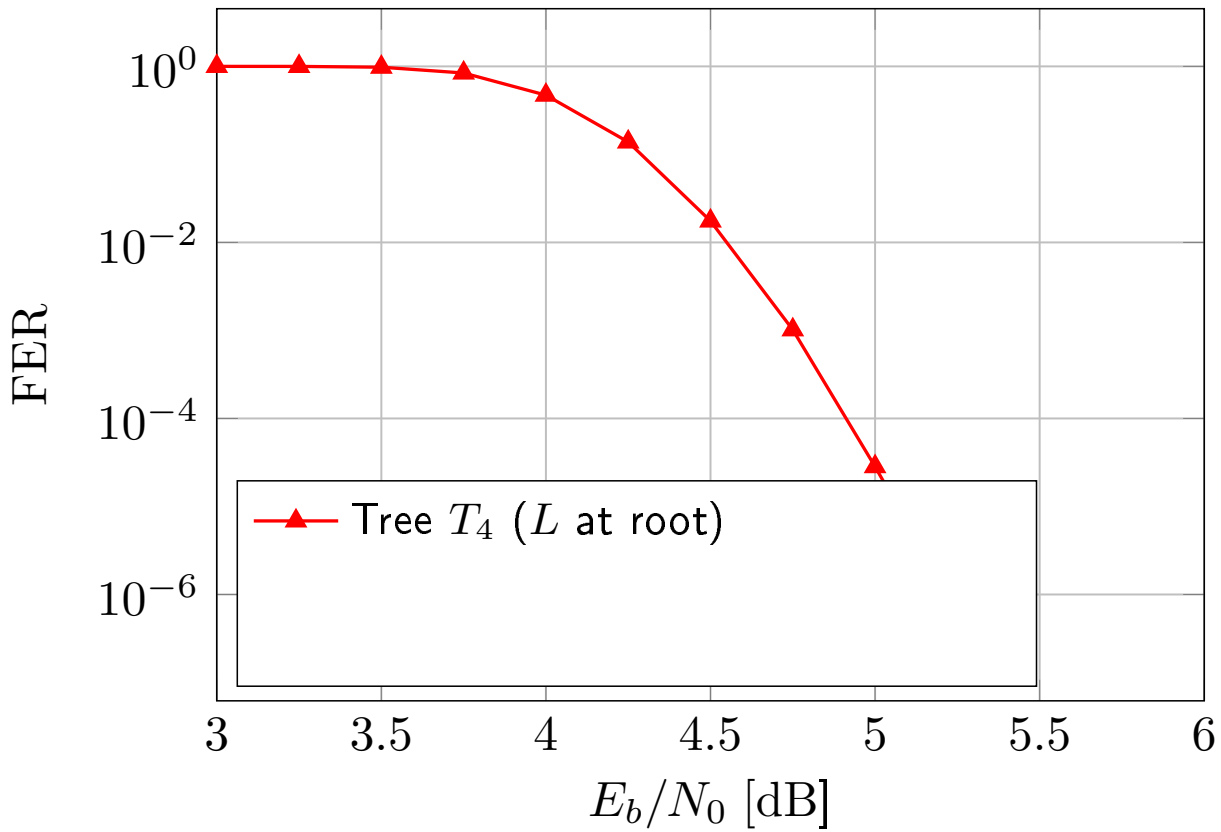


$\sigma_{\text{th}} = 0.5330$ $\sigma_{\text{th}} = 0.5328$ $\sigma_{\text{th}} = 0.5313$ $\sigma_{\text{th}} = 0.5305$

- $T_1 \geq_{\mathcal{T}} T_3 \geq_{\mathcal{T}} T_4$, but, e.g., T_2 and T_1 can not be compared with $\geq_{\mathcal{T}}$.
- Heuristic metric agrees well with density evolution results.

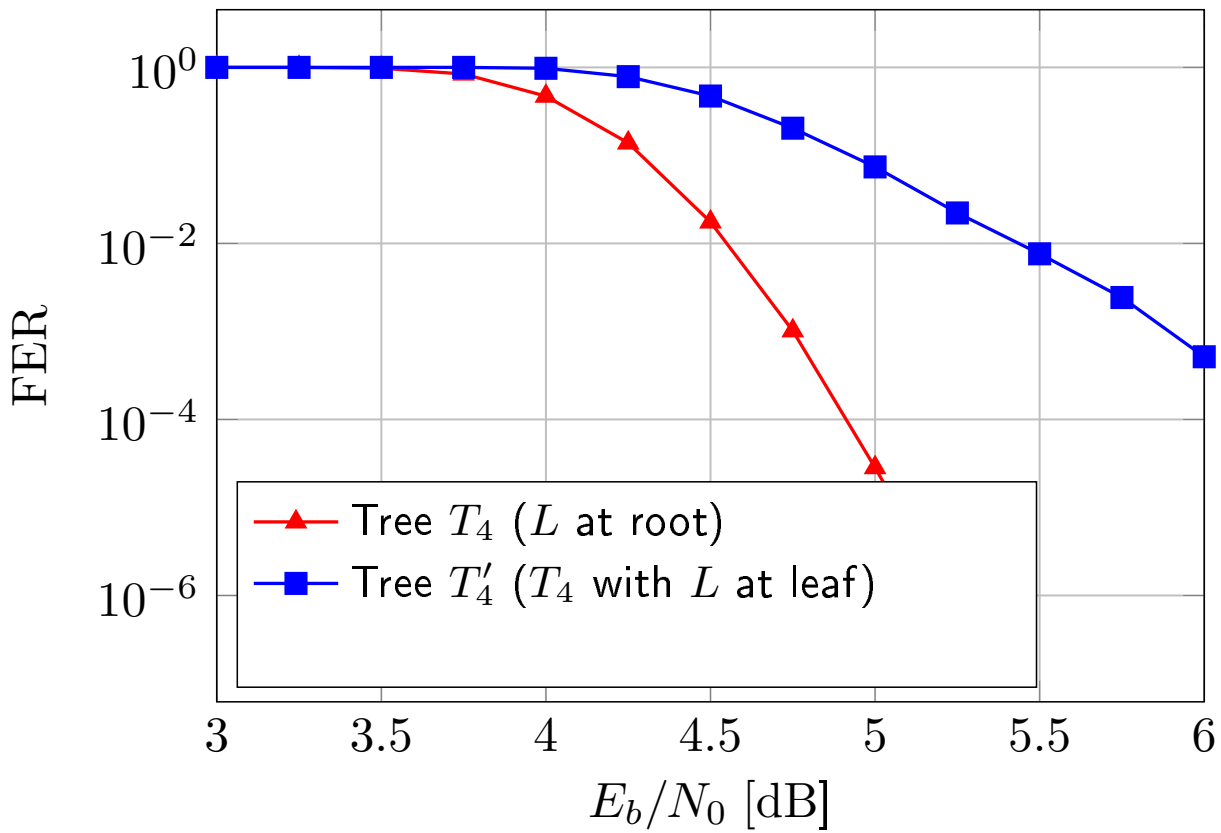
Practical Considerations: Channel LLR Position on LUT Tree

- **Good solution:** L adjacent to the root of the tree.



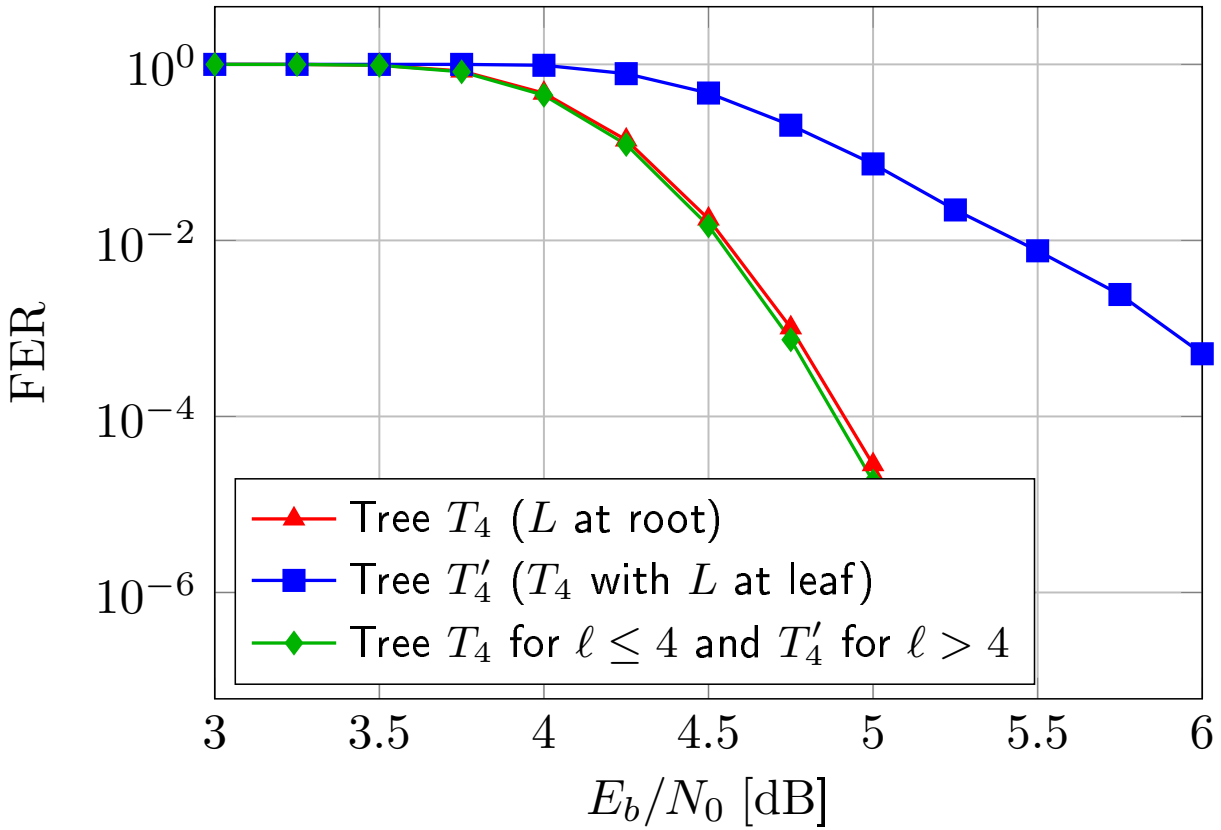
Practical Considerations: Channel LLR Position on LUT Tree

- **Good solution:** L adjacent to the root of the tree.
- **Bad solution:** L far away from the root of the tree.



Practical Considerations: Channel LLR Position on LUT Tree

- **Good solution:** L adjacent to the root of the tree.
- **Bad solution:** L far away from the root of the tree.
- **Ideal solution:** L close to root for first iterations, farther from root as it becomes more irrelevant.



Practical Considerations: Check Node

Check nodes are ideally also be designed using LUTs.

Unfortunately, CNs can have a large degree (number of inputs) $\rightarrow$ Even tree-structured LUTs become too large/complex.

Practical Considerations: Check Node

Check nodes are ideally also be designed using LUTs.

Unfortunately, CNs can have a large degree (number of inputs) $\rightarrow$ Even tree-structured LUTs become too large/complex.

- For **symmetric channels**:

Practical Considerations: Check Node

Check nodes are ideally also be designed using LUTs.

Unfortunately, CNs can have a large degree (number of inputs) $\rightarrow$ Even tree-structured LUTs become too large/complex.

- For **symmetric channels**:
  - Ensure labels are sorted identically to message values:

$$\mu_k < \mu_l \Leftrightarrow \mathcal{B}(\mu_k) < \mathcal{B}(\mu_l), \quad \forall k, l \in 1, \ldots, |\mathcal{M}|.$$

Practical Considerations: Check Node

Check nodes are ideally also be designed using LUTs.

Unfortunately, CNs can have a large degree (number of inputs) $\rightarrow$ Even tree-structured LUTs become too large/complex.

- For **symmetric channels**:
  - Ensure labels are sorted identically to message values:

$$\mu_k < \mu_l \Leftrightarrow \mathcal{B}(\mu_k) < \mathcal{B}(\mu_l), \quad \forall k, l \in 1, \ldots, |\mathcal{M}|.$$

  - Message sign **follows from index**:

$$\text{sign}(\mu_k) = \begin{cases} -1, & 1 \leq k \leq \frac{|\mathcal{M}|}{2}, \\ +1, & \frac{|\mathcal{M}|}{2} < k \leq |\mathcal{M}|. \end{cases}$$

Practical Considerations: Check Node

Check nodes are ideally also be designed using LUTs.

Unfortunately, CNs can have a large degree (number of inputs) $\rightarrow$ Even tree-structured LUTs become too large/complex.

- For **symmetric channels**:
  - Ensure labels are sorted identically to message values:

$$\mu_k < \mu_l \Leftrightarrow \mathcal{B}(\mu_k) < \mathcal{B}(\mu_l), \quad \forall k, l \in 1, \ldots, |\mathcal{M}|.$$

  - Message sign **follows from index**:

$$\text{sign}(\mu_k) = \begin{cases} -1, & 1 \leq k \leq \frac{|\mathcal{M}|}{2}, \\ +1, & \frac{|\mathcal{M}|}{2} < k \leq |\mathcal{M}|. \end{cases}$$

  - Minimum can be found **directly from indices**.

Practical Considerations: Check Node

Check nodes are ideally also be designed using LUTs.

Unfortunately, CNs can have a large degree (number of inputs) $\rightarrow$ Even tree-structured LUTs become too large/complex.

- For **symmetric channels**:
  - Ensure labels are sorted identically to message values:

$$\mu_k < \mu_l \Leftrightarrow \mathcal{B}(\mu_k) < \mathcal{B}(\mu_l), \quad \forall k, l \in 1, \ldots, |\mathcal{M}|.$$

  - Message sign **follows from index**:

$$\text{sign}(\mu_k) = \begin{cases} -1, & 1 \leq k \leq \frac{|\mathcal{M}|}{2}, \\ +1, & \frac{|\mathcal{M}|}{2} < k \leq |\mathcal{M}|. \end{cases}$$

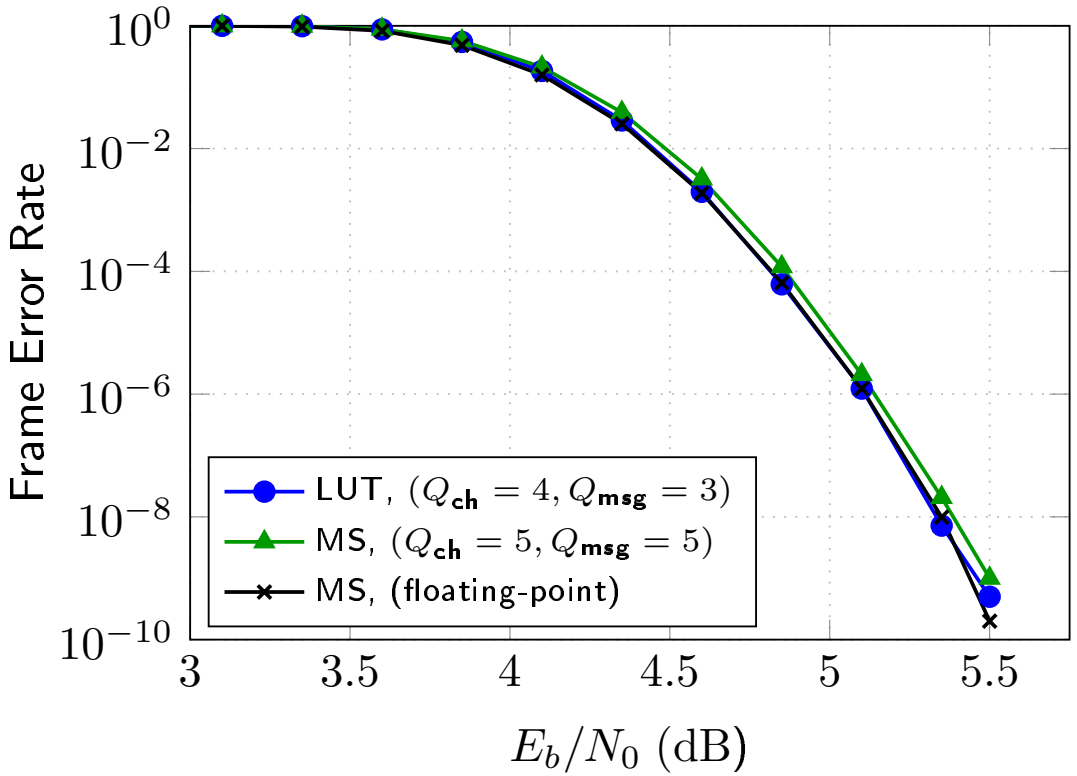
  - Minimum can be found **directly from indices**.

"Min-LUT" Decoder

Entire decoder can be implemented based on **message labels** and CN uses **standard min-sum rule**.

Quantized Message Passing: Quantization

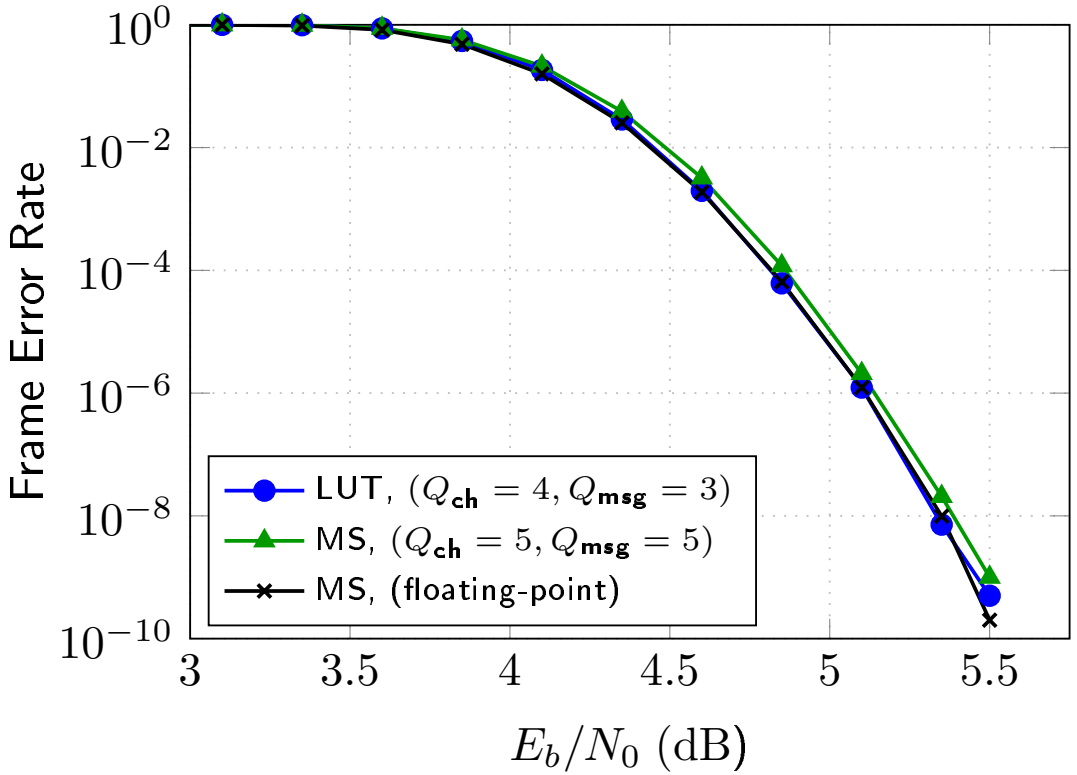
FER performance comparison to Min-Sum decoder with message quantization:



- Message quantization with $Q_{msg} \geq 3$ bit is sufficient.

Quantized Message Passing: Quantization

FER performance comparison to Min-Sum decoder with message quantization:



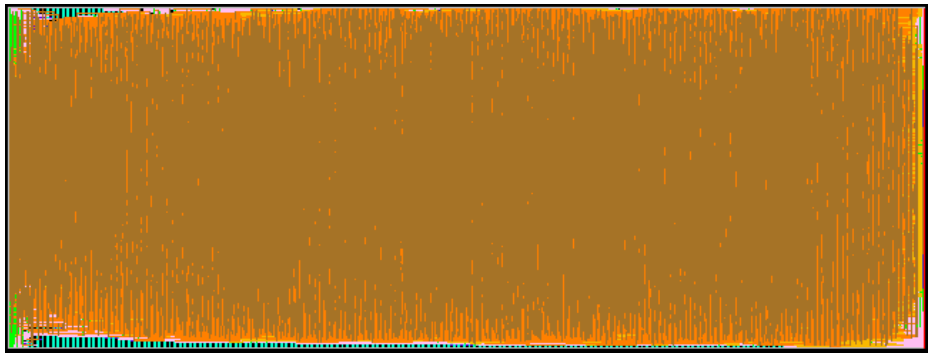
- Message quantization with $Q_{msg} \geq 3$ bit is sufficient.

Quantized Message Passing provides **better performance** than regular Min-Sum **with** 40% **fewer message quantization bits**.

Unrolled Quantized Message Passing: Results

Quantization of messages with $Q_{msg} = 3$ bit

- Automatic P&R is finally feasible with **65.9% layout density**

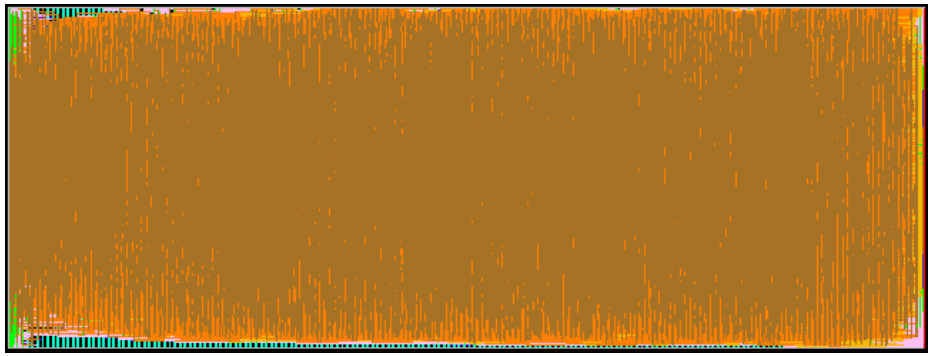


Layout in 28 nm FD-SOI technology

Unrolled Quantized Message Passing: Results

Quantization of messages with $Q_{msg} = 3$ bit

- Automatic P&R is finally feasible with **65.9% layout density**



Layout in 28 nm FD-SOI technology

	Unrolled Min-Sum	Unrolled LUT
Msg. Quantization	5 bit	3 bit
Components area (CN/VN) Delay (logic/routing)	3607 μm^2 / 755 μm^2 2.38 ns / 5×1.51 ns	1510 μm^2 / 646 μm^2 1.42 ns / 3×1.16 ns
Core area Throughput	23.3 mm^2 271 Gbps	**16.2 mm^2** **588 Gbps**
Energy efficiency Area efficiency	45.2 pJ/bit 11.6 Gpbs/mm^2	22.7 pJ/bit 36.3 Gpbs/mm^2

Conclusions

- Channel codes are moving to higher and higher data rates
- Process scaling provides diminishing returns in speed and power

Conclusions

- Channel codes are moving to higher and higher data rates
- Process scaling provides diminishing returns in speed and power

- Highly parallel architectures can only partially meet the increasing demand for high throughput
- Main limitations
  - Routing overhead
  - Registers and storage

Conclusions

- Channel codes are moving to higher and higher data rates
- Process scaling provides diminishing returns in speed and power

- Highly parallel architectures can only partially meet the increasing demand for high throughput
- Main limitations
  - Routing overhead
  - Registers and storage

- Further algorithm improvements needed to keep complexity under control
- Need more collaboration between algorithm and architecture design
- Wordlength reduction is one of the most promising objectives

More Information & Code Online

https://www.nt.tuwien.ac.at/UNFOLD