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 $\pm \text{src1} \times \text{src2} \pm \text{src3}$ 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.
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;
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

- $dp[63:0] = src1[63:0] \times src2[63:0]+src3[63:0]$

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

```verbatim
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

\[(\text{Booth.sum} + \text{Booth.carry})[105:0] == (\text{ABSMul.sum} + \text{ABSMul.carry})[105:0] == \text{src1}[52:0] \times \text{src2}[52:0]\]

Partition the relation into four sub relations

1) \((\text{Booth}.\text{sum} + \text{Booth}.\text{carry})[105:0] == (\sum_{0}^{26} p_{k} \ll 2k)[105:0]\)

2) \((\sum_{0}^{26} p_{k} \ll 2k)[105:0] == (\sum_{0}^{26}(\text{signed}(\text{src1}[52:0] \times B_{k}) \ll 2k)[105:0]\)

3) \((\sum_{0}^{26}(\text{signed}(\text{src1}[52:0]) \times B_{k}) \ll 2k)[105:0] == (\text{src1}[52:0] \times \sum_{0}^{26}(\text{signed}(B_{k}) \ll 2k)[105:0]\)

4) \((\text{src1}[52:0] \times \sum_{0}^{26}(\text{signed}(B_{k}) \ll 2k)[105:0] == \text{src1}[52:0] \times \text{src2}[52:0]\)

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

We prove a stronger relation:

\[ ((\text{cut}_\text{sum} + \text{cut}_\text{carry})[107:0] == (\text{ABSmul}_53.\text{sum} + \text{ABSmul}_53.\text{carry})[107:0]) \]

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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>fmaddsh</td>
<td>src1*src2+src3</td>
<td>half</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>R1-C0</td>
<td>4201</td>
<td>89</td>
<td>16370</td>
</tr>
<tr>
<td>fmsubsh</td>
<td>src1*src2-src3</td>
<td>half</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>R1-C0</td>
<td>4201</td>
<td>89</td>
<td>17658</td>
</tr>
<tr>
<td>fmaddss</td>
<td>src1*src2+src3</td>
<td>single</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>R1-C0</td>
<td>1989</td>
<td>56</td>
<td>38784</td>
</tr>
<tr>
<td>fmsubss</td>
<td>src1*src2-src3</td>
<td>single</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>R1-C0</td>
<td>1989</td>
<td>56</td>
<td>37745</td>
</tr>
<tr>
<td>fmaddsd</td>
<td>src1*src2+src3</td>
<td>double</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>R1-C1</td>
<td>380</td>
<td>34</td>
<td>3855</td>
</tr>
<tr>
<td>fmsubsd</td>
<td>src1*src2-src3</td>
<td>double</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>R1-C1</td>
<td>380</td>
<td>34</td>
<td>4731</td>
</tr>
<tr>
<td>fmaddse</td>
<td>src1*src2+src3</td>
<td>extended</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>R1-C1</td>
<td>380</td>
<td>31</td>
<td>9519</td>
</tr>
<tr>
<td>fmsubse</td>
<td>src1*src2-src3</td>
<td>extended</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>R1-C1</td>
<td>380</td>
<td>31</td>
<td>10019</td>
</tr>
<tr>
<td>fmaddph</td>
<td>src1*src2+src3</td>
<td>half</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>R1-C0</td>
<td>4764</td>
<td>112</td>
<td>20033</td>
</tr>
<tr>
<td>fmsubph</td>
<td>src1*src2-src3</td>
<td>half</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>R1-C0</td>
<td>4764</td>
<td>112</td>
<td>19572</td>
</tr>
<tr>
<td>fmaddps</td>
<td>src1*src2+src3</td>
<td>single</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>R1-C0</td>
<td>6301</td>
<td>77</td>
<td>36472</td>
</tr>
<tr>
<td>fmsubps</td>
<td>src1*src2-src3</td>
<td>single</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>R1-C0</td>
<td>6299</td>
<td>77</td>
<td>43730</td>
</tr>
<tr>
<td>fmaddpd</td>
<td>src1*src2+src3</td>
<td>double</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>R1-C1</td>
<td>518</td>
<td>32</td>
<td>4339</td>
</tr>
<tr>
<td>fmsubpd</td>
<td>src1*src2-src3</td>
<td>double</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>R1-C1</td>
<td>518</td>
<td>32</td>
<td>5644</td>
</tr>
<tr>
<td>fmaddpe</td>
<td>src1*src2+src3</td>
<td>extended</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>R1-C1</td>
<td>518</td>
<td>29</td>
<td>10237</td>
</tr>
<tr>
<td>fmsubpe</td>
<td>src1*src2-src3</td>
<td>extended</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>R1-C1</td>
<td>518</td>
<td>29</td>
<td>9283</td>
</tr>
<tr>
<td>fma2fma</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>329</td>
<td>16</td>
<td>3756</td>
</tr>
<tr>
<td>c2c</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>490</td>
<td>490</td>
<td>127538</td>
</tr>
</tbody>
</table>
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