Simplification of C-RTL Equivalent Checking for Fused Multiply Add Unit using Intermediate Models

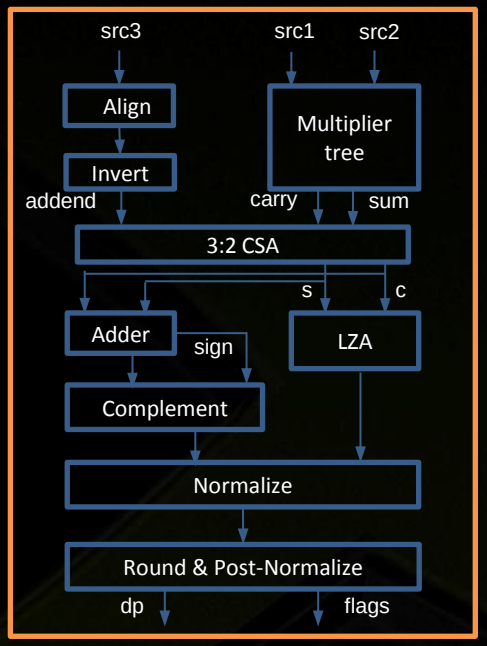
Bin Xue (bxue@nvidia.com)
Prosenjit Chatterjee (PChatterjee@nvidia.com)
Sandeep K. Shukla (shukla@vt.edu)

Outline

- Introduction
- Overview of the verification strategy
- C-RTL SEC of FMA instructions with intermediate models
- SEC between the abstract RTL model and the original model
- SEC between the C and the rewritten C model
- Experiment results
- Conclusion

Introduction

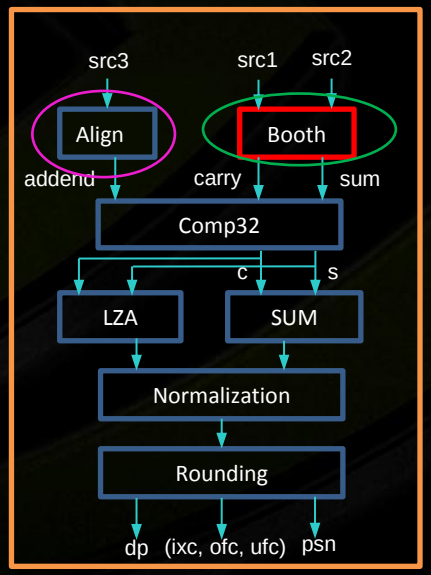
- FMA performs ± src1 × src2 ± src3 in a single instruction



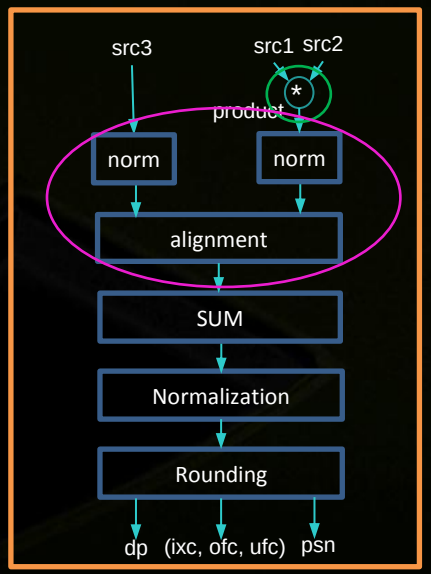
- C-RTL sequential equivalence checking (SEC)
- C-RTL SEC for FMA is a difficult problem

Gaps between C and RTL model for FMA

- Gap 1: multiplier Implementation RTL(booth algorithm) vs C(*);
- Gap 2: normalization and alignment sequence;
- Other common gaps: bit-level vs word level, parallel vs sequential, data width limit.



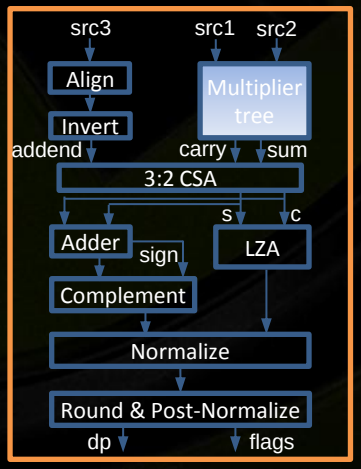
Original RTL model: R0



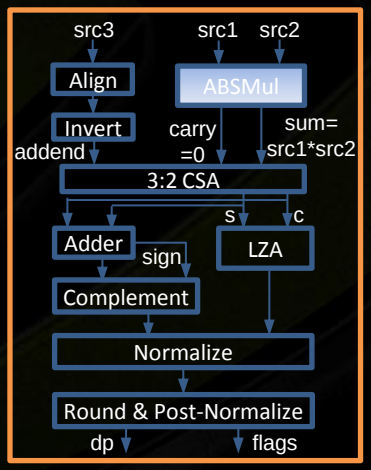
Original C model:C0

Overview of the verification strategy

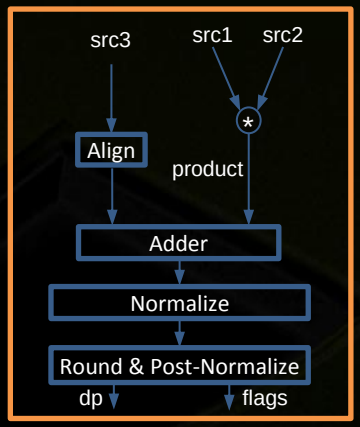
- Use intermediate models to bridge the gaps;
- For gap 1: replace the booth multiplier by ABSMul(*), leading to R1;
- For gap 2: rewrite C0 to get C1 which uses the same alignment and normalization sequence as the RTL;



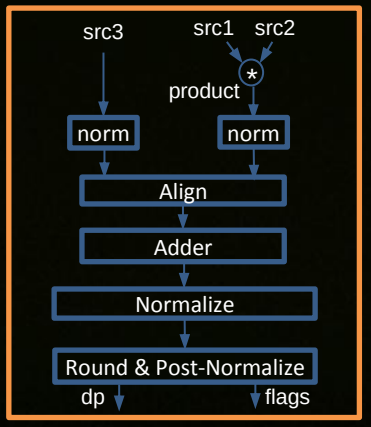
Original RTL model: R0



Abstract RTLmodel: R1

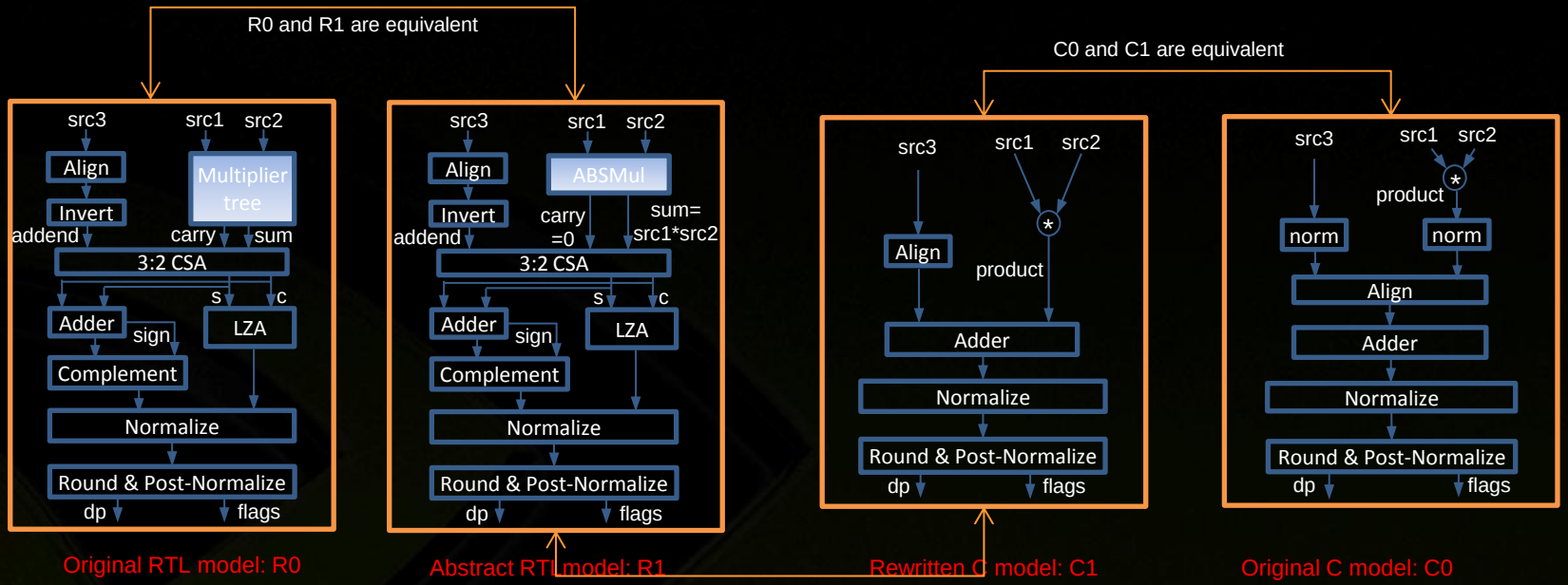


Rewritten C model: C1



Original C model: C0

Overview of the verification strategy



- **Prove FMA instructions between C1 and R1 or C0 and R1;**
- **Prove R0 and R1 are equivalent;**
- **Prove C0 and C1 are equivalent;**

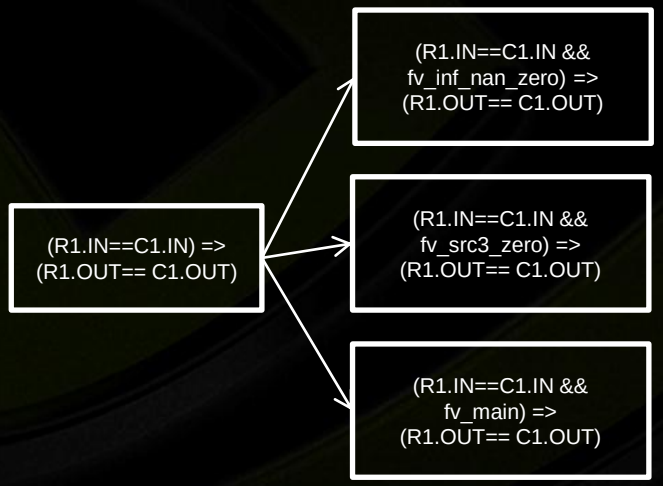
C-RTL SEC of FMA instructions between C1 and R1

- **Example: VFMADDPD**
  - dp[63:0] = src1[63:0] * src2[63:0]+src3[63:0]
  - dp[127:64] = src1[127:64] * src2[127:64]+src3[127:64]

- **Verification techniques**
  - Case splitting on src1, src2 and src3
  - assume guarantee proof on internal match points
    - flags
    - stages

Case splitting on src1, src2 and src3

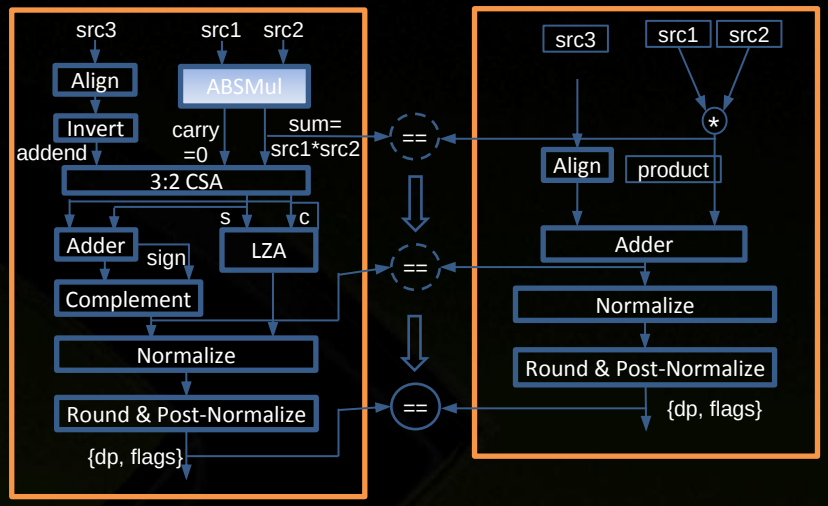
- inf_nan_zero: one of src1, src2, src3 is infinite or NAN (not a number), or one of src1, src2 is zero.
- src3_zero: src3 is zero
- main: the rest case



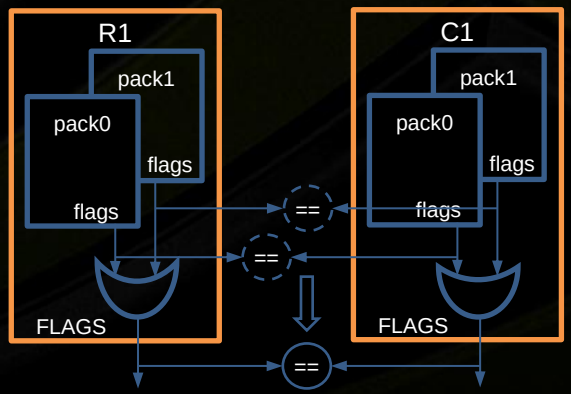

```
assign fv_inf_nan_zero =
        ((src1[62:52] == 11'h7ff) ||
         (src2[62:52] == 11'h7ff) ||
         (src3[62:52] == 11'h7ff)) ||
         (src2[62:0] == 63'h0) ||
         (src1[62:0] == 63'h0));

assign fv_src3_zero =
        !fv_inf_nan_zero &&
        (src3[62:0] == 63'h0)

assign fv_main =
        !(fv_inf_nan_zero ||
          fv_src3_zero)
```

Assume guarantee on internal match points

- Assume guarantee on internal stages



- Assume guarantee on flags



SEC between R1 and R0

- Step 1: verify the implementation of booth multiplier against ABSMul;
- Step 2: SEC between R1 with ABSMul and R0 with booth multiplier;

Verification of implementation of Booth

- **Relation to be verified**
  - (Booth.sum + Booth.carry)[105:0] ==

  (ABSMul.sum + ABSMul.carry)[105:0] == src1[52:0] * src2[52:0]

- **Partition the relation into four sub relations**

1) $(\text{Booth.sum} + \text{Booth.carry})[105:0] == (\sum_0^{26} \text{pp}_k \ll 2k)[105:0]$

2) $(\sum_0^{26} \text{pp}_k \ll 2k)[105:0] == (\sum_0^{26}(\$\text{signed}(\text{src1}[52:0] * \text{B}_k) \ll 2k)[105:0]$

3) $(\sum_0^{26}(\$\text{signed}(\text{src1}[52:0]) * \text{B}_k) \ll 2k)[105:0] ==$
$(\text{src1}[52:0] * \sum_0^{26}(\$\text{signed}(\text{B}_k) \ll 2k)[105:0]$

4) $(\text{src1}[52:0] * \sum_0^{26}(\$\text{signed}(\text{B}_k) \ll 2k)[105:0] == \text{src1}[52:0] * \text{src2}[52:0]$

where B_k are booth encoding of src2;

Prove equivalence between R0 and R1

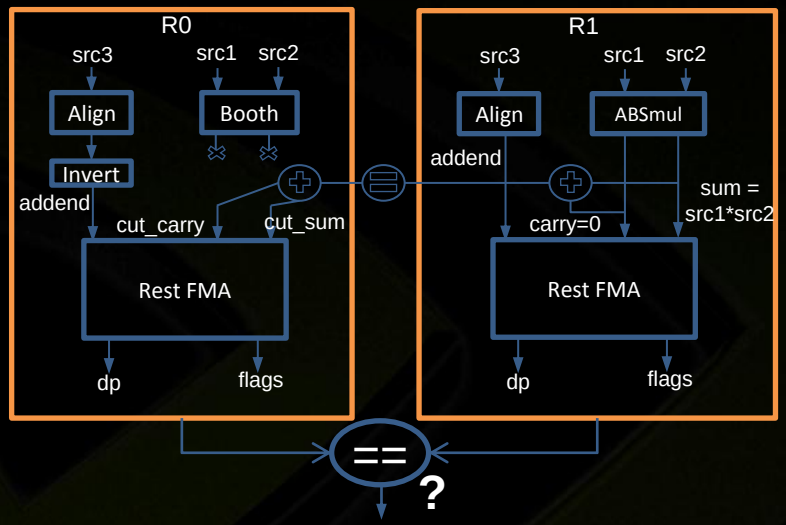
- **Booth multiplier is not equivalent to ABSMul, since**
  - Booth.sum [105:0] != ABSMul.sum [105:0]
  - Booth.carry[105:0] != ABSMul.carry[105:0]

- **We still need to prove**
  - if ((Booth.sum + Booth.carry)[105:0] == (ABSMul.sum + ABSMul.carry)[105:0])), then
  - {R0.dp[63:0], flags[2:0] == R1.dp[63:0], flags[2:0]}

- **However, Booth of R0 is too big for the proof to converge**

Prove equivalence between R0 and R1

- **Cut Booth.sum and Booth.carry to make them fully random, using cut_carry and cut_sum**




```
assume ((cut_sum + cut_carry)[107:0] ==
(ABSmul53.sum + ABSmul53.carry)[107:0]);
```

- **We prove a stronger relation:**

  ((cut_sum + cut_carry)[107:0] == (ABSmul.sum + ABSmul.carry)[107:0])

  For any arbitrary pair of cut sum and cut carry, if they satisfy the above equation, feeding them to the rest of R0 gets the equivalent outputs compared to R1.

SEC between the C and the rewritten C model

- **case splitting on shift_amt**
  - First, prove shift_amt is within [r0, r1];
  - Second, split the proof of C0 == C1 into (r1−r0+1) sub proofs for data paths and flags, each subproof has an additional assumption “shift amt == k”, where k is within [r0, r1].

Experiment result

- **C0 is implemented by standard C;**
- **R0 is the fma of next generation GPU;**
  - support half (16 bits), single (32 bits), double (64 bits) and extended precision (80 bits);
  - 160 bits data path, allows 2 packed extended, 2 packed double, 4 single or 8 half precision operations to execute in the pipeline in parallel;
- **All tasks are running on linux farm with 4G memory for each machine;**
- **Hector* from synopsys is chosen as the SEC tool;**

Experiment result

FMA instr.	Func.	Prec.	Packed	Case Split.	A.G.flags	A.G.stages	Comp	#Asserts	#Tasks	Exec.(s)
fmaddsh	src1*src2+src3	half	no	yes	no	yes	R1-C0	4201	89	16370
fmsubsh	src1*src2-src3	half	no	yes	no	yes	R1-C0	4201	89	17658
fmaddss	src1*src2+src3	single	no	yes	no	yes	R1-C0	1989	56	38784
fmsubss	src1*src2-src3	single	no	yes	no	yes	R1-C0	1989	56	37745
fmaddsd	src1*src2+src3	double	no	yes	no	yes	R1-C1	380	34	3855
fmsubsd	src1*src2-src3	double	no	yes	no	yes	R1-C1	380	34	4731
fmaddse	src1*src2+src3	extended	no	yes	no	yes	R1-C1	380	31	9519
fmsubse	src1*src2-src3	extended	no	yes	no	yes	R1-C1	380	31	10019
fmaddph	src1*src2+src3	half	yes	yes	yes	yes	R1-C0	4764	112	20033
fmsubph	src1*src2-src3	half	yes	yes	yes	yes	R1-C0	4764	112	19572
fmaddps	src1*src2+src3	single	yes	yes	yes	yes	R1-C0	6301	77	36472
fmsubps	src1*src2-src3	single	yes	yes	yes	yes	R1-C0	6299	77	43730
fmaddpd	src1*src2+src3	double	yes	yes	yes	yes	R1-C1	518	32	4339
fmsubpd	src1*src2-src3	double	yes	yes	yes	yes	R1-C1	518	32	5644
fmaddpe	src1*src2+src3	extended	yes	yes	yes	yes	R1-C1	518	29	10237
fmsubpe	src1*src2-src3	extended	yes	yes	yes	yes	R1-C1	518	29	9283
fma2fma	NA	NA	NA	NA	NA	NA	NA	329	16	3756
c2c	NA	NA	NA	NA	NA	NA	NA	490	490	127538

Conclusion

- **Propose two intermediate models to reduce the complexity of C-RTL SEC for FMA;**
  - Use abstract RTL model R1 where booth multiplier is replaced by the same multiplier used in the original C model C0;
  - Use a rewritten C model C1, which has the same alignment as the original RTL R0;
  - Other techniques to reduce the complexity;

- **Experiment results from a real industry project demonstrates the efficiency of our strategy.**

Questions