DFiant: A Dataflow Hardware Description Language (HDL)

Oron Port
Electrical Engineering
Technion – Israel Institute of Technology

Yoav Etsion
Computer Science
Technion – Israel Institute of Technology
Why Another Language?

Register-Transfer Level HDLs (e.g., VHDL)
- Concurrency
- Fine-grain control
- Bound to clock
- Explicit pipelining

DFiant: A Dataflow HDL
- Separating timing from functionality
- Concurrency
- Fine-grain control
- Automatic pipelining

High-Level Synthesis Languages and Tools (e.g., C and Vivado HLS)
- Automatic pipelining
- Not an HDL
- Problem with state

Architectures  Algorithms
Separating Register Functionality and Timing

1. **Delay.** Cycle delays are applied to cycle-accurate synchronous interfaces, pipelining, and time derivation.

2. **Synchronization.** Signals passing between clock domains and from asynchronous sources require synchronization.

3. **State.** Preserves a value.
Separating Register Functionality and Timing

1. **Delay.** Cycle delays are applied to cycle-accurate synchronous interfaces, pipelining, and timing derivation.

2. **Synchronization.** Signals passing between clock domains and from asynchronous sources require synchronization.

3. **State.** Preserves a value.
The DFiant Execution Model: **Concurrency**

**Main Code**

- A and B are independent
- C is dependent on both

**Constructed Hardware**

- InputA to Constructed A
- InputB to Constructed B
- OutputA from Constructed A
- OutputB from Constructed B
- OutputC from Constructed C
The DFiant Execution Model: **Concurrency**

**Main Code**

```scala
abstract class Main() {
    val inA : DFUInt[32] <> IN
    val inB : DFUInt[32] <> IN
    val outC : DFUInt[32] <> OUT

    val A = inA * 3
    val B = inB + 2
    val C = A - B
    outC := C
}
```
The DFiAnt Execution Model: State

Main Code

\[
\text{inA := outC.init(2).prev}
\]

\[
\begin{align*}
\text{B} \\
\text{A} \\
\text{B} \\
\text{A} \\
\text{B} \\
\text{C}
\end{align*}
\]

\[
\text{outC.prev} \leftarrow \text{outC}
\]

Implicitly Done!
The DFiAnt Execution Model: **State**

---

**Main Code**

```scala
abstract class Main()
{
  val inB : DFUInt[32] <> IN
  val outC : DFUInt[32] <> OUT
  val C = DFUInt[32].init(2)

  val inA = C.prev
  val A = inA * 3
  val B = inB + 2
  C := A - B
  outC := C
}
```
The DFiant Execution Model: **Code Composition**

type DFB = DFUInt[32] //Type alias, to save code space

abstract class Box(iT: DFB, iB: DFB) { //T=Top, B=Bottom
  val oT: DFB
  val oB: DFB
}

case class BoxY(iT: DFB, iB: DFB) extends Box(iT, iB){
  val (oT, oB) = (iT * iT, iT + iB)
}

case class BoxE(iT: DFB, iB: DFB) extends Box(iT, iB){
  val (oT, oB) = (iT + iB, iB)
}
The DFiant Execution Model: **Code Composition**

```scala
abstract class Box123(iT: DFB, iB: DFB) extends Box(iT, iB) {
  def b1Bld(iT: DFB, iB: DFB) : Box
  def b3Bld(iT: DFB, iB: DFB) : Box
  val b1 = b1Bld(iT, iB)
  val b2 = BoxE(b1.oB, b1.oT)
  val b3 = b3Bld(b2.oB, b2.oT)
  val (oT, oB) = (b3.oT, b3.oB)
}
```

```scala
case class BoxYEE(iT: DFB, iB: DFB) extends Box123(iT, iB) {
  def b1Bld(iT: DFB, iB: DFB) = BoxY(iT, iB)
  def b3Bld(iT: DFB, iB: DFB) = BoxE(iT, iB)
}
```
The DFiant Execution Model: **Code Composition**

RTL requires explicit pipelining

HLS has limited polymorphism & feedback troubles
The DFiant Library

- **Language Semantics:**
  - Implicit concurrency
  - Implicit state
  - Implicit order
  - Implicit default token consumption-production

- **Language Constructs:**
  - Object-oriented, polymorphic, composable
  - (Very) Strongly-typed
  - Bit-accurate selection, aliasing, inference, structures
  - IO Annotations
The DFiAnt Library

- **Language Semantics:**
  - Implicit concurrency
  - Implicit state
  - Implicit order
  - Implicit default token consumption-production

- **Language Constructs:**
  - Object-oriented, polymorphic, composable
  - (Very) Strongly-typed
  - Bit-accurate selection, aliasing, inference, structures
  - IO Annotations

- **Backend Features:**
  - Auto-pipelining
A 64-bit Fibonacci Series Generator

<table>
<thead>
<tr>
<th>DFInt</th>
<th>VHDL</th>
<th>C</th>
<th>Bluespec</th>
<th>Chisel</th>
</tr>
</thead>
<tbody>
<tr>
<td>def fib() : DFUInt[64]={</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>val out = DFUInt[64]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.init(1, 0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>out := out.prev(1) +</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>out.prev(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>out.prev(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0, 1, 1, 2, 3, 5, 8, ...

General formula:

\[ F_n = F_{n-1} + F_{n-2}, \]

\[ F_0 = 0, \quad F_1 = 1. \]
A 64-bit Fibonacci Series Generator

**DFiant**

```python
def fib():
    DFin = DFin[64] =
    out = DFin[64].init(1, 0)
    out := out.prev(1) + out.prev(2)
    out.prev(2)
```

**VHDL**

```vhdl
entity fib is
    port (  
        CLK : IN std_logic;
        DATA : OUT 64;
        RDY : IN std_logic;
        VLD : OUT std_logic;
    );
end fib;
architecture fib_arch of fib is
    signal DATA_out : 64;
    signal out_prev : 64;
    signal out_prev2 : 64;
    type state_type is
        (Out0, Out1, OutOthers);
    signal state : state_type := Out0;
begin
    DATA <= DATA_out;
    process (CLK) is
        begin
            VLD <= '0';
            if RDY = '1' then begin
                VLD <= '1';
                case state is
                    when Out0 =>
                        DATA_out <= Zero64;
                        state <= Out1;
                    when Out1 =>
                        DATA_out <= One64;
                        state <= OutOthers;
                    when OutOthers =>
                        DATA_out <= out_prev + out_prev2;
                end case;
                out_prev2 <= out_prev;
                out_prev <= DATA_out;
            end if;
            end process;
end fib_arch;
```

**C**

```c
U64 U64 fib()
{
  typedef enum
      (Out0, Out1, OutOthers)
      state_type;
  static state_type
      state = Out0;
  static U64 out_prev;
  static U64 out_prev2;
  typedef unsigned long;
  ouz long out = 0;
  switch (state) {
    case Out0:
      state = Out1;
      out = 0;
      break;
    case Out1:
      state = OutOthers;
      out = 1;
      break;
    case OutOthers:
      out = out_prev + out_prev2;
      return out;
  }
}
```

**Bluespec**

```bluespec
interface Fib;
  method ActionValue#(U64) get();
endinterface: Fib

(* synthesizable *)
module mkFib(Fib);
  typedef enum
      (Out0, Out1, OutOthers)
      stateType;
  static stateType
      state = (Out0);
  static U64 out_prev;
  static U64 out_prev2;
  typedef unsigned long
      out = 0;
  switch (state) {
    case Out0:
      state = Out1;
      out = 0;
      break;
    case Out1:
      state = OutOthers;
      out = 1;
      break;
    case OutOthers:
      out = out_prev + out_prev2;
      return out;
  }
end module mkFib
```

**Chisel**

```chisel
class Fib extends Module {
  val io = IO(new Bundle {
    val out = Output(UInt(64,W))
    val rdy = Input(Bool())
    val vld = Input(Bool())
  })
  val sOut0 :: sOut1 :: sOutOthers :: Nil = Enum3;
  val state = RegInit(init = sOut0)
  val out = Wire(UInt(64,W))
  val out_prev = RegNext(out)
  val out_prev2 = RegNext(out_prev)

  when (io.rdy) {
    io.vld := true.B
    switch (state) {
      case (sOut0) {
        out := 6.0
        state := sOut1
      }
      case (sOut1) {
        out := 1.0
        state := sOutOthers
      }
      case (sOutOthers) {
        out := out_prev + out_prev2
      }
    }.
    otherwise(io.vld := false.B)
    io.out := RegNext(out)
  }
endmodule: mkFib
```

0, 1, 1, 2, 3, 5, 8, ...

**General formula:**

\[ F_n = F_{n-1} + F_{n-2}, \]

\[ F_0 = 0, \quad F_1 = 1. \]
A 64-bit Fibonacci Series Generator

**DFiant**

```python
def fib():
    val out = UFInt[64] =
        val out = UFInt[64]
        out := UFInt(1, 0)
        out := out + out

end fib
```

**VHDL**

```vhdl
entity fib is
  port (...
    DATA : UF64;
    RDY : std_logic;
    VLD : std_logic;
  );
end fib;
architecture Fib of fib is
  signal DATA : UF64;
  signal out : UF64;
  signal out 2 : UF64;
  type state_type is (Out0, Out1, OutOthers);
  signal state, state_type := Out0;
begin
  DATA <= DATA;
  process (CLK)
  begin ...
    end process;
end Fib;
```

**C**

```c
U64 fib() {
  typedef {
    (Out0),
    (Out1),
    (Others)
  } state_type;
  static U64 out_prev;
  static U64 out_prev2;
  switch (state) {
    case Out0:
      out_prev = 0;
      break;
    case Out1:
      out_prev = out;
      break;
    case Others:
      out_prev = out_prev + out_prev2;
      return out;
  }
  return out_prev;
}
```

**Bluespec**

```bluespec
(* synthesis *)
module mpFib(F:UF64);
  typedef enum {
    Out0, Out1, Others
  } state_type;
  static U64 out_prev;
  static U64 out_prev2;
  switch (state) {
    case Out0:
      out : 0;
    case Out1:
      out : out;
    case Others:
      out : out + out_prev2;
  }
endmodule
```

**Chisel**

```chisel
class Fib extends Module {
  val io = IO{
    val clk = In(out, 64.w);
    val rdy = In(out, 1.w);
    val vld = Out(out, 1.w);
    ...
  }
  val sOut0, sOut1, sOutOthers
  val sNil = Bool(0);
  val state = Seq.init(0);
  val out = Wire(out, 64);
  val out_prev = Wire(out, 64);
  val out_prev2 = Wire(out, 64);

  def fib()
  io.rdy := true;
  switch (state) {
    case (sOut0)
      out := 0;
      state := sNil
    case (sOut1)
      out := out;
      state := sNil
    case (sOutOthers)
      out := out + out_prev2;
      state := sNil
    }
  }
```

**General formula:**

\[ F_n = F_{n-1} + F_{n-2}, \]

\[ F_0 = 0, \quad F_1 = 1. \]
Conclusions

- DFiant provides seamless concurrent semantics.
- The DFiant code is target device and timing agnostic.
  - Consequently, always correct!

Preliminary Results:
- Implemented AES Encryption and Double-Precision Floating Point Multiplier.
- 33%-50% Lines of Code saved in relation to RTL designs.
- Matching and even surpassing RTL performance.

Future Work:
- Dynamic token-flow control.
- Better backend algorithms (e.g., energy saving).
Contact:
soronpo@campus.technion.ac.il
https://www.researchgate.net/profile/Oron_Port

Follow the DFiant project at:
https://www.researchgate.net/project/DFiant-A-Dataflow-Hardware-Description-Language