NANDA 2022

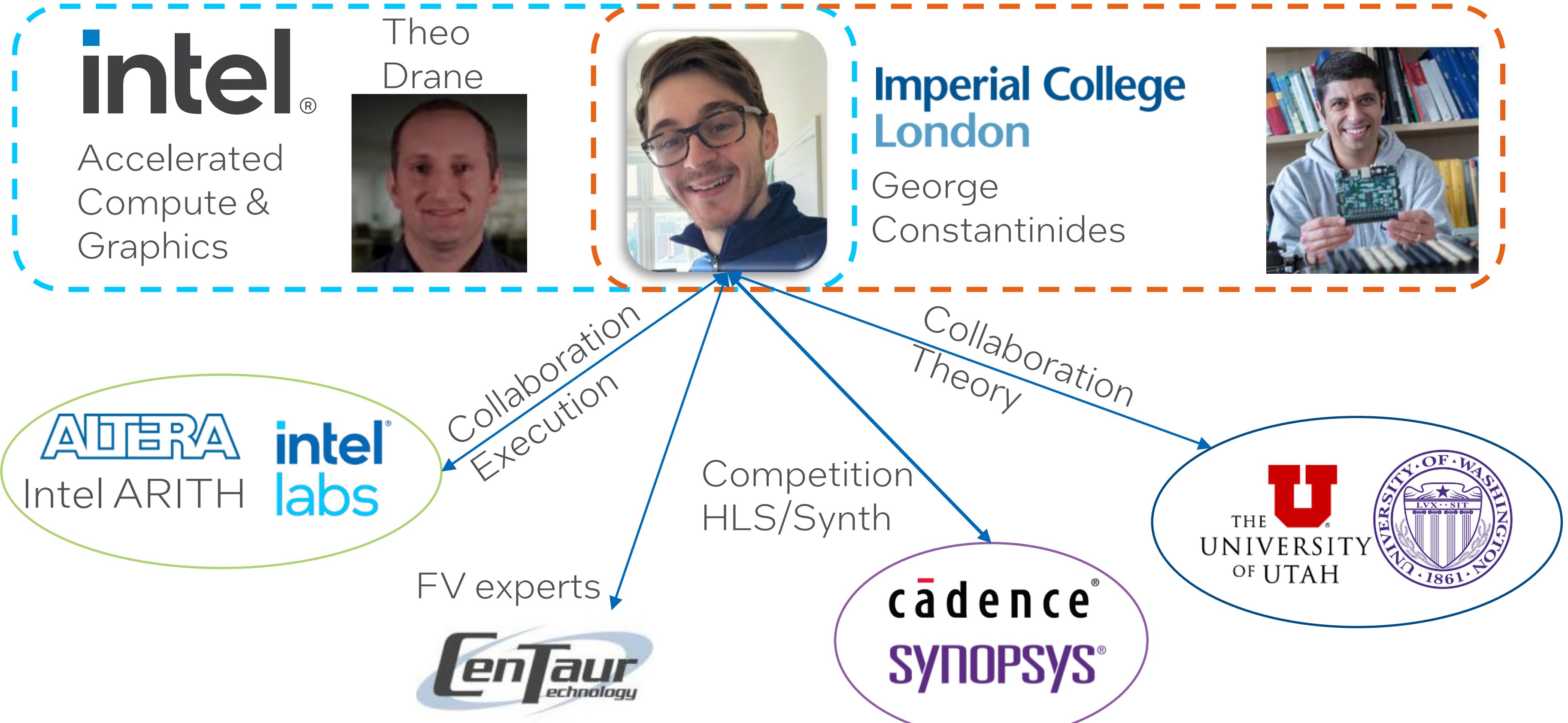
ROVER: RTL Optimisation via Verified E-Graph Rewriting

Samuel Coward — Intel GFx & Imperial College

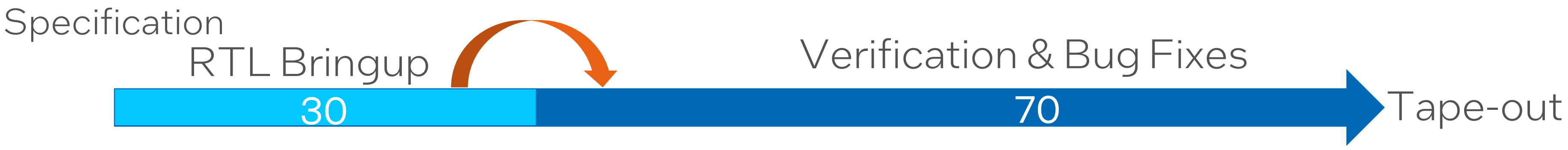
George Constantinides — Imperial College

Theo Drane — Intel GFx

Industrial PhD – A Novel Design



Hardware Development Timeline



High Level Synth

Logic Synth

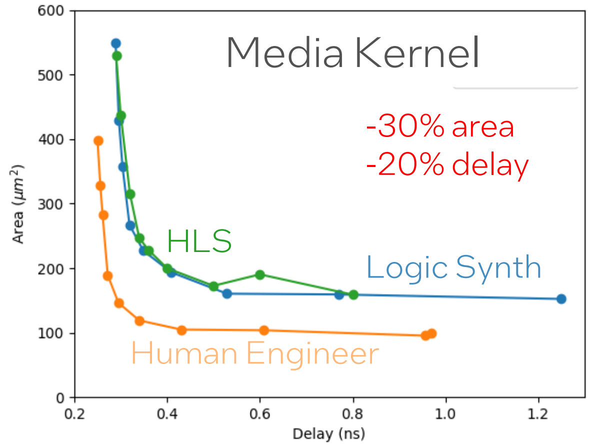
Equivalence Checking

Simulation

RTL Engineer

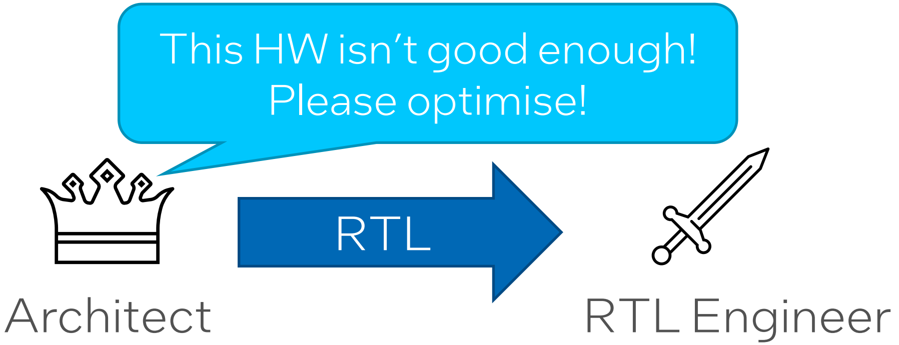
$100k min

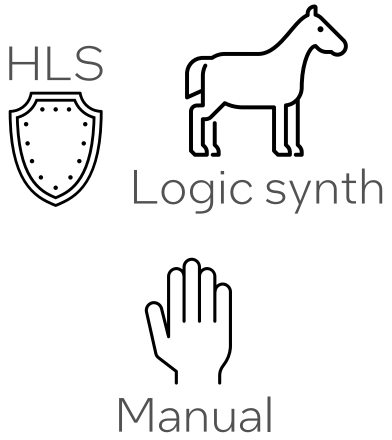
- Achieve correctness & fix bugs
- Select & implement an arch
- Fill gap left by tools

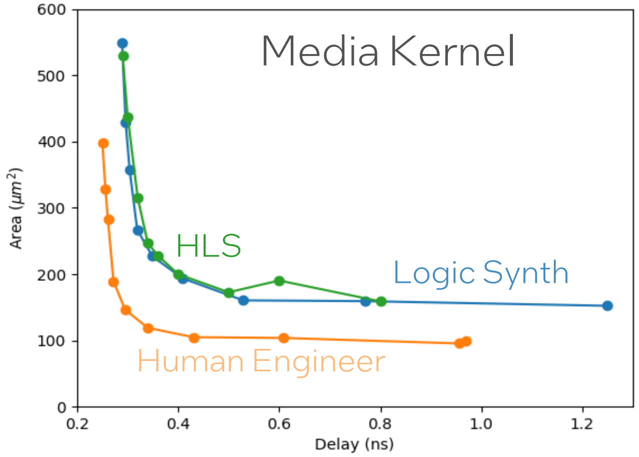


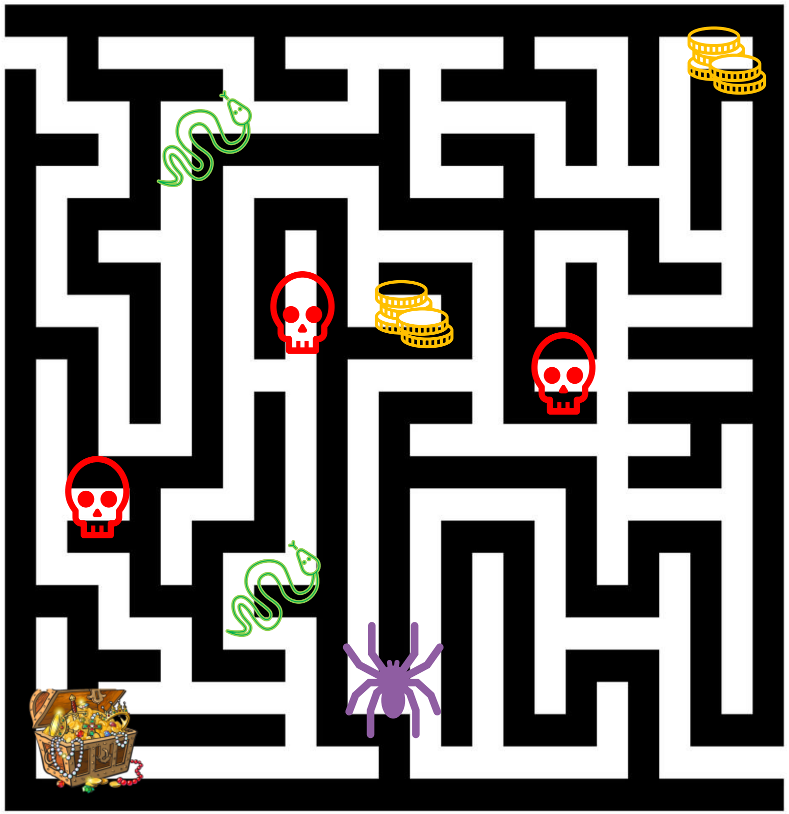
Exploring the Maze

Architectural Design Space



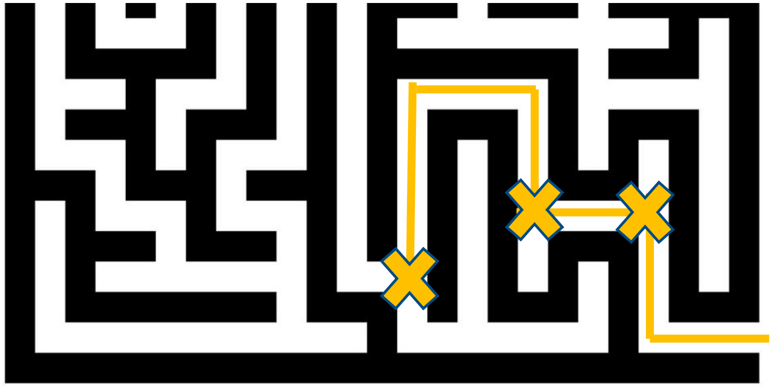


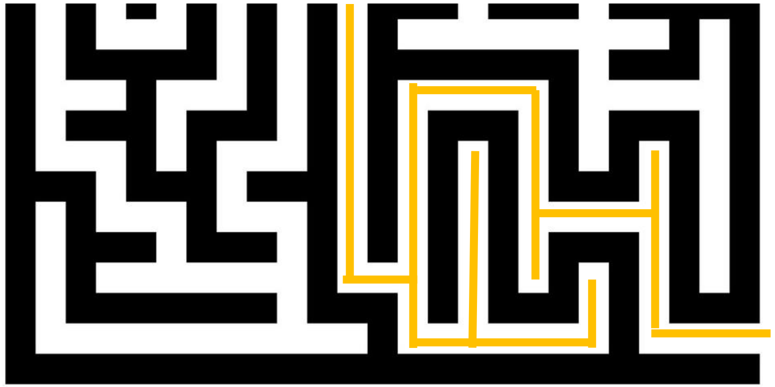




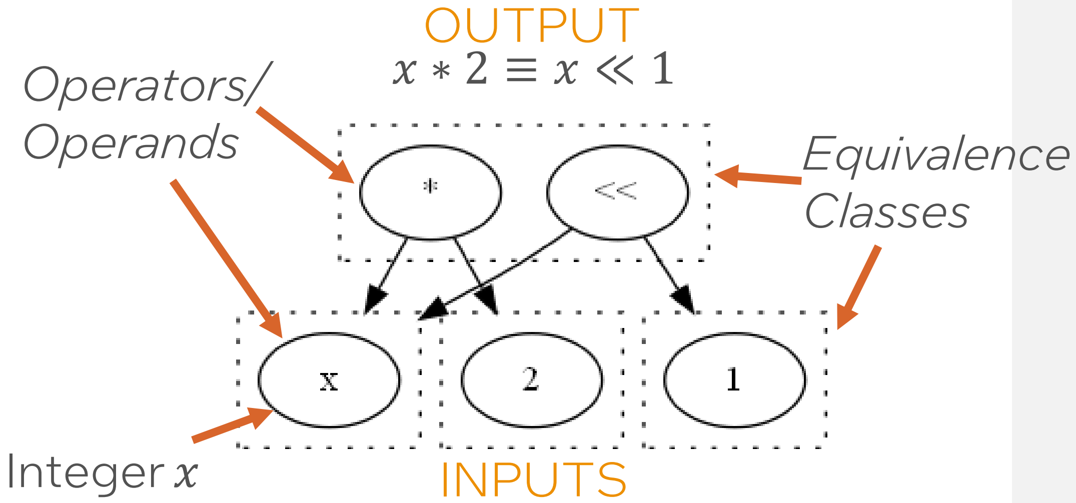
E-Graphs

egg: e-graphs good

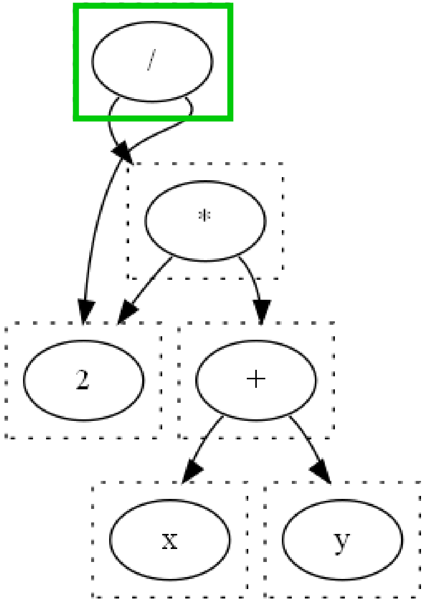




- Data structure – efficient rewrite engines
- Compact representation of equivalent designs
- Maintain history
- Enable efficient design space exploration
- Constructive rewrite application – phase ordering

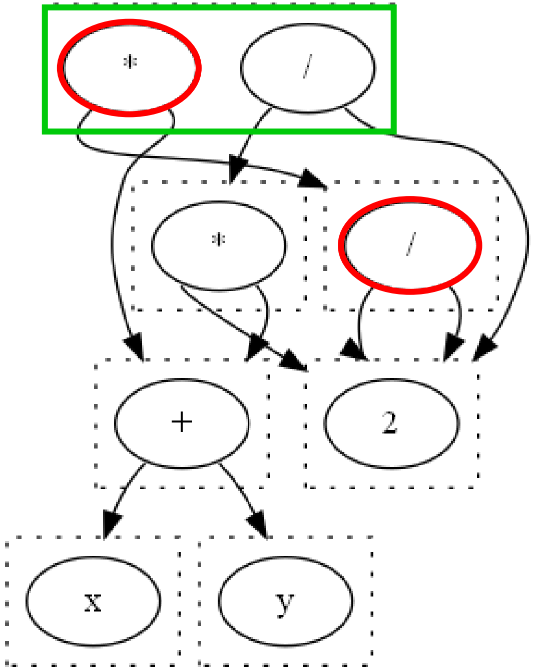


Rewriting the E-graph



$$\frac{(2*(x+y))}{2}$$

$$\frac{(a*b)}{c} \rightarrow \frac{a}{c}*b$$

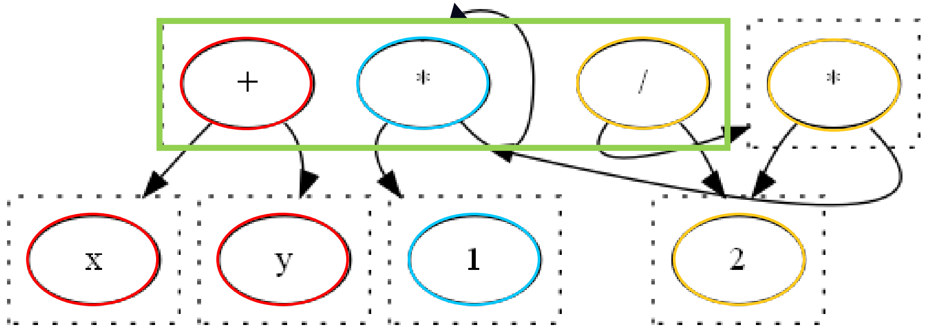


$\frac{(2*(x+y))}{2}$ and $\frac{2}{2}*(x+y)$

Applying further rewrites

$\frac{2}{2} * (x + y)$ $\xrightarrow{\frac{a}{a} \to 1}$ $1 * (x + y)$ $\xrightarrow{1 * a \to a}$ $(x + y)$

Final E-graph – choose "best" from equivalent designs?



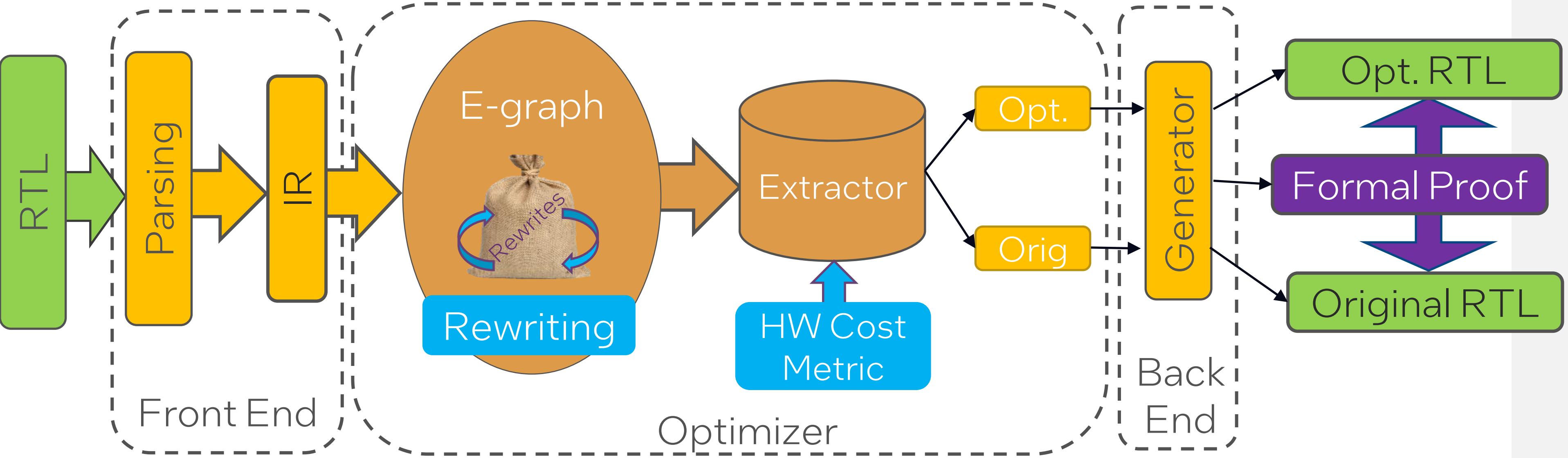
- $(x + y)$
- $\frac{(2*(x+y))}{2}$
- $1 * (x + y), 1 * 1 * (x + y), \dots$

Contains full history and infinitely many equivalent designs

ROVER: Automating Datapath Optimisation

Goals: Increase productivity & increase IP quality

Target: Numerical ASIC Hardware Optimisation



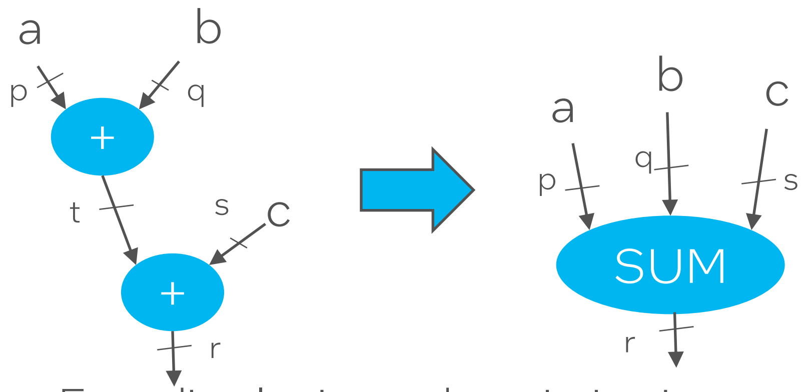
- Combinational RTL
- Operations on unsigned fixed width bitvectors

Supported Operators:
Logical - $\gg, \ll, ?, \{, \}, >, <, \sim$
Arithmetic - $+, -, \times$

Rewrites – Parameterizable & Conditional

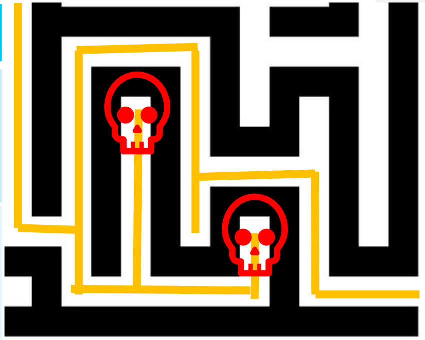
28 rewrites

- arithmetic identities
- logic identities
- tool correlation
- constant expansion
- arithmetic logic exchange



Encoding logic synth optimisations

Rewrite	Sufficient Condition
$\left(\left(a_p + b_q\right)_s \ll c_t\right)_r \rightarrow \left(\left(a_p \ll c_t\right)_r + \left(b_q \ll c_t\right)_r\right)_r$	$r \leq s \vee \max(p,q) < s$
$\left(\left(a_p \ll b_q\right)_u \ll c_s\right)_r \rightarrow \left(a_p \ll \left(b_q + c_s\right)_t\right)_r$	$t > \max(q,s) \wedge u \geq r$

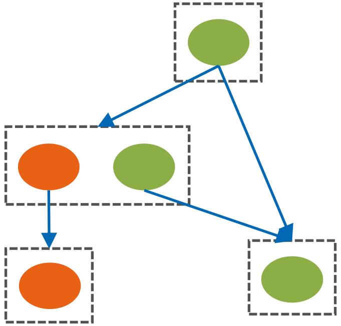


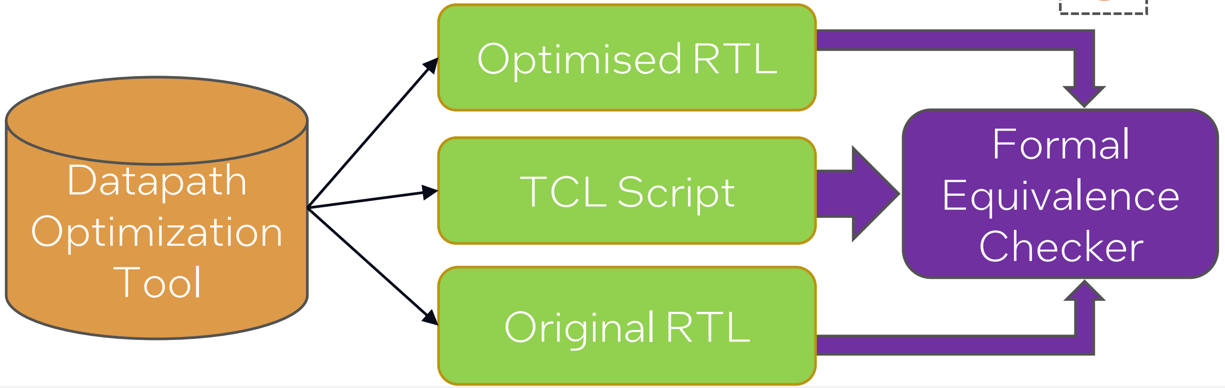
Extraction & Verification

- Theoretical 2 input gate count cost metric – area only
- Bitwidth dependent operator cost
- Integer Linear Programming (common sub-expressions)

$$\text{minimize: } \sum_{n \in \mathcal{N}} \text{cost}(n) x_n \text{ subject to:}$$

$$x_{root} = 1 \quad \text{and} \quad \forall (n, c) \in E.\, x_n \le \sum_{n' \in \mathcal{N}_c} x_{n'}$$





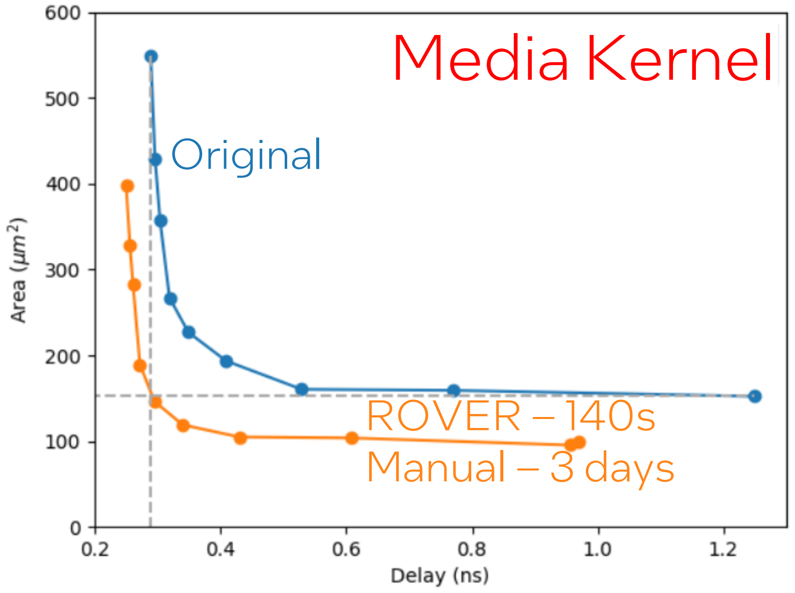
- Increased confidence
- No need to trust rewrites
- Detects broken rewrites

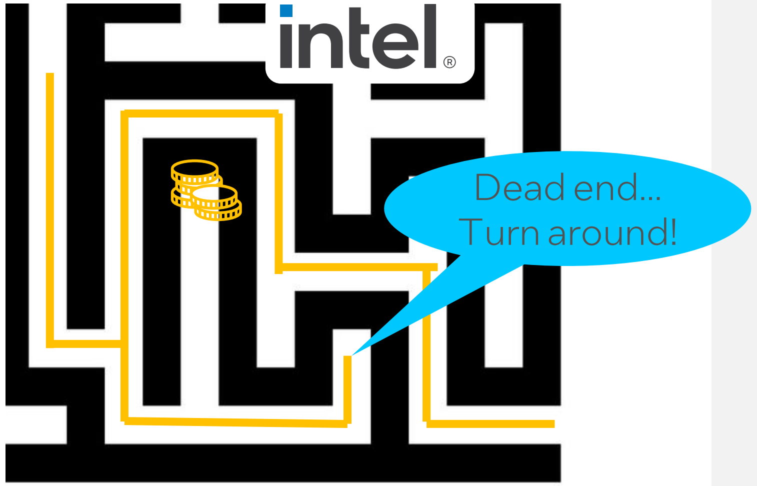
Results

Existing Intel interpolation RTL

Automatically finds optimal architecture

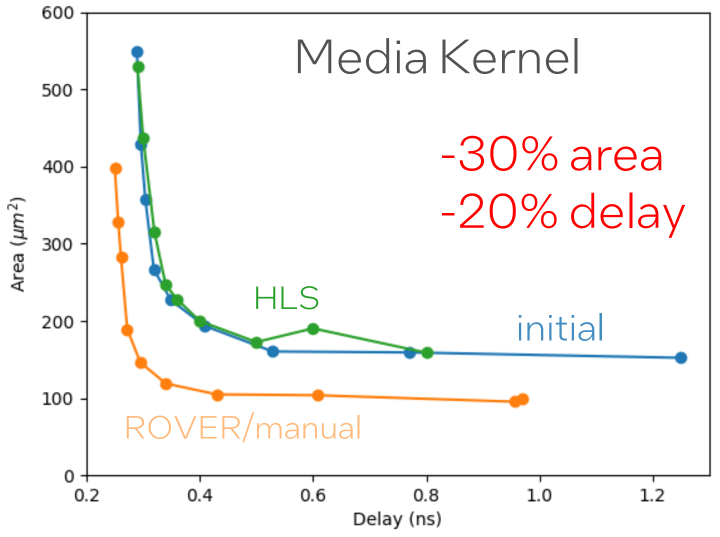
Matches manual optimization





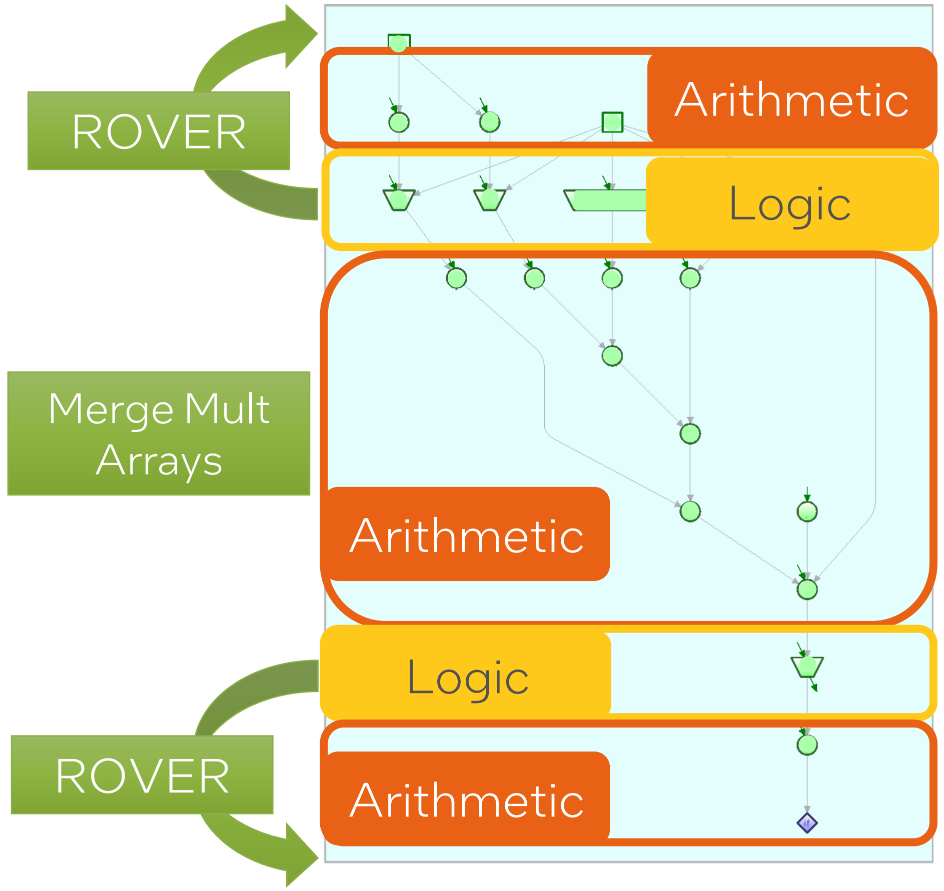
Design	Area Change	Runtime (sec)
FIR Filter	-60%	100
ADPCM Decoder	-1%	19
Shifted FMA	-32%	<1
MCM	+15%	31

HLS Comparison



Stratus – bit level optimisations
- didn't cross architectural boundary

ROVER – clustered arithmetic ops
– moved logic out the way



Multiple Constant Multiplication

Optimal circuit to generate:

$$\{a_1 \times x, a_2 \times x, \ldots, a_n \times x\}$$

integer const integer variable

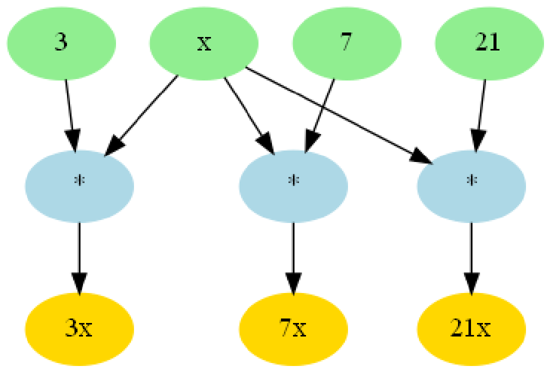
Matches operator count from [1]

Rewrites:

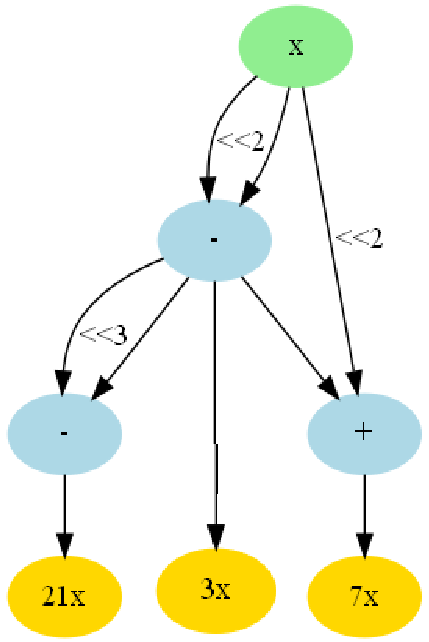
$$1 \rightarrow 2 - 1$$
$$c \rightarrow 2 \times (c \gg 1) + c[0]$$

$$7 = 2 \times 3 + 1 = 2 \times (2 + 1) + 1 = 4 + 3$$

Naive



Optimised

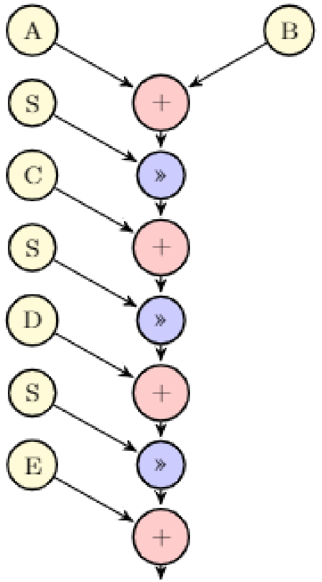


[1] - N. M. Sarband, O. Gustafsson and M. Garrido, "Obtaining Minimum Depth Sum of Products from Multiple Constant Multiplication," *2018*

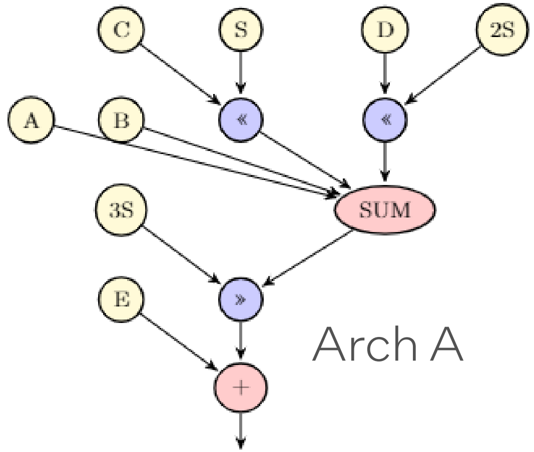
Parameterisable RTL – FIR Kernel

Easy-to-use but do we sacrifice quality?

Yes – optimal RTL design varies with parameter values

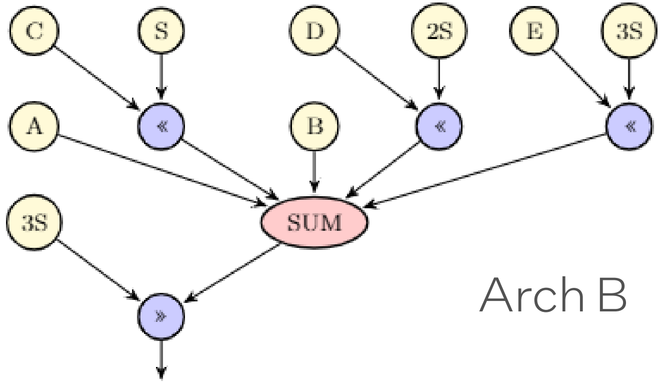


Original



Arch A

Optimal for bitwidths
8 → 28



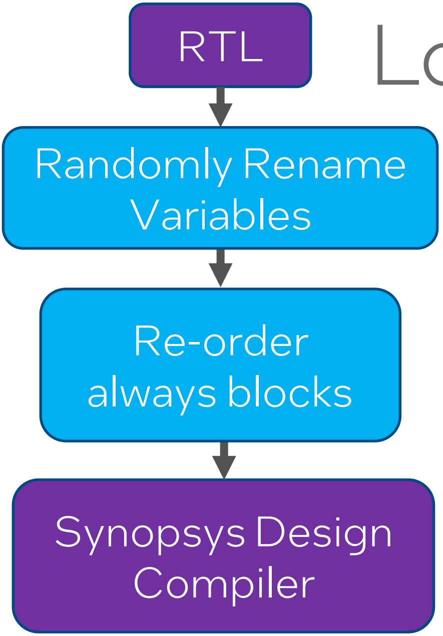
Arch B

Optimal for bitwidths
32 → 56

Cost Metric Validation

- Agreed with logic synthesis on 56% of parameterizable testcases
- Limited by lack of delay model
- Does logic synthesis generate predictable results?

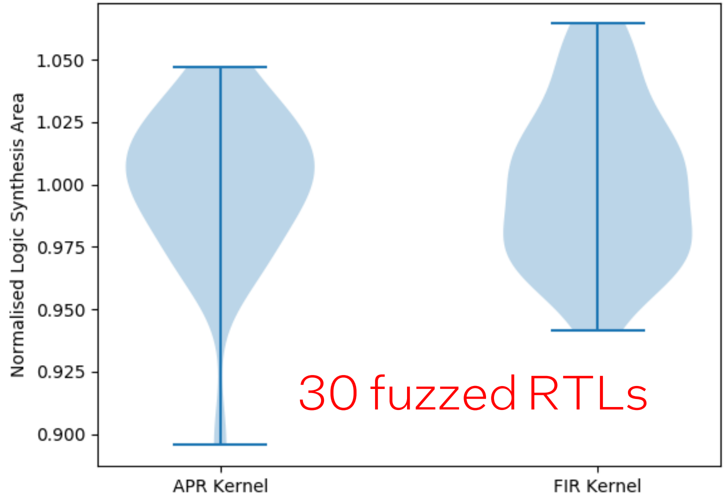
Logic Synthesis Fuzzing



Semantically preserving mutations

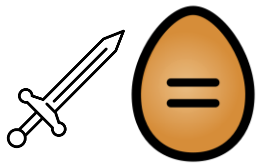
Expect identical results

Up to 15% difference…



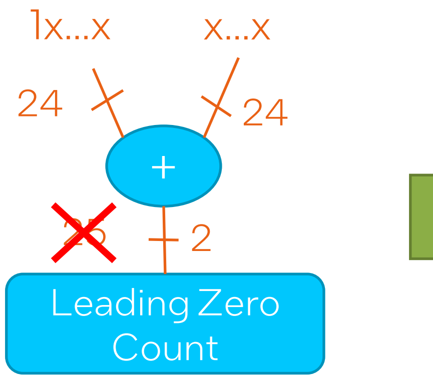
Conclusions

- Applied E-Graphs to datapath optimisation
- Matched human designer on Intel testcase
- Verified RTL generated



RTL Engineer

But, are syntactic rewrites enough?



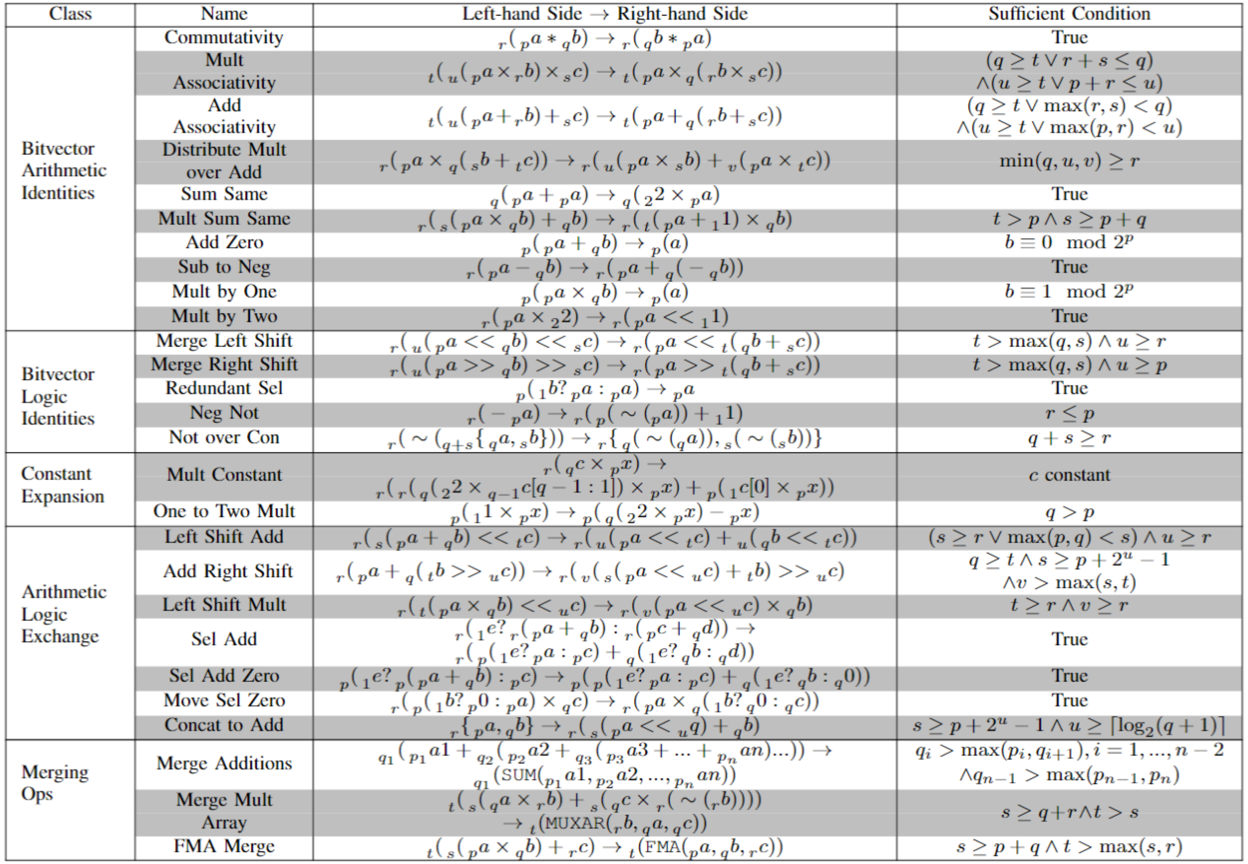
Where is the leading 1?

Solution?

- Intermediate Value Analysis
- Domain Specific Rewrites

Combining E-Graphs with Abstract Interpretation

Rewrite Table



Class	Name	Left-hand Side $\rightarrow$ Right-hand Side	Sufficient Condition
Bitvector Arithmetic Identities	Commutativity	${}_r({}_p a * {}_q b) \rightarrow {}_r({}_q b * {}_p a)$	True
	Mult Associativity	${}_t({}_u({}_p a \times {}_r b) \times {}_s c) \rightarrow {}_t({}_p a \times {}_q({}_r b \times {}_s c))$	$(q \geq t \vee r + s \leq q)$ $\wedge(u \geq t \vee p + r \leq u)$
	Add Associativity	${}_t({}_u({}_p a + {}_r b) + {}_s c) \rightarrow {}_t({}_p a + {}_q({}_r b + {}_s c))$	$(q \geq t \vee \max(r, s) < q)$ $\wedge(u \geq t \vee \max(p, r) < u)$
	Distribute Mult over Add	${}_r({}_p a \times {}_q({}_s b + {}_t c)) \rightarrow {}_r({}_u({}_p a \times {}_s b) + {}_v({}_p a \times {}_t c))$	$\min(q, u, v) \geq r$
	Sum Same	${}_q({}_p a + {}_p a) \rightarrow {}_q({}_2 2 \times {}_p a)$	True
	Mult Sum Same	${}_r({}_s({}_p a \times {}_q b) + {}_q b) \rightarrow {}_r({}_t({}_p a + {}_1 1) \times {}_q b)$	$t > p \wedge s \geq p + q$
	Add Zero	${}_p({}_p a + {}_q b) \rightarrow {}_p(a)$	$b \equiv 0 \mod 2^p$
	Sub to Neg	${}_r({}_p a - {}_q b) \rightarrow {}_r({}_p a + {}_q(-{}_q b))$	True
	Mult by One	${}_p({}_p a \times {}_q b) \rightarrow {}_p(a)$	$b \equiv 1 \mod 2^p$
	Mult by Two	${}_r({}_p a \times {}_2 2) \rightarrow {}_r({}_p a << {}_1 1)$	True
Bitvector Logic Identities	Merge Left Shift	${}_r({}_u({}_p a << {}_q b) << {}_s c) \rightarrow {}_r({}_p a << {}_t({}_q b + {}_s c))$	$t > \max(q, s) \wedge u \geq r$
	Merge Right Shift	${}_r({}_u({}_p a >> {}_q b) >> {}_s c) \rightarrow {}_r({}_p a >> {}_t({}_q b + {}_s c))$	$t > \max(q, s) \wedge u \geq p$
	Redundant Sel	${}_p({}_1 b ? {}_p a : {}_p a) \rightarrow {}_p a$	True
	Neg Not	${}_r(-{}_p a) \rightarrow {}_r({}_p(\sim({}_p a)) + {}_1 1)$	$r \leq p$
	Not over Con	${}_r(\sim({}_{q+s}\{{}_q a, {}_s b\})) \rightarrow {}_r\{{}_q(\sim({}_q a)), {}_s(\sim({}_s b))\}$	$q + s \geq r$
Constant Expansion	Mult Constant	${}_r({}_q c \times {}_p x) \rightarrow$ ${}_r({}_r({}_q({}_2 2 \times {}_{q-1} c[q-1:1]) \times {}_p x) + {}_p({}_1 c[0] \times {}_p x))$	c constant
	One to Two Mult	${}_p({}_1 1 \times {}_p x) \rightarrow {}_p({}_q({}_2 2 \times {}_p x) - {}_p x)$	$q > p$
Arithmetic Logic Exchange	Left Shift Add	${}_r({}_s({}_p a + {}_q b) << {}_t c) \rightarrow {}_r({}_u({}_p a << {}_t c) + {}_u({}_q b << {}_t c))$	$(s \geq r \vee \max(p, q) < s) \wedge u \geq r$
	Add Right Shift	${}_r({}_p a + {}_q({}_t b >> {}_u c)) \rightarrow {}_r({}_v({}_s({}_p a << {}_u c) + {}_t b) >> {}_u c)$	$q \geq t \wedge s \geq p + 2^u - 1$ $\wedge v > \max(s, t)$
	Left Shift Mult	${}_r({}_t({}_p a \times {}_q b) << {}_u c) \rightarrow {}_r({}_v({}_p a << {}_u c) \times {}_q b)$	$t \geq r \wedge v \geq r$
	Sel Add	${}_r({}_1 e ? {}_r({}_p a + {}_q b) : {}_r({}_p c + {}_q d)) \rightarrow$ ${}_r({}_p({}_1 e ? {}_p a : {}_p c) + {}_q({}_1 e ? {}_q b : {}_q d))$	True
	Sel Add Zero	${}_p({}_1 e ? {}_p({}_p a + {}_q b) : {}_p c) \rightarrow {}_p({}_p({}_1 e ? {}_p a : {}_p c) + {}_q({}_1 e ? {}_q b : {}_q 0))$	True
	Move Sel Zero	${}_r({}_p({}_1 b ? {}_p 0 : {}_p a) \times {}_q c) \rightarrow {}_r({}_p a \times {}_q({}_1 b ? {}_q 0 : {}_q c))$	True
	Concat to Add	${}_r\{{}_p a, {}_q b\} \rightarrow {}_r({}_s({}_p a << {}_u q) + {}_q b)$	$s \geq p + 2^u - 1 \wedge u \geq \lceil \log_2(q+1) \rceil$
Merging Ops	Merge Additions	${}_{q_1}({}_{p_1} a1 + {}_{q_2}({}_{p_2} a2 + {}_{q_3}({}_{p_3} a3 + \ldots + {}_{p_n} an)\ldots)) \rightarrow$ ${}_{q_1}(\texttt{SUM}({}_{p_1} a1, {}_{p_2} a2, \ldots, {}_{p_n} an))$	$q_i > \max(p_i, q_{i+1}), i = 1, \ldots, n-2$ $\wedge q_{n-1} > \max(p_{n-1}, p_n)$
	Merge Mult Array	${}_t({}_s({}_q a \times {}_r b) + {}_s({}_q c \times {}_r(\sim({}_r b))))$ $\rightarrow {}_t(\texttt{MUXAR}({}_r b, {}_q a, {}_q c))$	$s \geq q + r \wedge t > s$
	FMA Merge	${}_t({}_s({}_p a \times {}_q b) + {}_r c) \rightarrow {}_t(\texttt{FMA}({}_p a, {}_q b, {}_r c))$	$s \geq p + q \wedge t > \max(s, r)$