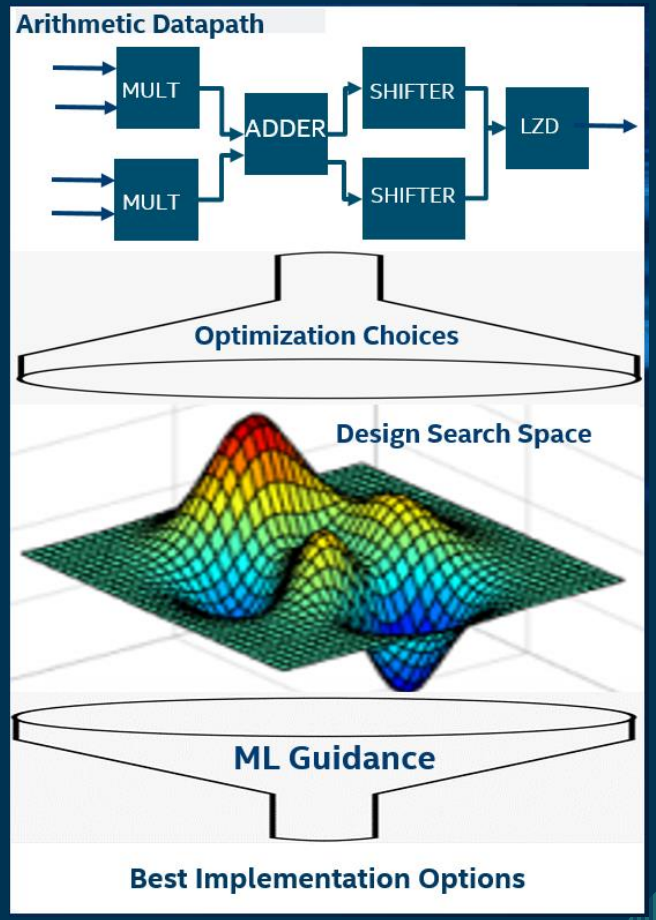
Arithmetic datapath design exploration with Machine Learning

Mahesh Gupta Vutukuri (mahesh.g.Vutukuri@intel.com)
Sreekanth Madgula (Sreekanth.Madgula@intel.com)
Ashutosh Garg (Ashutosh.garg@intel.com)

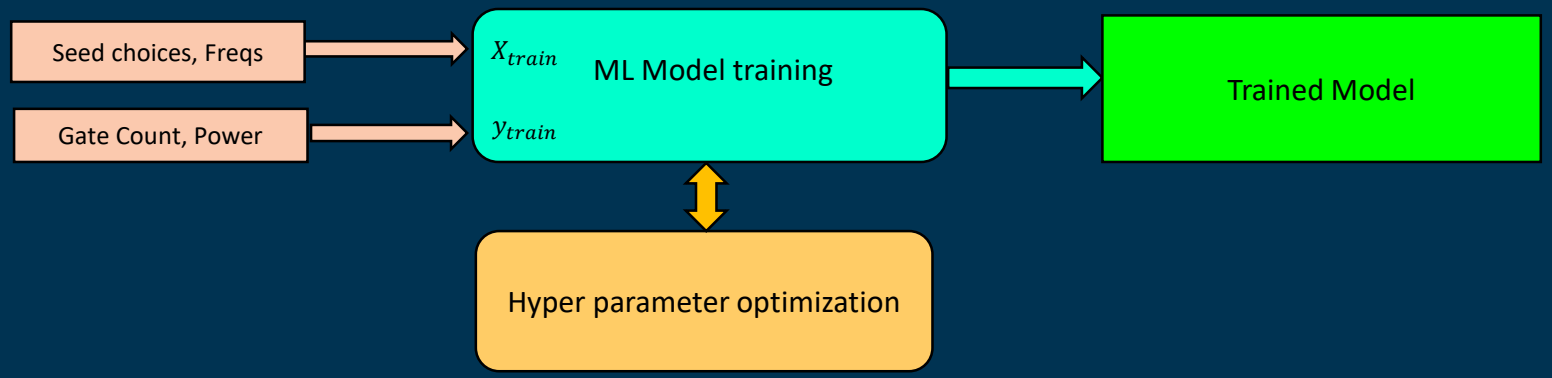
Intel Corporation

Motivation

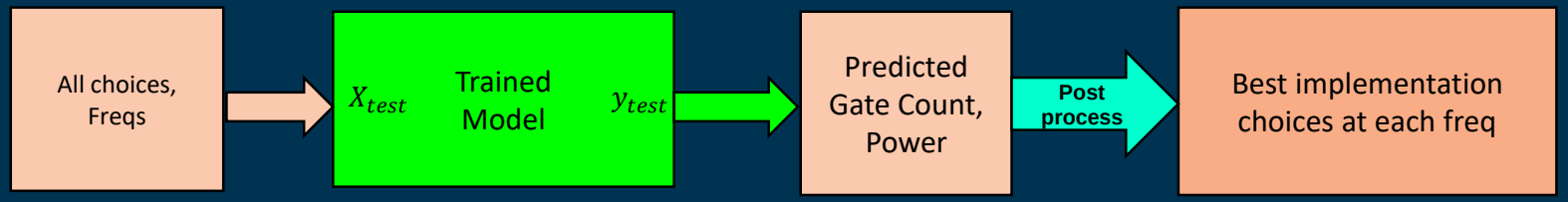
- All Hardware accelerators implement large arithmetic datapaths
- Large design search space to explore for optimization
  - 20 independent binary choices means 1 million options !!!
- Machine Learning (ML) techniques for intelligent guidance assist in selecting a few high-quality choices for implementation
- Proof of Concept on a small arithmetic datapath is explored
- Best Choice by Model agrees with Human choices, shows up some new points as well



ML Based Exploration - Overview



- Write Seed RTLs, Synthesize to get gate count, power
- Define input and outputs for a supervised ML regression algorithm
- Do training !!!



- Use trained model to predict gate count, power for all possible RTLs
- Rank and use best choice as your guidance

ML Training : Input Seed Data Preparation

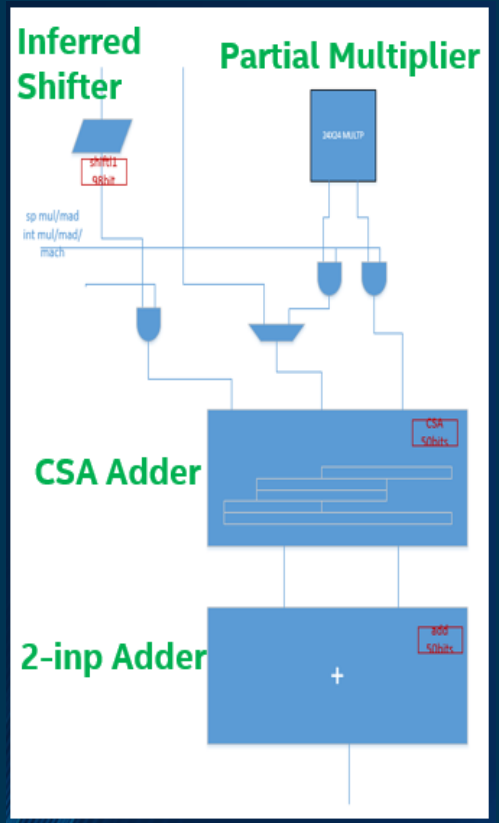
Component	Implementation	Encoding	RTL1	RTL2	RTL3	RTL4	RTL5	RTL6
Mult+Add	full mult w/ 3 input addition	0	3	0	1	3	1	3
	full mult w/2 input addition	1						
	partial mult w/3 input addition	2						
	partial mult w/csa+2input addition	3						
Shifter	inferred	0	0	0	0	1	0	0
	Instantiated	1						
LZD	top_hierarchy	0	0	0	0	0	1	1
	current_hierarchy	1						
Pipelining	Manual	0	0	0	0	0	1	1
	Auto	1						

	Input Features (Xn)					Output Features (Y)	
Training Set	Freq	Mult+adder*	Shifter	LZD	Pipeline	GateCount(K)	Power (uW)
	Freq_1	3	0	0	0	322	67.3
	Freq_1	0	0	0	0	320	68.1
	Freq_1	1	0	0	0	314	67.1
	...	...	....	...	...	...	...
	Freq_11	3	1	0	0	403	108.9
	Freq_11	1	0	1	1	464	134.4
	Freq_11	3	0	1	1	428	120.4

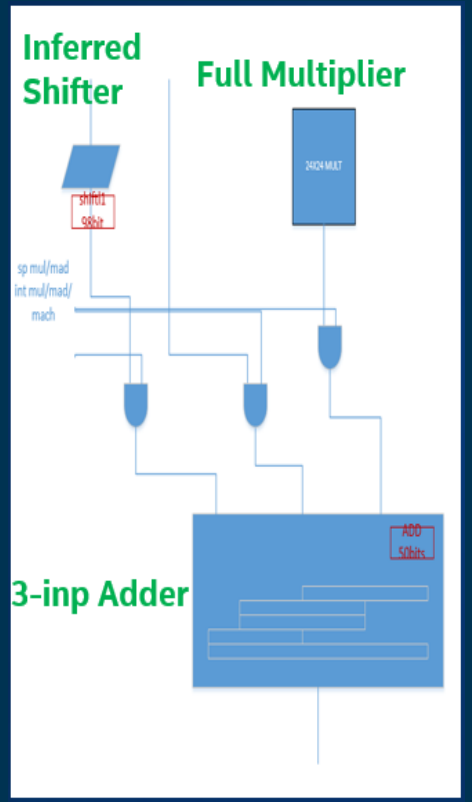
- 6 Seed RTLs are written and encoded -: Input Features of ML Model
- Synthesized at 11 different frequencies -: Input Samples of ML Model
- Syn Gate Count, Cdyn :- Target features of ML Model

* Mult+Adder combination is not independent since different multiplier choices will impact corresponding adder choices.
** Gate count and power numbers obfuscated for IP reasons but represent the trend.

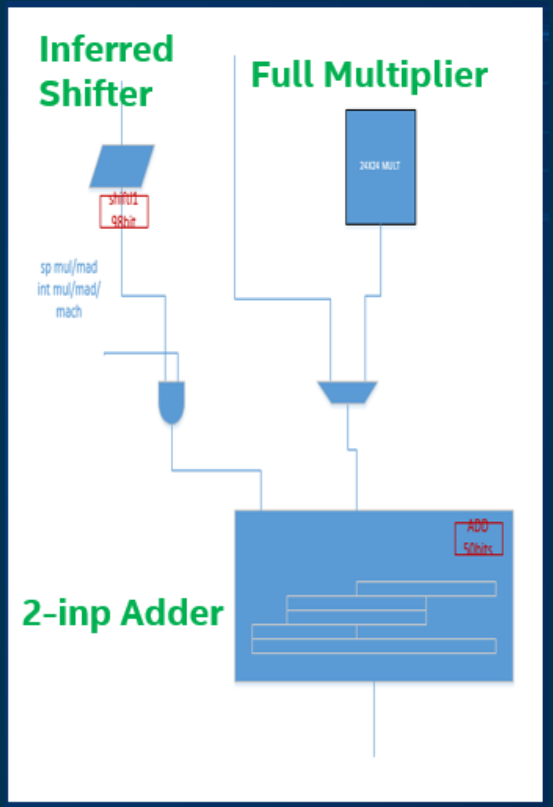
Seed RTL examples



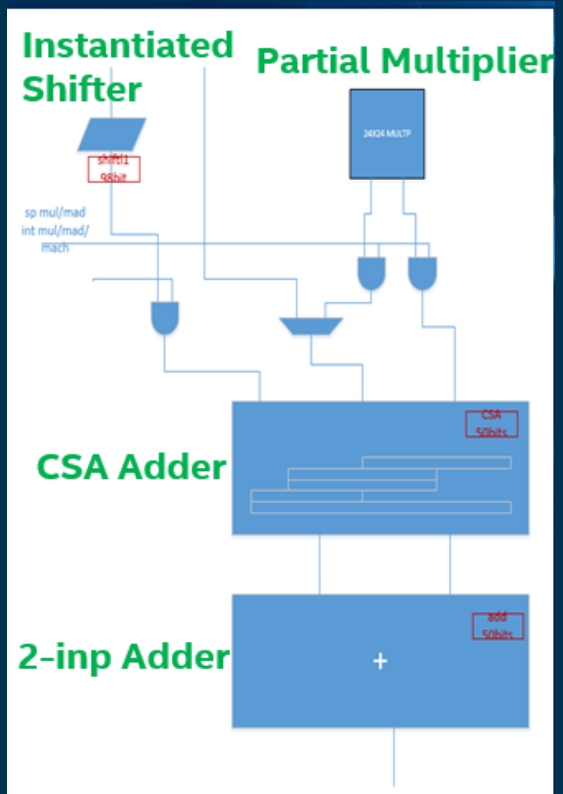
RTL1: Partial multiplier + CSA and 2-input adder



RTL2: Full multiplier and 3-input adder



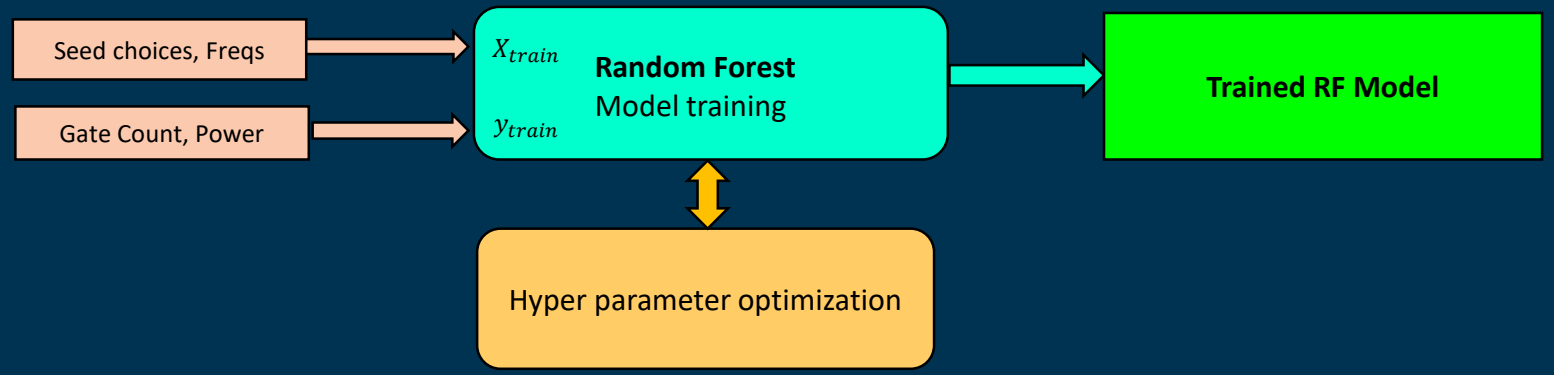
RTL3: Full multiplier and 2-input adder



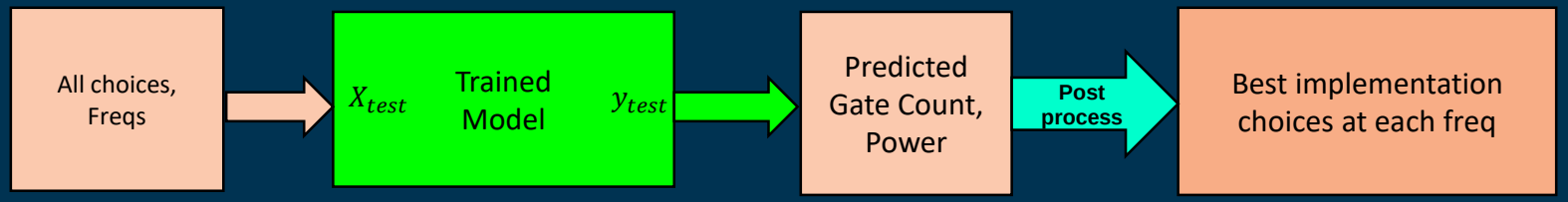
RTL4: Same as RTL1 + instantiated shifter

ML Training

- Posed ML Algorithm selection along with Hyper parameter tuning as Model search space problem
- Obtained Best Model using Group Kfold Cross-validation MAPE
  - Random Forest Model is the winner
- Trained RF Model with all input training data



ML Inference



	X_test				
RTL_ID	Freq	mult_adder	shifter	lzd	pipeline
RTL1	freq1	3	0	0	0
RTL1	freq2	3	0	0	0
RTL1	freq3	3	0	0	0
...	...	...	...	...	...
RTL1	freq_11	3	0	0	0
...	...	...	...	...	...
RTL4	freq1	3	1	0	0
RTL4	freq2	3	1	0	0
...		...	...	...	...
RTL4	freq_11	3	1	0	0
...	...	...	...	...	...
RTL7	freq1	0	0	0	1
RTL7	freq2	0	0	0	1
...	...	...	...	...	...
RTL7	freq11	0	0	0	1
...	...	...	...	...	...
RTL20	freq1	2	0	0	0
RTL20	freq2	2	0	0	0
...	...	...	...	...	...
RTL20	freq11	2	0	0	0
...	...	...	...	...	...
RTL32	freq1	3	1	1	1
RTL32	freq2	3	1	1	1
...	...	...	...	...	...
RTL32	freq11	3	1	1	1

Y_test
Power(uW)
60.0
60.1
60.1
84.7
60.8
61.2
78.9
64.2
64.6
95.3
59.9
59.8
87.3
64.4
63.2
87.9

Post process

Frequency	RTL_Power_GC_Sorted
Low freq range	[[3, 20], [1]]
	[[3, 20], [1]]
	[[3, 20], [1]]
	[[3, 20], [1]]
	[[3, 20], [1]]
	[[1], [3, 20]]
	[[1], [3, 20]]
High freq range	[[4], [16, 24]]
	[[4], [16, 24]]
	[[1], [4]]
	[[1], [4]]
	[[1], [4]]
	[[31], [4]]
	[[4], [16, 24]]

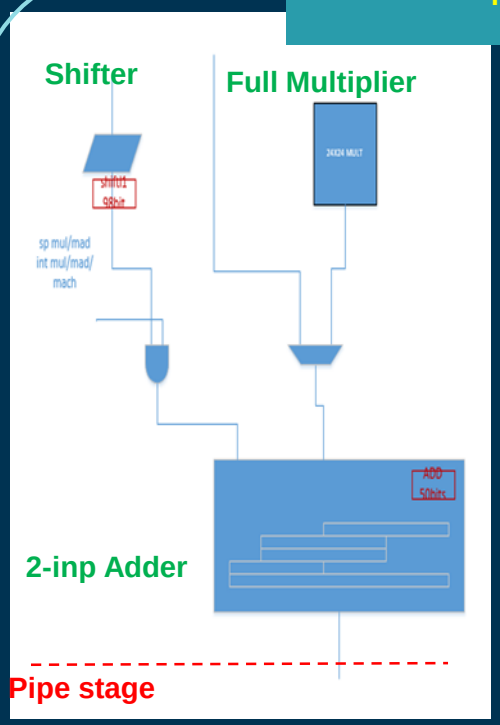
Analysis of Results

Frequency	RTL_Power_GC_Sorted
Low freq range	[[3, 20], [1]]
	[[3, 20], [1]]
	[[3, 20], [1]]
	[[3, 20], [1]]
	[[3, 20], [1]]
	[[1], [3, 20]]
	[[1], [3, 20]]
High freq range	[[4], [16, 24]]
	[[4], [16, 24]]
	[[1], [4]]
	[[1], [4]]
	[[1], [4]]
	[[31], [4]]
	[[4], [16, 24]]

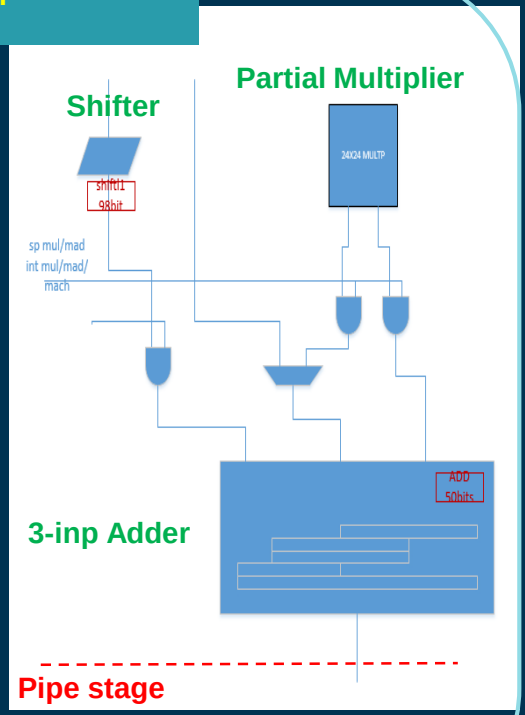
- RTL[3,20]@ low freq, RTL4 at high freq, RTL1 for wide range of frequencies
- New RTL20 is predicted to be better at low frequencies
- Proof of pudding is in eating
  - Wrote RTL20 and Synthesized
  - Compared gate count, power predicted vs realized
  - Gate count error : 2.1%
  - Power error : 5.8%

Analysis of Results .. contd

Low Frequencies

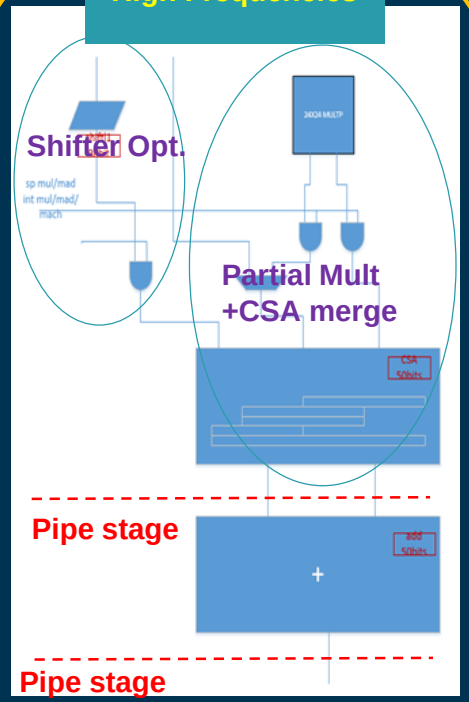


RTL3: Full multiplier and 2-input adder



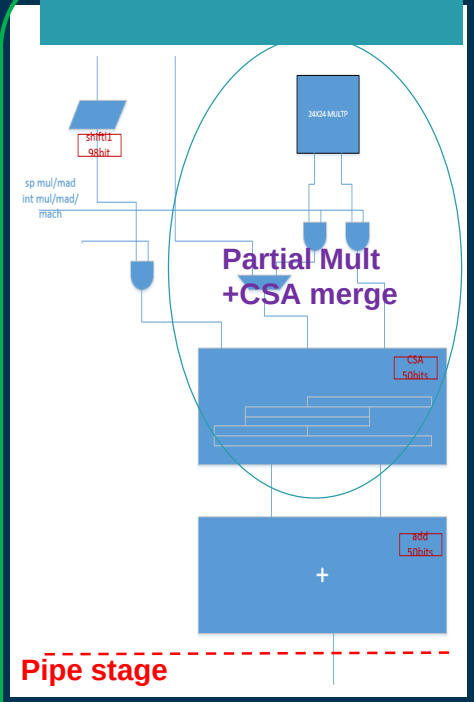
RTL20: Partial multiplier and 3-input adder

High Frequencies



RTL4: Same as RTL1 + instantiated shifter

Wide Range of Frequencies



RTL1: Partial multiplier + CSA and 3-input adder

Summary & Next steps

ML method is a “quick prototyping tool” for datapath design which accelerated design space exploration during early design cycle

ML was leveraged to guide component selection for optimization

- ✓ Evaluated multiple u-arch choices at a fraction of the effort
- ✓ Saved valuable designer time focusing on a few best choices

Many new choice predictions were too close to seed data

- ✓ Indicated limited set of input samples in training data, bigger scope problem will not have this problem
- ✓ Component level data to provide additional granularity

PoC work is presented for datapath design exploration

- ✓ Use case of arithmetic datapath lends well to ML problem formulation
- ✓ Further work is needed in applying to other datapaths and control logic

Bibliography

[1] Hung-Yi Liu and L. P. Carloni, "On learning-based methods for design-space exploration with High-Level Synthesis," 2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC), Austin, TX, 2013, pp. 1-7.

[2] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011

[3] https://github.com/hyperopt/hyperopt