Formal methods for FPGAs

John Wickerson

5th September 2022
Formal methods for FPGAs

John Wickerson

September 2022
Theorem transf_c_program_correct:

\[
\forall p \forall tp, \quad \text{transf}_c\_\text{program} p = \text{OK} tp \rightarrow \\
\text{backward}\_\text{simulation} (\text{Csem.}\text{semantics} p) (\text{Asm.}\text{semantics} tp).
\]

Proof.

intros. apply c_\text{semantic}\_\text{preservation}. apply transf_c\_\text{program}\_\text{match}; auto.

Qed.

"All mathematics can be reduced to rules for manipulating formulas without any reference to the meanings of the formulas."

Encyclopædia Britannica

From this proposition it will be defined, that \(1 + 1 = 2\).
Formal methods for...

- specifying the semantics of CPU/FPGA devices
  work led by Dan Iorga, with Alastair Donaldson

- proven-correct high-level synthesis
  work led by Yann Herklotz

- more efficient high-level synthesis
  work led by Jianyi Cheng, with George Constantinides

- high-level synthesis of weak-memory concurrency
  work led by Nadesh Ramanathan, with George Constantinides
CPU/FPGA devices

Intel® Xeon®
Scalable processor with integrated FPGA

Intel®
Core X-series

FPGA
CPU/FPGA Interfaces

//FPGA:
x=1;
y=1;

//CPU:
print(y);
print(x);
CPU/FPGA devices

//FPGA:
x=1;
y=1;

//CPU:
print(y);
print(x);
CPU/FPGA devices

- Upstream channels
- Write-request pool
- Read-request pool
- Downstream channels
- Shared Memory
- Write buffer
- FPGA
- CPU
John Wickerson

Formal methods for FPGAs

Write Request

\[
(WP, RP, UB, DB, SM) \xrightarrow{WR_{\text{FPGA}}} (WP + (W, c, l, o, m), RP, UB, DB, SM)
\]

Read Request

\[
(WP, RP, UB, DB, SM) \xrightarrow{R_{\text{FPGA}}} (WP + (R, c, l, m), UB, DB, SM)
\]

Fence Request One Channel

\[
(WP, RP, UB, DB, SM) \xrightarrow{F_{\text{FPGA}}} (WP + (F, c, l, m), RP, UB, DB, SM)
\]

Fence Request All Channels

\[
(WP, RP, UB, DB, SM) \xrightarrow{F_{\text{FPGA}}} (WP + (F, l, m), RP, UB, DB, SM)
\]

Flush Write Request to Upstream Buffer

\[
WP = \text{head} + (W, c, l, o, m) + \text{tail} \quad (F, c, l, m) \neq \text{head} \quad (F, l, m) \neq \text{head} \\
(WP, RP, UB, DB, SM) \xrightarrow{WR_{\text{FPGA}}} (\text{head} + \text{tail}, RP, UB[c := UB[c] + (W, l, o, m)], DB, SM)
\]

Write to Memory

\[
UB[c] = (W, l, o, m) + \text{tail} \\
(WP, RP, UB, DB, SM) \xrightarrow{WR_{\text{FPGA}}} (WP + \text{tail}, DB, SM) \\
\]

Fence Response One Channel

\[
WP = (F, c, l, m) + \text{tail} \\
UB[c] = \emptyset \\
(WP, RP, UB, DB, SM) \xrightarrow{F_{\text{FPGA}}} (\text{tail}, RP, UB, DB, SM)
\]

Fence Response All Channels

\[
WP = (F, l, m) + \text{tail} \\
\forall c \in \text{Chan. } iB[c] = \emptyset \\
(WP, RP, UB, DB, SM) \xrightarrow{F_{\text{FPGA}}} (\text{tail}, RP, UB, DB, SM)
\]

Flush Read Request to Upstream Buffer

\[
RP = \text{head} + (R, c, l, m) + \text{tail} \\
(WP, RP, UB, DB, SM) \xrightarrow{R_{\text{FPGA}}} (WP, \text{head} + \text{tail}, UB[c := UB[c] + (R, l, m)], DB, SM)
\]

Read from Memory

\[
UB[c] = (R, l, m) + \text{tail} \\
SM[l] = v \\
(WP, RP, UB, DB, SM) \xrightarrow{R_{\text{FPGA}}} (WP, RP, UB[c := \text{tail}], DB[c := DB[c] + (i, o, m)], SM)
\]

Read Response

\[
DB[c] = (l, o, m) + \text{tail} \\
(WP, RP, UB, DB, SM) \xrightarrow{R_{\text{FPGA}}} (WP, RP, UB, DB[c := \text{tail}], SM)
\]

CPU Write

\[
(SM, WB) \xrightarrow{\text{CPU Write}(l, o, i)} (SM, WB) \\
(SM, WB) \\
(SM, WB)[i := WB[i] + (l, o)]
\]

CPU Flash Write Buffer to Memory

\[
WB[i] = (l, o) + \text{tail} \\
(SM, WB) \xrightarrow{\text{CPU Write}(l, o)} (SM, WB) \\
(SM, WB)[i := n, WB[t := \text{tail}]]
\]

CPU Fence

\[
WB[i] = \emptyset \\
(SM, WB) \xrightarrow{\text{CPU Fence}(l, o)} (SM, WB)
\]

CPU Read from Memory

\[
SM[l] = v \\
(SM, WB) \xrightarrow{\text{CPU Read}(l, o)} (SM, WB)
\]

CPU Read from Write Buffer

\[
WB[i] = \text{head} + (l, o) + \text{tail} \\
(SM, WB) \xrightarrow{\text{CPU Read}(l, o)} (SM, WB)
\]

FPGA Step

\[
(WP, RP, UB, DB, SM) \xrightarrow{\text{FPGA}} (WP', RP', UB', DB', SM') \\
(WP, RP, UB, DB, SM, WB) \xrightarrow{\text{FPGA}} (WP', RP', UB', DB', SM', WB)
\]

CPU Step

\[
(SM, WB) \xrightarrow{\text{CPU}} (SM', WB') \\
(WP, RP, UB, DB, SM') \xrightarrow{\text{CPU}} (WP, RP, UB, DB, SM, WB')
\]
CPU/FPGA Interfaces

//FPGA:
\[ x=1; \quad y=1; \]

//CPU:
\[ \text{print}(y); \quad \text{print}(x); \]
//CPU:
print(y);
print(x);

//FPGA:
x=1;
wfence;
y=1;
CPU/FPGA devices

- We built a model of how shared memory works in Intel CPU/FPGA devices

- Can be used as a foundation for reasoning about CPU/FPGA programs

- Can be used to automatically generate conformance tests
Formal methods for...

- specifying the semantics of CPU/FPGA devices
  work led by Dan Iorga, with Alastair Donaldson

- proven-correct high-level synthesis
  work led by Yann Herklotz

- more efficient high-level synthesis
  work led by Jianyi Cheng, with George Constantinides

- high-level synthesis of weak-memory concurrency
  work led by Nadesh Ramanathan, with George Constantinides
Proven-correct HLS
Proven-correct HLS

Theorem transf_c_program_correct:
  forall p tp,
  transf_c_program p = OK tp ->
  backward_simulation (Csem.semantics p) (Asm.semantics tp).

Proof.
  intros. apply c_semantic_preservation. apply transf_c_program_match; auto.
Qed.

Theorem transf_hls_correct:
  forall p tp,
  transf_hls p = OK tp ->
  backward_simulation (Csem.semantics p) (Verilog.semantics tp).

Proof.
  intros. apply c_semantic_preservation. apply transf_hls_match; auto.
Qed.
Proven-correct HLS

C:

```c
int main() {
    int x[2] = {3, 6};
    int i = 1;
    return x[i];
}
```

Verilog:

```verilog
module main(clk, reset, d_in, return_val);
    input [3:0] reset, clk;
    output reg [3:0] finish = 0, return_val = 0;
    reg [15:0] reg[5][3] = 0, d_in = 0, d_out = 0, reg[5][3], w_en = 0;
    reg [15:0] state = 0, reg[2][3] = 0, reg[4][3] = 0, d_out = 0, reg[2][3] = 0;
```

Informal

- Sequential semantics
- Parallel semantics
- Byte-addressed memory
- Word-addressed memory
- Infinite memory
- Finite memory
Proven-correct HLS
Proven-correct HLS
Proven-correct HLS

- 40379 pass
- 114849 compile-time errors
- 39 run-time errors

**Verified components**

- input C: parsing, etc.
- bitstream: FPGA synthesis, etc.

**Flow graph construction**

**Code generation**

**Common subexpression elimination, constant propagation, etc.**
Proven-correct HLS

- We built a proven-correct HLS tool called Vericert
- Vericert is implemented using the Coq proof assistant
- About half as fast as an existing (unverified) HLS tool
- Ongoing work to add more optimisations, chiefly scheduling
- Vericert is open-source and hosted on GitHub

Any questions?