Generalized Tree Architecture with High-Radix Processing for an SC Polar Decoder

Tae-Hwan Kim

Korea Aerospace University

Outline

- Background
    - Polar codes
    - SC decoding / decoder architecture
- Proposed Architecture
    - Generalized tree arch. with high-radix proc.
    - Architecture synthesis for SC decoders
    - Case study: 1K-bit SC decoder
- Conclusion

Polar Code [*]

- (Ideally) most powerful ECC ever
  - Asymptotically achieving channel capacity
  - Compelling BER performance in practice
- Ch. Polarization by Combining/Splitting Ch.’s
- Adopted for 5G comm. standard
- Being considered for the storage applications

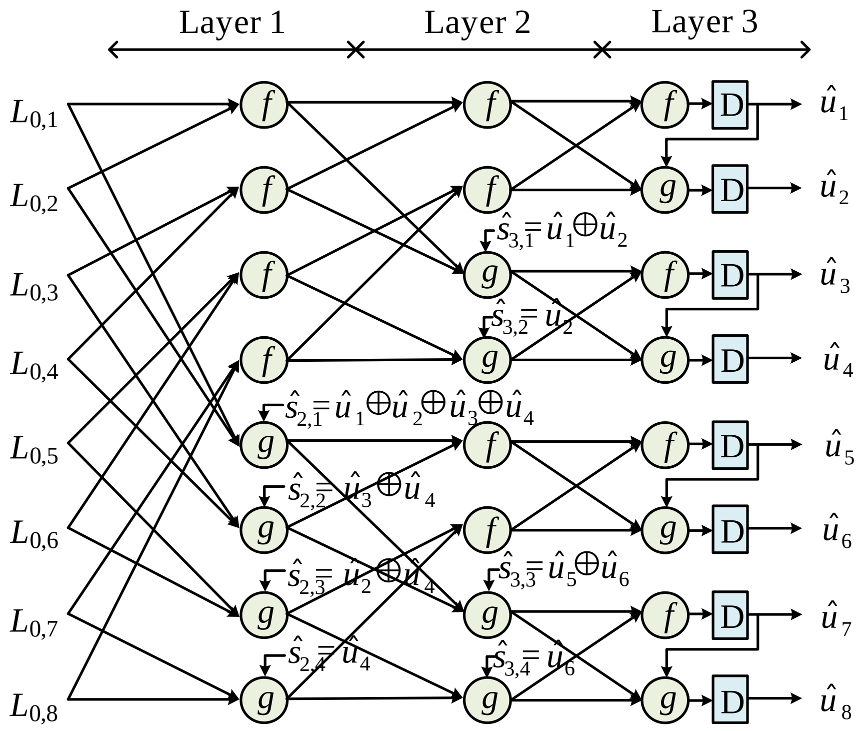
[*] E. Arıkan, “Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels,” *IEEE Trans. Inf. Theory*, vol. 55, no. 7, pp. 3051–3073, Jul. 2009.

SC Decoding [*]

- Standard algorithm to decode a polar code.
- Naïve but super important in practice.
  - Core component for other advanced algorithms
    - SC List, SC Flip, etc.
- Problematic serialized operations.
  - Critical to realize a high-speed decoding.

[*] E. Arıkan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Trans. Inf. Theory*, vol. 55, no. 7, pp. 3051–3073, Jul. 2009.

SC Decoding: Algorithm



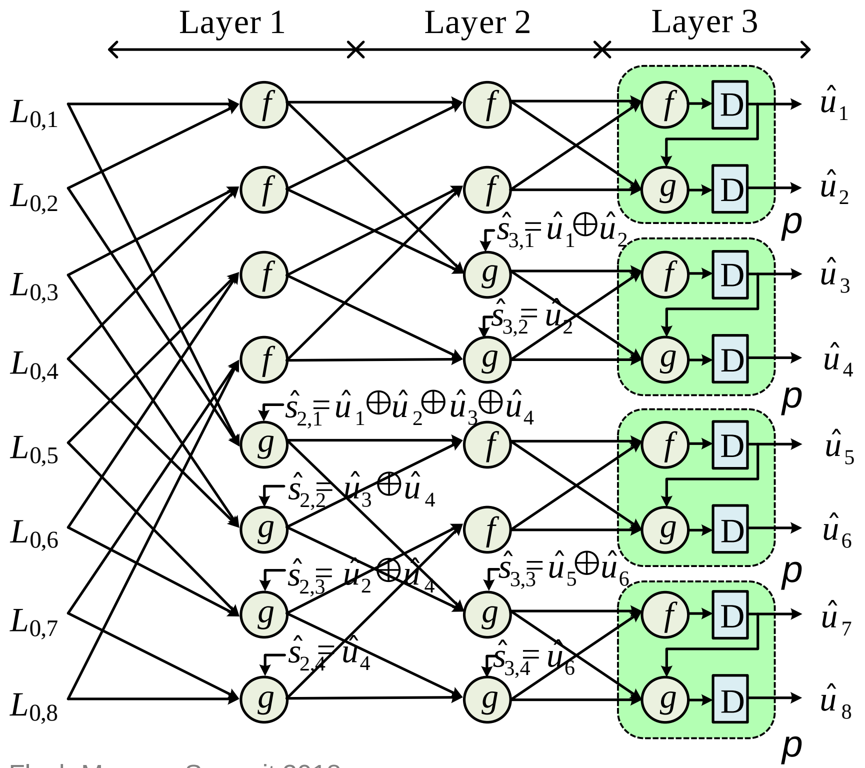
Layer 1
Layer 2
Layer 3
$L_{0,1}$
$L_{0,2}$
$L_{0,3}$
$L_{0,4}$
$L_{0,5}$
$L_{0,6}$
$L_{0,7}$
$L_{0,8}$
f
g
D
$\hat{s}_{3,1} = \hat{u}_1 \oplus \hat{u}_2$
$\hat{s}_{3,2} = \hat{u}_2$
$\hat{s}_{2,1} = \hat{u}_1 \oplus \hat{u}_2 \oplus \hat{u}_3 \oplus \hat{u}_4$
$\hat{s}_{2,2} = \hat{u}_3 \oplus \hat{u}_4$
$\hat{s}_{2,3} = \hat{u}_2 \oplus \hat{u}_4$
$\hat{s}_{2,4} = \hat{u}_4$
$\hat{s}_{3,3} = \hat{u}_5 \oplus \hat{u}_6$
$\hat{s}_{3,4} = \hat{u}_6$
$\hat{u}_1$
$\hat{u}_2$
$\hat{u}_3$
$\hat{u}_4$
$\hat{u}_5$
$\hat{u}_6$
$\hat{u}_7$
$\hat{u}_8$

$$\hat{u}_i = \begin{cases} 1, \text{if } i \in A \text{ and } L(u_i) < 0 \\ 0, \text{otherwise,} \end{cases}$$

$$\underbrace{L(u_i)}_{\text{}} = \log \frac{P\left(\mathbf{y}, \hat{\mathbf{u}}_0^{i-1} | u_i = 0\right)}{P\left(\mathbf{y}, \hat{\mathbf{u}}_0^{i-1} | u_i = 1\right)}$$

Calculated through layers in a trellis.

SC Decoding: Algorithm



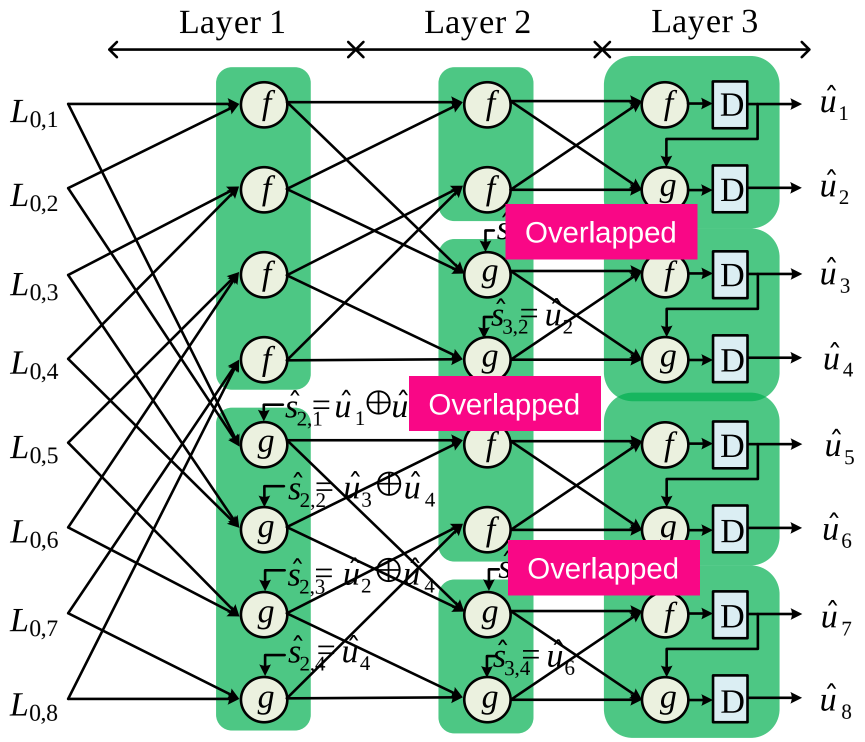
Layer 1
Layer 2
Layer 3
$L_{0,1}$
$L_{0,2}$
$L_{0,3}$
$L_{0,4}$
$L_{0,5}$
$L_{0,6}$
$L_{0,7}$
$L_{0,8}$
f
g
D
$\hat{s}_{3,1}=\hat{u}_1\oplus\hat{u}_2$
$\hat{s}_{3,2}=\hat{u}_2$
$\hat{s}_{2,1}=\hat{u}_1\oplus\hat{u}_2\oplus\hat{u}_3\oplus\hat{u}_4$
$\hat{s}_{2,2}=\hat{u}_3\oplus\hat{u}_4$
$\hat{s}_{2,3}=\hat{u}_2\oplus\hat{u}_4$
$\hat{s}_{2,4}=\hat{u}_4$
$\hat{s}_{3,3}=\hat{u}_5\oplus\hat{u}_6$
$\hat{s}_{3,4}=\hat{u}_6$
$\hat{u}_1$
$\hat{u}_2$
$\hat{u}_3$
$\hat{u}_4$
$\hat{u}_5$
$\hat{u}_6$
$\hat{u}_7$
$\hat{u}_8$
p

$$f(a,b) = sign(a)sign(b)\min(|a|,|b|)$$
$$g(a,b,s) = (-1)^s a + b$$

Can be merged into "p kernel" in the last layer. [*]
→ Multibit decoding.

[*] B. Yuan and K. K. Parhi, "Low-latency successive-cancellation polar decoder architectures using 2-bit decoding," *IEEE TCAS I*, vol. 61, no. 4, pp. 1241–1254, Apr. 2014.

SC Decoding: Scheduling

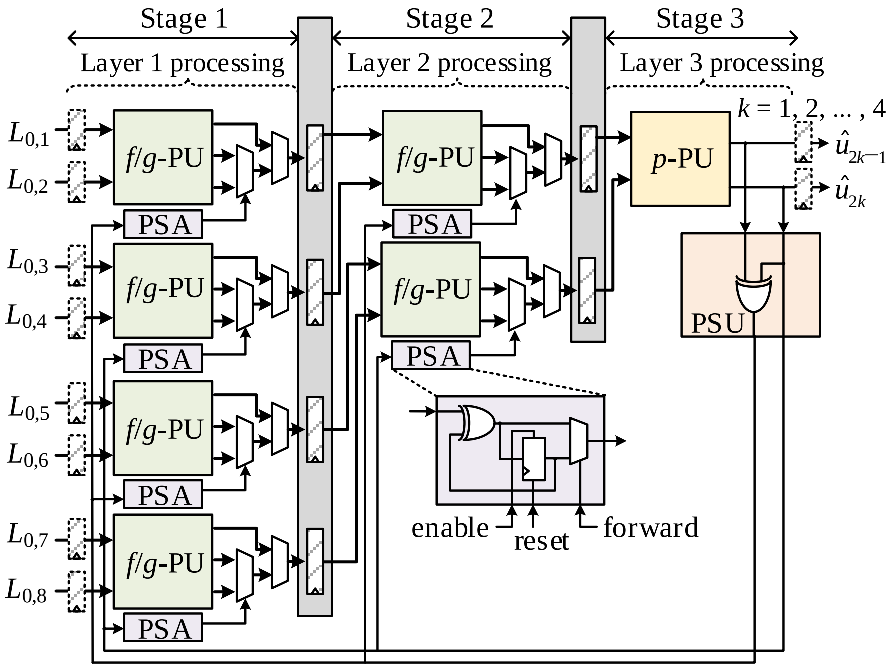


Layer 1
Layer 2
Layer 3
$L_{0,1}$
$L_{0,2}$
$L_{0,3}$
$L_{0,4}$
$L_{0,5}$
$L_{0,6}$
$L_{0,7}$
$L_{0,8}$
$\hat{s}_{2,1} = \hat{u}_1 \oplus \hat{u}$
$\hat{s}_{2,2} = \hat{u}_3 \oplus \hat{u}_4$
$\hat{s}_{2,3} = \hat{u}_2 \oplus \hat{u}_4$
$\hat{s}_{2,4} = \hat{u}_4$
$\hat{s}_{3,2} = \hat{u}_2$
$\hat{s}_{3,4} = \hat{u}_6$
Overlapped
Overlapped
Overlapped
$\hat{u}_1$
$\hat{u}_2$
$\hat{u}_3$
$\hat{u}_4$
$\hat{u}_5$
$\hat{u}_6$
$\hat{u}_7$
$\hat{u}_8$

Processing in each step depends on the partial sum.

$N - 1$ steps for a N-bit code.

Conv. Tree Arch. for SC Decoding [*,**]

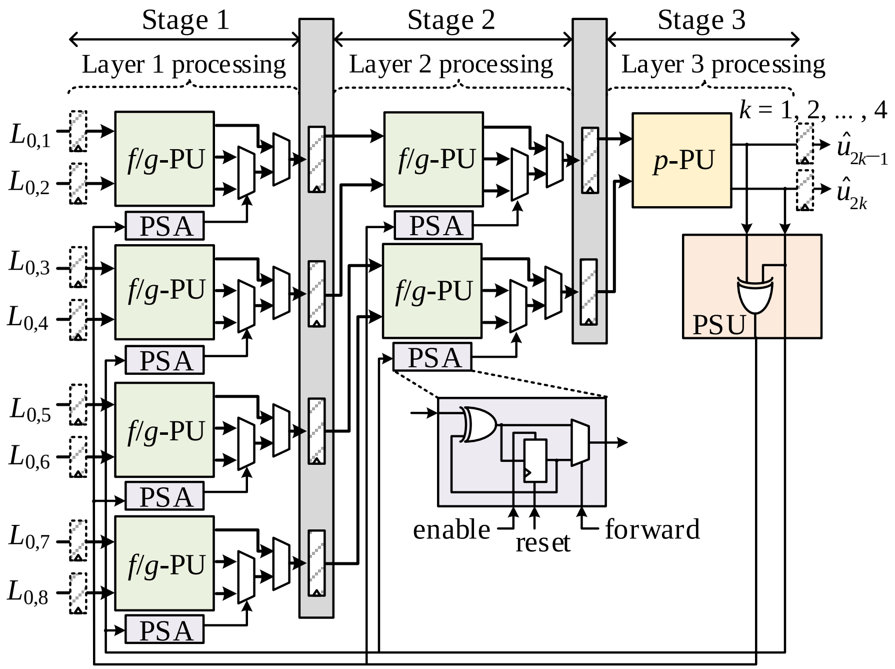


Each PU processes a kernel.

Each stage is made by instantiating as many PUs as to process several kernels that can be scheduled with in a step.

[*] C. Leroux, I. Tal, A. Vardy, and W. J. Gross, “Hardware architectures for successive cancellation decoding of polar codes,” *ICASSP*, May 2011, pp. 1665–1668.
[**] B. Yuan and K. K. Parhi, “Low-latency successive-cancellation polar decoder architectures using 2-bit decoding,” *IEEE TCAS I*, vol. 61, no. 4, pp. 1241–1254, Apr. 2014.

Conv. Tree Arch. for SC Decoding



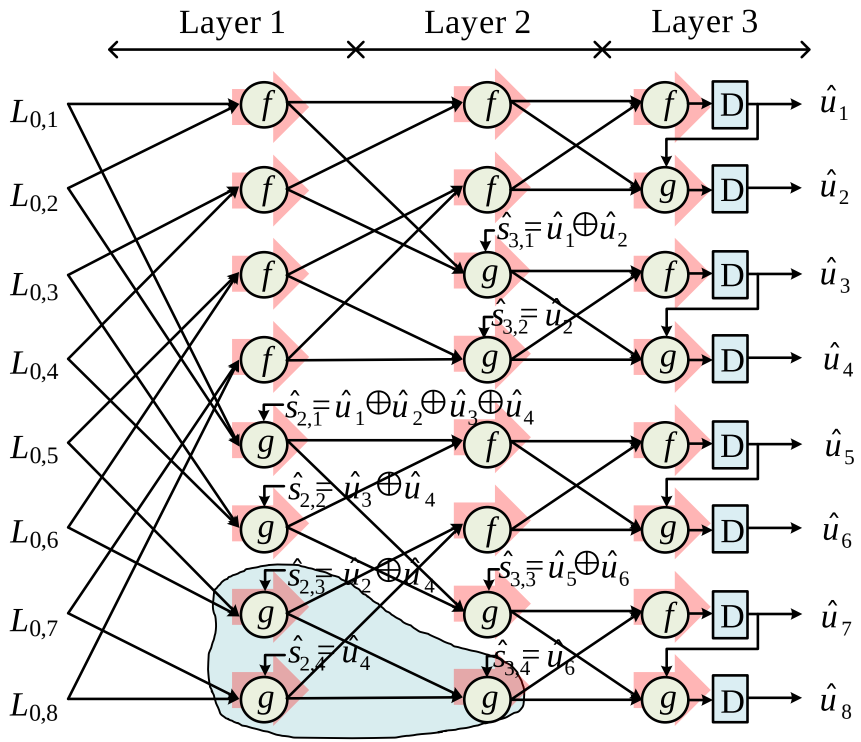
Cycle	1	2	3	4	5	6	7
Stage 1	*f*			*g*			
Stage 2		*f*	*g*		*f*	*g*	
Stage 3			*p*	*p*		*p*	*p*

Throughput:

K/7 bits per cycle,

K: the information bits

Motivation of High-Radix Kernel Proc.



Layer 1
Layer 2
Layer 3
$L_{0,1}$
$L_{0,2}$
$L_{0,3}$
$L_{0,4}$
$L_{0,5}$
$L_{0,6}$
$L_{0,7}$
$L_{0,8}$
$\hat{s}_{3,1} = \hat{u}_1 \oplus \hat{u}_2$
$\hat{s}_{3,2} = \hat{u}_2$
$\hat{s}_{2,1} = \hat{u}_1 \oplus \hat{u}_2 \oplus \hat{u}_3 \oplus \hat{u}_4$
$\hat{s}_{2,2} = \hat{u}_3 \oplus \hat{u}_4$
$\hat{s}_{2,3} = \hat{u}_2 \oplus \hat{u}_4$
$\hat{s}_{3,3} = \hat{u}_5 \oplus \hat{u}_6$
$\hat{s}_{2,4} = \hat{u}_4$
$\hat{s}_{3,4} = \hat{u}_6$
D
$\hat{u}_1$
$\hat{u}_2$
$\hat{u}_3$
$\hat{u}_4$
$\hat{u}_5$
$\hat{u}_6$
$\hat{u}_7$
$\hat{u}_8$

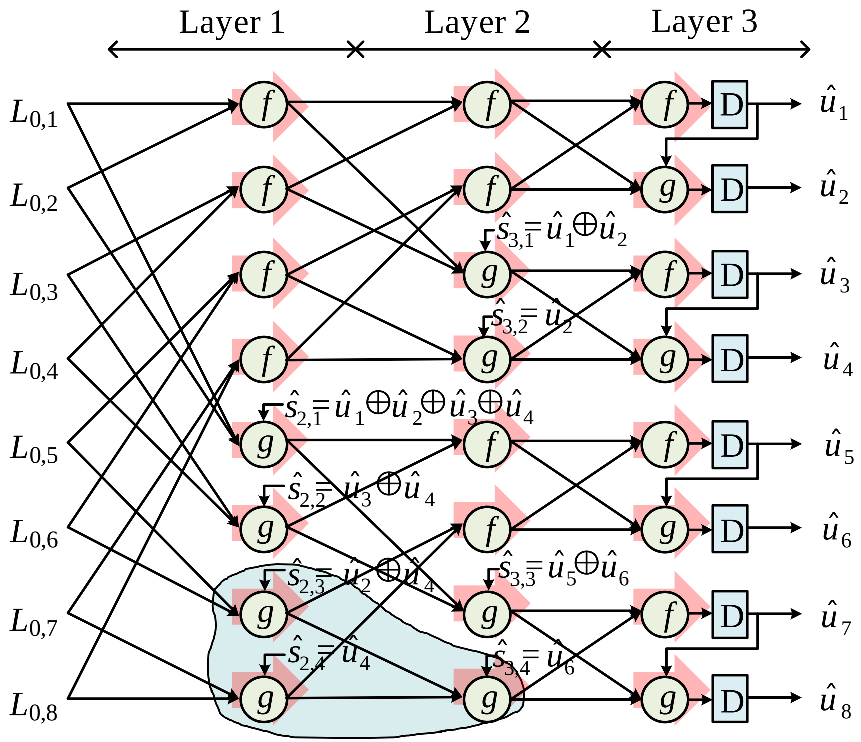
Radix-4 Kernel

Conv. arch. considers only "per-cycle radix-2 kernel proc".

How about "per-cycle high-radix kernel proc."?

Combining several radix-2 kernels through $\log_2 r$ layers → radix-r kernel. ($r > 2$: high-radix kernel)

Motivation of High-Radix Kernel Proc.


Layer 1
Layer 2
Layer 3
$L_{0,1}$
$L_{0,2}$
$L_{0,3}$
$L_{0,4}$
$L_{0,5}$
$L_{0,6}$
$L_{0,7}$
$L_{0,8}$
$\hat{s}_{3,1}=\hat{u}_1\oplus\hat{u}_2$
$\hat{s}_{3,2}=\hat{u}_2$
$\hat{s}_{2,1}=\hat{u}_1\oplus\hat{u}_2\oplus\hat{u}_3\oplus\hat{u}_4$
$\hat{s}_{2,2}=\hat{u}_3\oplus\hat{u}_4$
$\hat{s}_{2,3}=\hat{u}_2\oplus\hat{u}_4$
$\hat{s}_{3,3}=\hat{u}_5\oplus\hat{u}_6$
$\hat{s}_{2,4}=\hat{u}_4$
$\hat{s}_{3,4}=\hat{u}_6$
D
$\hat{u}_1$
$\hat{u}_2$
$\hat{u}_3$
$\hat{u}_4$
$\hat{u}_5$
$\hat{u}_6$
$\hat{u}_7$
$\hat{u}_8$

Radix-4 Kernel

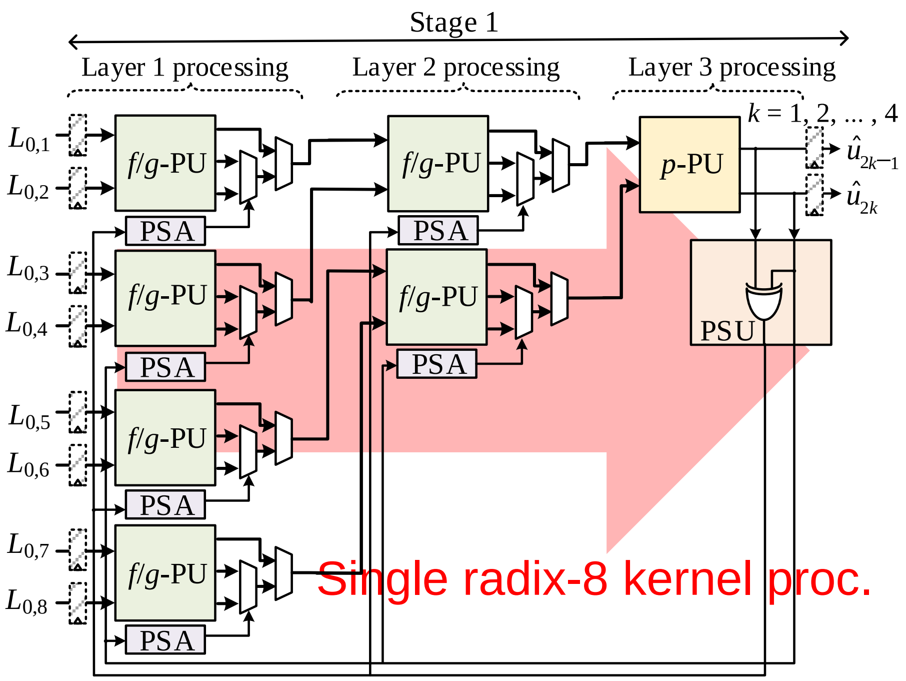
Conv. arch. considers only "per-cycle radix-2 kernel proc".

How about "per-cycle high-radix kernel proc."?

# of cycles ↓ ☺ cycle period ↑ ☹

# of stages ↓
# of reg's ↓ ☺

Modified 8-bit SC Decoder for Per-Cycle Radix-8 Kernel Proc.

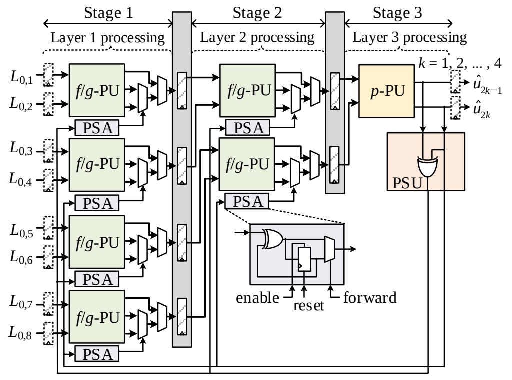
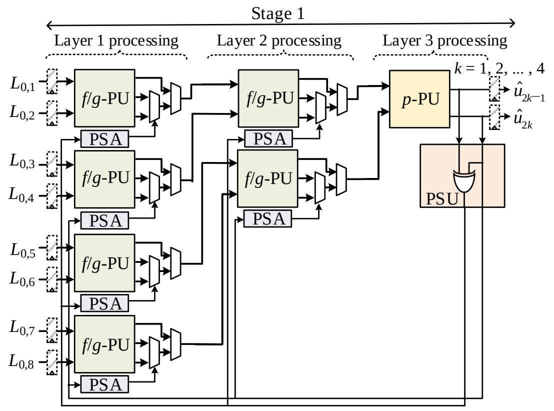


Single stage processes a radix-8 kernel.

Cycle	1	2	3	4
Stage 1	*ffp*	*fgp*	*gfp*	*ggp*

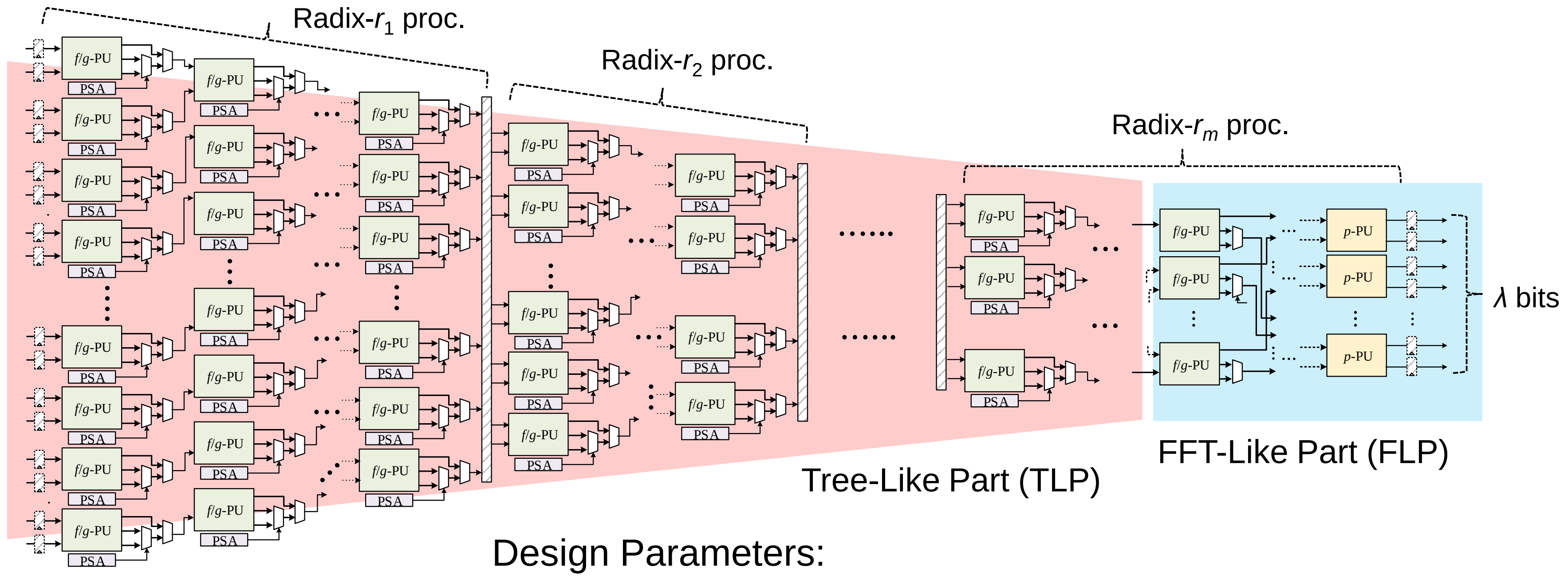
Throughput:
K/4 bits per cycle

Conv. Vs. Modified

	Conventional Arch. (radix-2 kernel proc.)	Modified Arch. (radix-8 kernel proc.)
Arch.	 Stage 1 Stage 2 Stage 3 Layer 1 processing Layer 2 processing Layer 3 processing $k = 1, 2, \ldots, 4$ $L_{0,1}$ $L_{0,2}$ $L_{0,3}$ $L_{0,4}$ $L_{0,5}$ $L_{0,6}$ $L_{0,7}$ $L_{0,8}$ f/g-PU PSA p-PU PSU $\hat{u}_{2k-1}$ $\hat{u}_{2k}$ enable reset forward	 Stage 1 Layer 1 processing Layer 2 processing Layer 3 processing $k = 1, 2, \ldots, 4$ $L_{0,1}$ $L_{0,2}$ $L_{0,3}$ $L_{0,4}$ $L_{0,5}$ $L_{0,6}$ $L_{0,7}$ $L_{0,8}$ f/g-PU PSA p-PU PSU $\hat{u}_{2k-1}$ $\hat{u}_{2k}$
Throughput	*K* / 7 bits per cycle ☹	*K* / 4 bits per (longer) cycle 😐
Complexity	6 *f*/*g*-PU + 6 PSA + *p*-PU + 6Q REGs + 2-bit ENC ☹	6 *f*/*g*-PU + 6 PSA + *p*-PU + 2-bit ENC 🙂

How to generalize the architecture considering the high-radix proc.?

Supporting High-Radix Kernel Processing; Generalized Tree Arch.



Design Parameters:

Radix set: $\mathbf{r} = [r_1, r_2, \ldots r_m]$, $r_i = 2^{k_i}$, $1 \leq k_i \leq n$, $\sum k_i = n$.

Multibit decoding factor: $\lambda \leq r_m$

How to Determine the Design Parameters?

- Gen. arch. is structured completely by ($\mathbf{r}$, λ).
- How to determine ($\mathbf{r}$, λ) ?
  - Based on what criterion?
    - To achieve/meet a certain design objective or constraint
    - Need to find the relations

      – ($\mathbf{r}$, λ) ➔ overall complexity / throughput

$(\mathbf{r}, \lambda) \rightarrow$ Complexity

$$N_{(\mathrm{MUX})} = \underbrace{(2\boxed{Q}+1)\cdot(2^n-\lambda)}_{\text{MUXes in the TLP}} + \underbrace{\boxed{Q}\cdot(\log_2\lambda-1)\cdot\frac{\lambda}{2}}_{\text{MUXes in the FLP}}, \quad N_{(\mathrm{XOR})} = \underbrace{\frac{\lambda}{2}\cdot\log_2\lambda}_{\text{PSU}} + \underbrace{2^n-\lambda}_{\text{PSAs}},$$

$$N_{(\mathrm{REG})} = \underbrace{\boxed{Q}\cdot\sum_{i=1}^{m-1}\left(\prod_{k=i+1}^{m} r_k\right)}_{\text{LLR storage (between stages)}} + \underbrace{2^n-\lambda}_{\text{PSAs}}, \quad N_{(f/g\text{ - PU})} = \underbrace{2^{n-1}+2^{n-2}+\cdots+\lambda}_{\text{TLP}} + \underbrace{(\log_2\lambda-1)\cdot\frac{\lambda}{2}}_{\text{FLP}}$$

$$C = \boxed{C_{(f/g\text{-PU})}}\cdot N_{(f/g\text{-PU})} + \boxed{C_{(p\text{-PU})}}\cdot N_{(p\text{-PU})} + \boxed{C_{(\mathrm{REG})}}\cdot N_{(\mathrm{REG})} + \boxed{C_{(\mathrm{MUX})}}\cdot N_{(\mathrm{MUX})} + \boxed{C_{(\mathrm{XOR})}}\cdot N_{(\mathrm{XOR})},$$

🟩 Constant parameter; Determined by microarch., technology, simulations

(r, λ) ➔ Critical Path

$$\delta_{(\text{Stage } i)} = \boxed{\delta_{(\text{REG})}} + \left(\max\left(\boxed{\delta_{(f\text{-PU})}}, \boxed{\delta_{(g\text{-PU})}} + \underbrace{\boxed{\delta_{(\text{MUX})}}}_{g^{(0)} \text{ or } g^{(1)}} \right) + \underbrace{\boxed{\delta_{(\text{MUX})}}}_{f \text{ or } g} \right) \cdot \log_2 r_i, \quad \text{for } 1 \le i < m,$$

$$\delta_{(\text{Stage } m)} = \boxed{\delta_{(\text{REG})}} + \underbrace{\left(\max\left(\boxed{\delta_{(f\text{-PU})}}, \boxed{\delta_{(g\text{-PU})}} + \boxed{\delta_{(\text{MUX})}}\right) + \boxed{\delta_{(\text{MUX})}}\right) \cdot \log_2 \left(r_m / \lambda\right)}_{\text{Tail of the TLP}}$$

$$+ \underbrace{\boxed{\delta_{(\lambda\text{-bit CSD})}}}_{\text{FLP}} + \underbrace{\boxed{\delta_{(\text{XOR})}} \cdot (1 + \log_2 \lambda) + 3\boxed{\delta_{(\text{MUX})}}}_{\text{PSU and PSA}},$$

$$\delta = \max\left(\delta_{(\text{Stage } 1)}, \delta_{(\text{Stage } 2)}, \cdots, \delta_{(\text{Stage } m)}\right)$$

🟩 Constant parameters; Determined by microarch. & technology

(r, λ) ➔ Throughput

$$\phi_i = \begin{cases} \left(\prod_{j=1}^{m} r_j\right)/\lambda, & \text{if } i = m \\ \prod_{j=1}^{i} r_j, & \text{otherwise.} \end{cases}$$

$$\phi = \phi_m + \sum_{i=1}^{m-1} \phi_i/2.$$

$$K/(\phi \cdot (\delta + \delta_{(\text{setup})})) \text{ bits per second}$$

You can find the details of the derivations from our paper:

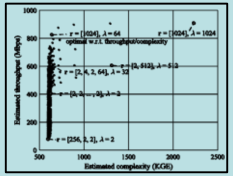
H.-Y. Yoon and T.-H. Kim, “Generalized Tree Architecture for Efficient Successive-Cancellation Polar Decoding,” *ICCD 2018* (to Appear)

Arch. Synthesis for SC Decoders

Constant Parameters:
N, *Q*, *δ*, *C*

Design Objective/Constraint (Throughput, Latency, Complexity, …)

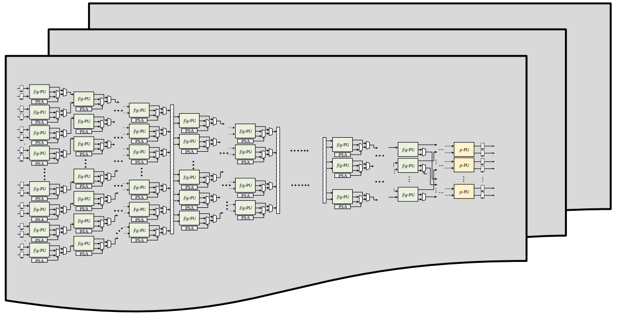
Design Space Explorer



Architecture Synthesizer

- Automatic HDL Generation (Datapath + Controller)

SC Decoder Synthesizer



SC Decoder as a Soft-Core RTL (in Verilog HDL),
Expected Throughput,
Expected Complexity

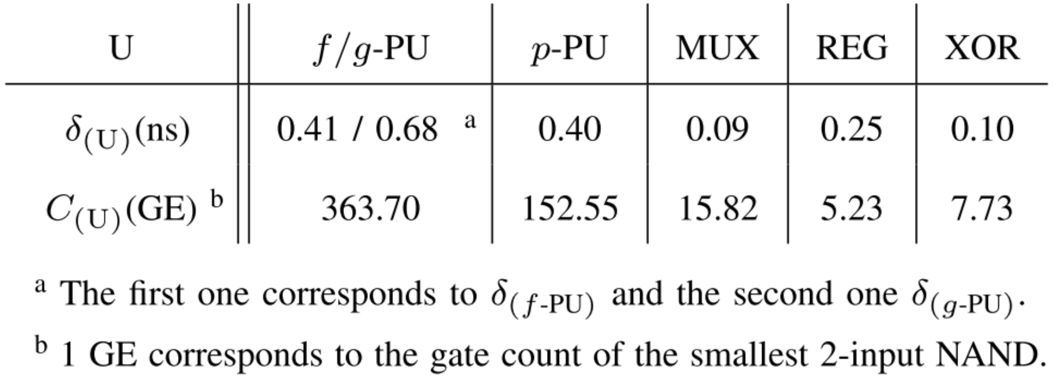
"Fully Automated"

Case Study: 1K-bit SC Decoder

- Technology: 0.18μm CMOS
- Design objective in the case study
  - Maximize throughput/complexity
    - FOM has been defined accordingly.

Case Study: 1K-bit SC Decoder

- Constant parameters in the target tech.
  - δ, C



U	f/g-PU	p-PU	MUX	REG	XOR
$\delta_{(\mathrm{U})}$(ns)	0.41 / 0.68 [a]	0.40	0.09	0.25	0.10
$C_{(\mathrm{U})}$(GE) [b]	363.70	152.55	15.82	5.23	7.73

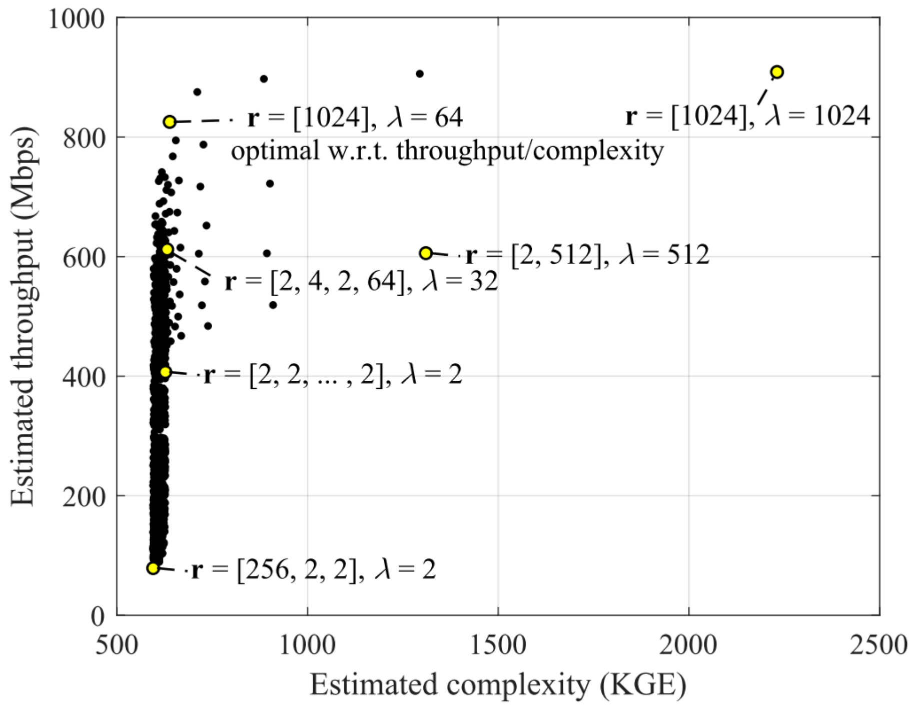
[a] The first one corresponds to $\delta_{(f\text{-PU})}$ and the second one $\delta_{(g\text{-PU})}$.

[b] 1 GE corresponds to the gate count of the smallest 2-input NAND.

  - LLR size (Q) has been set to 6.

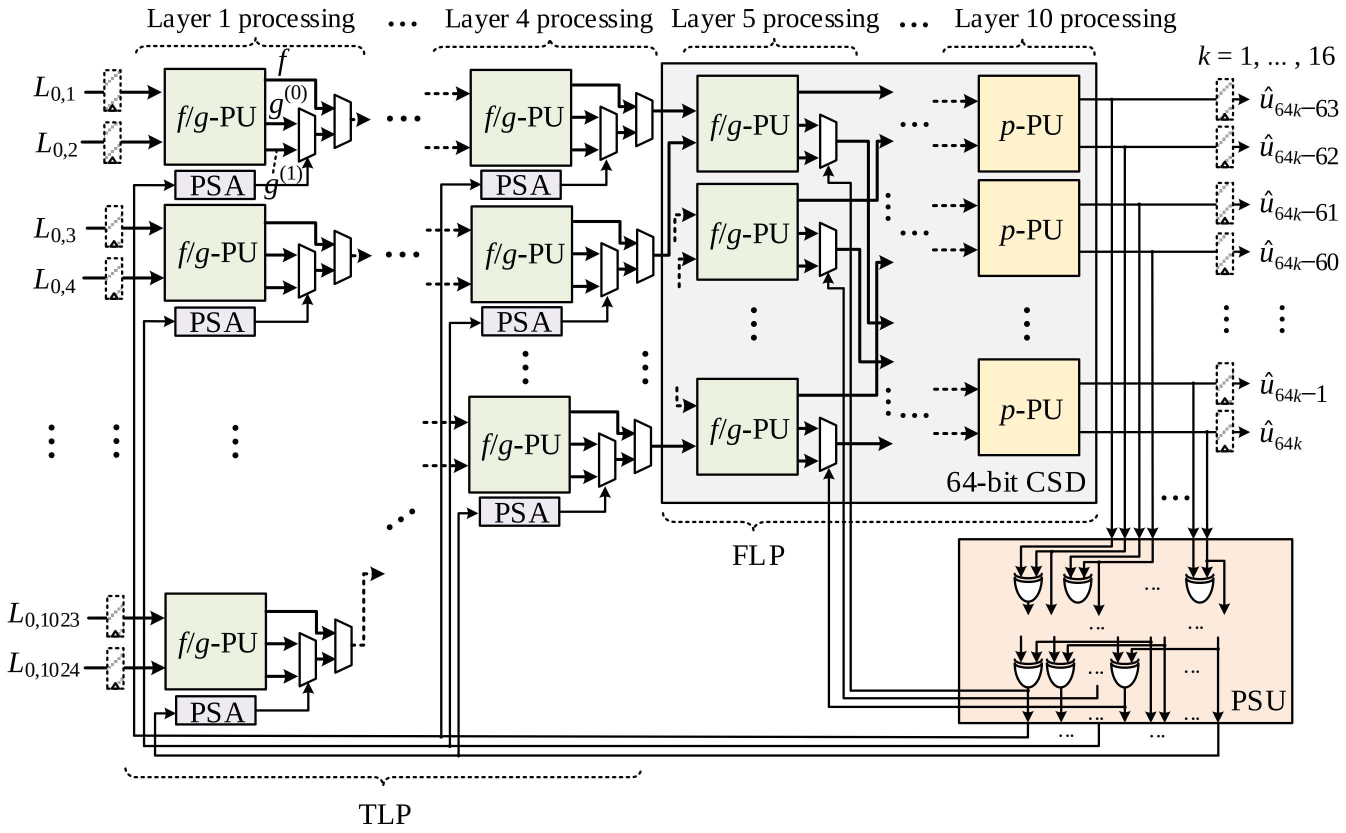
Case Study: 1K-bit SC Decoder

- Design Space



Case Study: 1K-bit SC Decoder

r = [1024], λ = 64



Comparison with Prev. Results

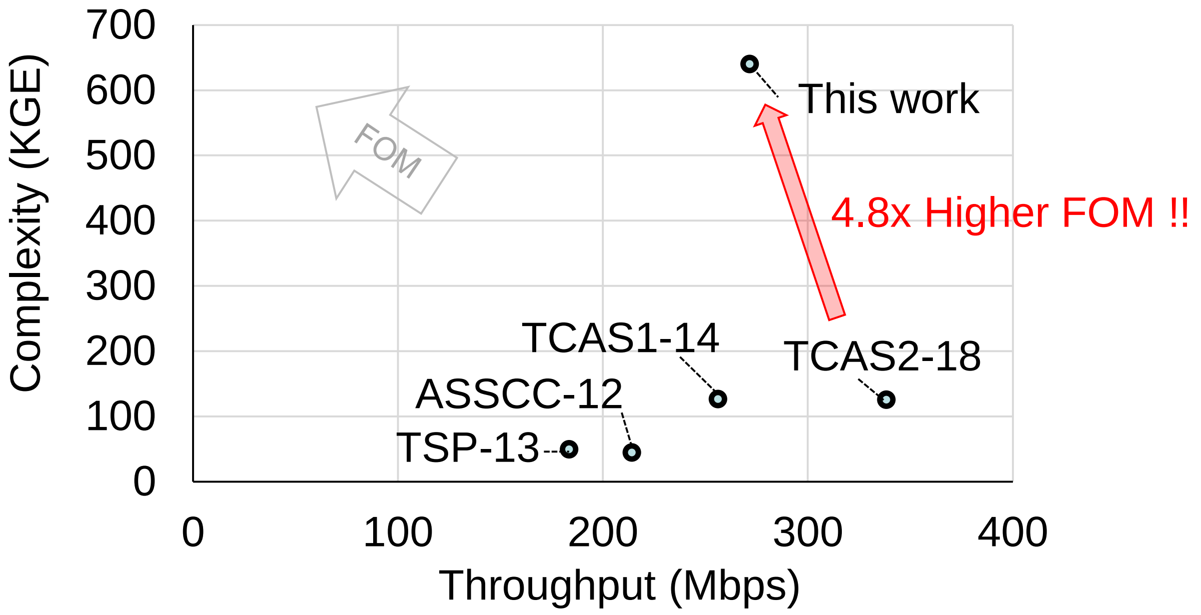
Work	ASSCC-12 [*]	TSP-13 [**]	TCAS1-14 [***]	TCAS2-18 [****]	This work
Architecture	Semi-parallel	Semi-parallel	Tree (λ = 2)	Tree (λ = 2)	Gen. Tree (**r** = [1024], λ = 64)
CMOS tech.	180nm	65nm	45nm	180nm	180nm
Operating freq.	150MHz	500MHz	750MHz	377MHz	20MHz
Norm. throughput	49Mbps	44Mbps	125Mbps	126Mbps	640Mbps
Complexity	183.6KGE	214.3KGE	338.4KGE	256.3KGE	271.8KGE
FOM	0.26Kbps/GE	0.20Kbps/GE	0.36Kbps/GE	0.48Kbps/GE	2.35Kbps/GE

All the decoders in the table show the same BER performance (1K-bit SC).
All the decoders in the table show constant throughput irrespective of the rate.
All the results in the table (except the 1st one) were obtained for the pre-layout design.

[*] A. Mishra et al., “A successive cancellation decoder ASIC for a 1024-bit polar code in 180nm CMOS,” *ASSCC*, Nov. 2012, pp. 205–208.
[**] C. Leroux, A. J. Raymond, G. Sarkis, and W. J. Gross, “A semi-parallel successive-cancellation decoder for polar codes,” *IEEE TSP*, vol. 61, no. 2, pp. 289–299, Jan. 2013.
[***] B. Yuan and K. K. Parhi, “Low-latency successive-cancellation polar decoder architectures using 2-bit decoding,” *IEEE TCAS I*, vol. 61, no. 4, pp. 1241–1254, Apr. 2014.
[****] H-.Y. Yoon and T.-H. Kim, “Efficient Successive-Cancellation Polar Decoder Based on Redundant LLR Representation,” *IEEE TCAS II*, to be published.

Flash Memory Summit

Comparison with Prev. Results



Conclusion

- Generalized tree arch. considering high-radix kernel proc. for SC polar decoders
- Architecture synthesis for SC polar decoders
- Case study to validate the idea
- First step to design a high-performance polar decoder
  - Applicable to implement other adv. polar decoders.

Thank you !!

taehwan.kim@kau.ac.kr

Appendix: Implementation Results

Design params.	**r** = [2, 2, … 2], λ = 2 (Conv.)	**r** = [1024], λ = 64 (Optimized)
$N_{(f/g\text{-PU})}$	1022	1120
$N_{(p\text{-PU})}$	1	32
$N_{(\text{MUX})}$	13286	13440
$N_{(\text{REG})}$	7154	960
$N_{(\text{XOR})}$	1023	1152
Latency (cycles)	1023	16
C (est. complexity) (KGE)	627	638.77
Est. throughput (Mbps)	406.9	825.38
Est. FOM (KBps/GE)	0.64 (1.0)	1.29 (2.0)
Real complexity (KGE)	212	271
Real throughput (Mbps)	250	640
Real FOM (KBps/GE)	1.17 (1.0)	2.36 (2.0)