DFiant Hardware Description Language

Design By Example

Oron Port

Contents

I Introduction

II Basic Syntax

1 DFiant Syntax 11

1.1 DFiant as a Scala library 11

III Scheduling

2 Dataflow definition convention 15

3 Scheduling Syntax 17

3.1 DF implicit consumption & production 17

3.2 DF conditional consumption & production 17

3.3 new <DFVar>(init) initialized constructor 17

3.4 <DFVar>:= assignment 17

3.5 <DFVar>.prev history value access 17

3.6 <DFVar>.prevInit initialized history value access 17

3.7 <DFVar>.isNotEmpty checks for a valid token at capable producer 17

3.8 <DFVar>.isNotFull checks for an empty slot at capable consumer 17

3.9 <DFVar>.next future value access 17

3.10 <DFVar>.dontConsume prevent value consumption 17

3.11 <DFVar>.dontProduce prevent value production 17

4 Static Scheduling 19
4.1 SISO Example: Identity function 19
4.2 SISO Example: Triplet reverse ordering 20
4.3 SISO Example: Triplet identity function 20
4.4 SISO Example: Sum of three, sliding window 21
4.5 SISO Example: Sum of three, sliding window, initialized 21
4.6 SISO Example: Sum of triplet, Downsampling 3:1 22
4.7 SISO Example: Dual increment, Upsampling 1:3 23
4.8 SISO Example: Place first elements 24
4.9 SISO Example: Drop first elements 25
4.10 SISO Example: Tokens counter 26
4.11 SISO Example: Unstuck repeater 26
4.12 MISO Example: Round robin selector 27
4.13 MISO Example: Round robin greedy balanced selector 28
4.14 MISO Example: Round robin greedy priority selector 29
4.15 NISO Example: Fibonacci series generator 30
4.16 NISO Example: Toggling bit generator 30

5 Dynamic Scheduling 31
5.1 SISO Example: Odd numbers filter 31
5.2 SISO Example: Repeating numbers filter 32

IV Timers

V IO Interface

VI VHDL/Verilog Interface

VII Vendor Libraries

VIII Latex References

6 Introduction to DFiant 43
6.1 Background 43
6.2 Citation 44
6.3 Lists 44
6.3.1 Numbered List 44

6.3.2 Bullet Points 44
6.3.3 Descriptions and Definitions 44

7 Latex template reference 45

7.1 Theorems 45
7.1.1 Several equations 45
7.1.2 Single Line 45

7.2 Definitions 45

7.3 Notations 46

7.4 Remarks 46

7.5 Corollaries 46

7.6 Propositions 46
7.6.1 Several equations 46
7.6.2 Single Line 46

7.7 Examples 46
7.7.1 Equation and Text 46
7.7.2 Paragraph of Text 47

7.8 Exercises 47

7.9 Problems 47

7.10 Vocabulary 47

Introduction

II

Basic Syntax

1 DFiant Syntax 11
1.1 DFiant as a Scala library

1. DFiant Syntax

1.1 DFiant as a Scala library

DFiant is a Scala library and creates its own DSL (Domain Specific Language) within Scala's syntax boundaries. DFiant extends Scala by creating its own classes, types, definition, operators, etc. Therefore, all Scala code is a valid DFiant code, as long as it does not interact with DFiant-exclusive code. Of course this will not result in any runtime DFiant produce, but will pass compilation nonetheless. DFiant requires no special parser, allowing Scala IDE's and tools to be used for DFiant code development, debugging and deployment.

The following table is a summary of errors and their trapping mechanisms:

Error Level	Trapping Mechanism	Typical errors
1	IDE Error Highlighting	Type safety, DFMutability safety
2	Scala Compilation Error	Type safety, DFMutability safety
3	DFiant Compiler Error (Scala Runtime Exception)	TBD
4	DFiant Simulator Error (Scala Runtime Exception)	TBD

The following table contains references to chapters of the Scala Language Specification. Some chapters were extended in DFiant, but most remain unmodified.

Chapter	Title	DFiant Extension
1	Lexical Syntax	Unmodified
2	Identifiers, Names and Scopes	Unmodified
3	Types	Unmodified
4	Basic Declarations and Definitions	Mutable dataflow stream variables. Immutable dataflow stream values. Bit selection & casting Temporal Past & Future token access.
5	Classes and Objects	Structural IN/OUT abstract representation
6	Expressions	Dataflow 'If' expression Design 'If' expression Token temporal consumption & production control
7	Implicit	Unmodified
8	Pattern Matching	Unmodified
9	Top-Level Definitions	Unmodified
10	XML Expressions and Patterns	Unmodified
11	Annotations	DFiant Compiler Annotations Solution Constraints
12	The Scala Standard Library	The DFiant Standard Library Basic Library
13	Syntax Summary	Extended by DFiant

```
UnicodeEscape ::= ‘\’ ‘u’ {‘u’} hexDigit hexDigit hexDigit hexDigit
hexDigit      ::= ‘0’ | ... | ‘9’ | ‘A’ | ... | ‘F’ | ‘a’ | ... | ‘f’
```

III

Scheduling

2 Dataflow definition convention 15

3 Scheduling Syntax 17
3.1 DF implicit consumption & production
3.2 DF conditional consumption & production
3.3 new <DFVar>(init) initialized constructor
3.4 <DFVar>:= assignment
3.5 <DFVar>.prev history value access
3.6 <DFVar>.prevInit initialized history value access
3.7 <DFVar>.isNotEmpty checks for a valid token at capable producer
3.8 <DFVar>.isNotFull checks for an empty slot at capable consumer
3.9 <DFVar>.next future value access
3.10 <DFVar>.dontConsume prevent value consumption
3.11 <DFVar>.dontProduce prevent value production

4 Static Scheduling 19
4.1 SISO Example: Identity function
4.2 SISO Example: Triplet reverse ordering
4.3 SISO Example: Triplet identity function
4.4 SISO Example: Sum of three, sliding window
4.5 SISO Example: Sum of three, sliding window, initialized
4.6 SISO Example: Sum of triplet, Downsampling 3:1
4.7 SISO Example: Dual increment, Upsampling 1:3
4.8 SISO Example: Place first elements
4.9 SISO Example: Drop first elements
4.10 SISO Example: Tokens counter
4.11 SISO Example: Unstuck repeater
4.12 MISO Example: Round robin selector
4.13 MISO Example: Round robin greedy balanced selector
4.14 MISO Example: Round robin greedy priority selector
4.15 NISO Example: Fibonacci series generator
4.16 NISO Example: Toggling bit generator

5 Dynamic Scheduling 31
5.1 SISO Example: Odd numbers filter
5.2 SISO Example: Repeating numbers filter

2. Dataflow definition convention

To define required dataflow functionality, we naturally use sequences to describe inputs/outputs to/from a function. We assume the sequences to be finite.[1]

Definition 2.0.1 — Dataflow sequence. A dataflow sequence of a variable *var* at length of N is defined as follows:

$$S^N_{var} \triangleq (var_0, var_1, \cdots, var_N) \tag{2.1}$$

Definition 2.0.2 — Dataflow sequence set. A dataflow sequence set SS_{DF} is a set of finite dataflow sequences and is defined as follows:

$$SS_{DF} \triangleq \left\{ S^{N_A}_{var_A}, S^{N_B}_{var_B}, \cdots \right\} \tag{2.2}$$

Definition 2.0.3 — Dataflow function. A dataflow function $f_{DF} : SS_{DF,in} \to SS_{DF,out}$ defines the relation between the input sequence set $SS_{DF,in}$ and the output sequence set $SS_{DF,out}$. Each function is definable as follows:

$$SS_{DF,out} = f_{DF}\left(SS_{DF,in}\right) \tag{2.3}$$

R If there is a condition at which an output is not defined by the function, then the output is invalid at that condition, thus the function does not fire and no output will be produced at that condition.

Definition 2.0.4 — Dataflow Single Input, Single Output (SISO) function. A dataflow SISO function has a single input sequence and a single output sequence.

[1] Our dataflow is used to describe hardware. Eventually all hardware stops working :-)

Corollary 2.0.1 A dataflow SISO function may consume at most a single token at any given time and produce a single token at any given time.

Notation 2.1. *Any dataflow function may be described as follows:*

$$(out_0, out_1, \cdots, out_T) = f(in_0, in_1, \cdots, in_N)$$

Notation 2.2. *We will usually use a shorthand approach:*

$$out_t = \begin{cases} h_1(in_t, t) & condition_1 \\ h_2(in_t, t) & condition_2 \\ \vdots & \vdots \end{cases}$$

If none of the conditions is matched, the function is invalid and will not fire (no token is produced).

3. Scheduling Syntax

3.1 DF implicit consumption & production

Most DF scheduling in DFiant is implicit. The default scheduling behavior is detailed in this section. The behavior aims to minimize chance for deadlocks in case the designer forgets to specify the scheduling explicitly.

Implicit consumption of a produced token

Every DF variable (either mutable or immutable) can produce a token. This token will always be consumed, even if not used (no read from the variable), unless specified otherwise (see TBD).

Implicit production of a consumable token

Every constructed DF mutable variable always produces the previous token value it was assigned, unless specified otherwise (see). . This value will always be consumed, even if not used (no read from the variable), unless specified otherwise (see TBD).

3.2 DF conditional consumption & production

3.3 new <DFVar>(init) initialized constructor

3.4 <DFVar>:= assignment

3.5 <DFVar>.prev history value access

3.6 <DFVar>.prevInit initialized history value access

3.7 <DFVar>.isNotEmpty checks for a valid token at capable producer

3.8 <DFVar>.isNotFull checks for an empty slot at capable consumer

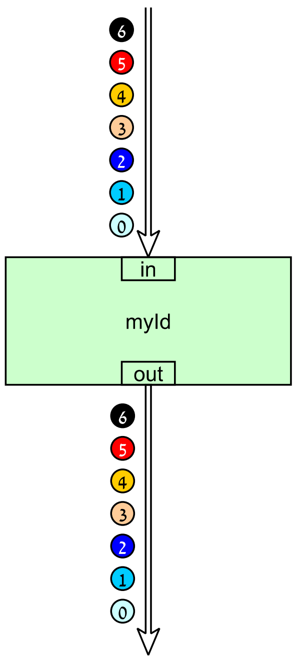
3.9 <DFVar>.next future value access

3.10 <DFVar>.dontConsume prevent value consumption

3.11 <DFVar>.dontProduce prevent value production

4. Static Scheduling

4.1 SISO Example: Identity function



Requirement

An identity function requires the output sequence to be identical in value and ordering to the input sequence.

Example

$myId(0,1,2,3,4,5,6) => (0,1,2,3,4,5,6)$

Definition

$$out_t = myId\left(S_{in}^N\right) = \begin{cases} in_t & t < N \end{cases}$$

All produced tokens are identical to the consumed tokens. An output token is valid (produced) as long as its index is smaller than the number of input (consumed) tokens.

Code

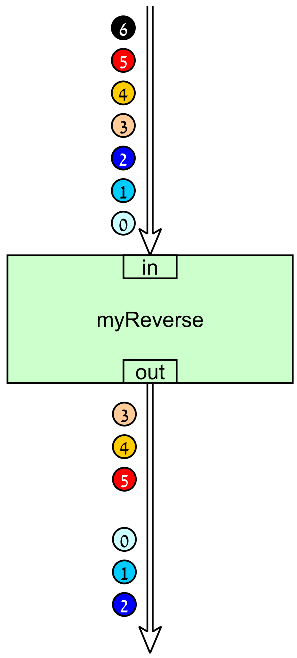
```
def myId(in : DFBits) : DFBits = {
  val out = DFBits(in.width)
  out := in
  return out
}
```

It is possible to use Scala's less verbose approach, as the following code demonstrates. Throughout the examples of this guide we will usually choose the former approach, for consistency and to aid Scala beginner coders.

Code (less verbose)

```
def myId(in : DFBits) : DFBits =
  val out = DFBits(in.width) := in
```

4.2 SISO Example: Triplet reverse ordering



Requirement

Every three numbers are reversed at the output.

Example

$myReverse(0,1,2,3,4,5,6) => (2,1,0,5,4,3)$
Pay notice that the 7th token (value of 6) will be consumed by the function but there will not be a 7th token produced. The function would require two more input token to be able to produce an output.

Definition

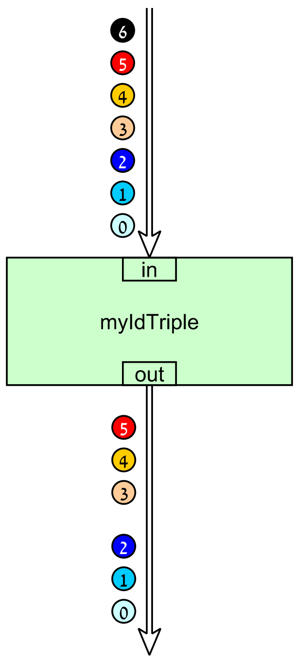
$$out_t = myReverse\left(S_{in}^N\right) = \begin{cases} in_{t+2} & t \bmod 3 = 0,\ t < \lfloor N/3 \rfloor \cdot 3 \\ in_t & t \bmod 3 = 1,\ t < \lfloor N/3 \rfloor \cdot 3 \\ in_{t-2} & t \bmod 3 = 2,\ t < \lfloor N/3 \rfloor \cdot 3 \end{cases}$$

Explanation.....

Code

```
def myReverse(in : DFBits) : DFBits = {
  val out = DFBits(in.width)
  out <-- in.getNextSeq(3).reverse
  return out
}
```

4.3 SISO Example: Triplet identity function



Requirement

Every three numbers are produced AS-IS at the output.

Example

$myIdTriple(0,1,2,3,4,5,6,7) => (0,1,2,3,4,5)$
Pay notice that the 7th and 8th tokens (values of 6 and 7) will be consumed by the function but will not be produced.

Definition

$$out_t = myIdTriple\left(S_{in}^N\right) = \left\{ in_t \quad t < \lfloor N/3 \rfloor \cdot 3 \right.$$

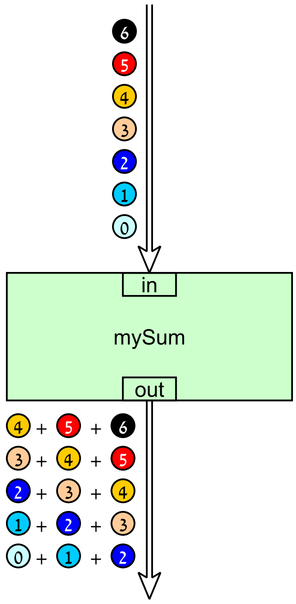
Code

```
def myIdTriple(in : DFBits) : DFBits = {
  val out = DFBits(in.width)
  out <-- in.getNextSeq(3)
  return out
}
```

Remark

$$myReverse\left(myReverse\left(*\right)\right) \equiv myIdTriple(*) \not\equiv myId(*)$$

4.4 SISO Example: Sum of three, sliding window



Requirement

The output is a sliding window sum of every three consecutive inputs.

Example

$mySum(0,1,2,3,4,5,6,7) => (3,6,9,12,15,18)$
Pay notice that N consumed tokens will result in N-2 maximum produced tokens.

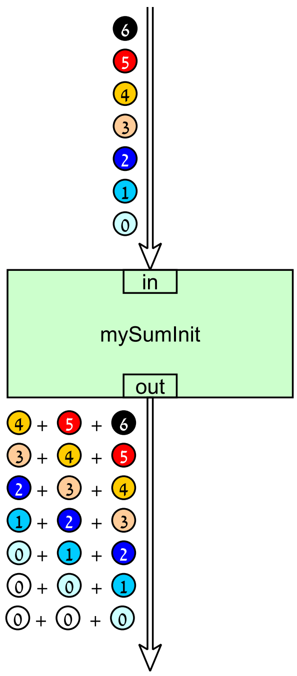
Definition

$$out_t = mySum\left(S_{in}^{N}\right) = \left\{ in_t + in_{t-1} + in_{t-2} \quad 2 \le t < N \right.$$

Code

```
def mySum(in : DFBits) : DFBits = {
  val out = DFBits(in.width)
  out := in + in.prev + in.prev(2)
  return out
}
```

4.5 SISO Example: Sum of three, sliding window, initialized



Requirement

The output is a sliding window sum of every three consecutive inputs. If not enough input tokens are consumed for the first two sum procedures, the inputs are treated as zero values.

Example

$mySumInit(0,1,2,3,4,5,6,7) => (0,1,3,6,9,12,15,18)$

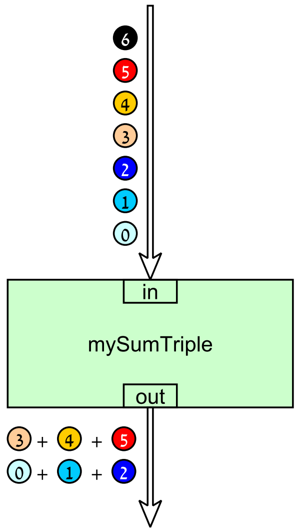
Definition

$$out_t = mySumInit\left(S_{in}^{N}\right) = \begin{cases} in_t + 0 + 0 & t = 0 < N \\ in_t + in_{t-1} + 0 & t = 1 < N \\ in_t + in_{t-1} + in_{t-2} & 2 \le t < N \end{cases}$$

Code

```
def mySumInit(in : DFBits) : DFBits = {
  val out = DFBits(in.width)
  out := in + in.prev.init(0) + in.prev(2).init((0,0)
  return out
}
```

4.6 SISO Example: Sum of triplet, Downsampling 3:1



Requirement

The output is a downsampled sum of every triplet input.

Example

$mySumTriple(0,1,2,3,4,5,6) => (3,12)$

Definition

$$out_t = mySumTriple\left(S_{in}^{N}\right) = \left\{ in_{3t} + in_{3t+1} + in_{3t+2} \quad t < \lfloor N/3 \rfloor \right.$$

Code using two adders

```
def mySumTriple(in : DFBits) : DFBits = {
  val out = DFBits(in.width)
  val in_nseq = in.split(3)
  out := in_nseq(0) + in_nseq(1) + in_nseq(2)
  return out
}
```

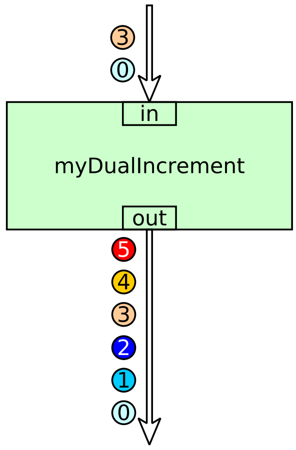
Code using a single adder from library

```
import DFiant.basiclib.{Adder, Reusable}
def mySumTriple(in : DFBits) : DFBits = {
  val adder = Reusable[Adder](in.width, in.width)
  return adder(adder(in, in.next), in.next(2))
}
```

Code using a single adder, do it yourself

```
import DFiant.basiclib.Adder
case class myReusableAdder(width : Integer) {
  //TBD
}
def mySumTriple(in : DFBits) : DFBits = {
  val adder = myReusableAdder(in.width)
  return adder(adder(in, in.next), in.next(2))
}
```

4.7 SISO Example: Dual increment, Upsampling 1:3



Requirement

For each input token *in*, the output will produce three tokens: *in*, $in+1$, and $in+2$.

Example

$myDualIncrement(0,3) => (0,1,2,3,4,5)$

Definition

$$out_t = myDualIncrement\left(S_{in}^{N}\right) =$$

$$= \begin{cases} in_{t/3} & t \; mod \; 3 = 0, t < 3N \\ in_{(t-1)/3} + 1 & t \; mod \; 3 = 1, t < 3N \\ in_{(t-2)/3} + 2 & t \; mod \; 3 = 2, t < 3N \end{cases}$$

Code using two adders

```
def myDualIncrement(in : DFBits) : DFBits = {
  val out = DFBits(in.width)
  out := in
  out.assignNext(1, in + 1)
  out.assignNext(2, in + 2)
  return out
}
```

Code using two adders, shorthand approach

```
def myDualIncrement(in : DFBits) : DFBits = {
  return Seq(in, in + 1, in + 2).merge(in.width)
}
```

Code using two incrementors

```
def myDualIncrement(in : DFBits) : DFBits = {
  val temp = in + 1
  return Seq(in, temp, temp + 1).merge(in.width)
}
```

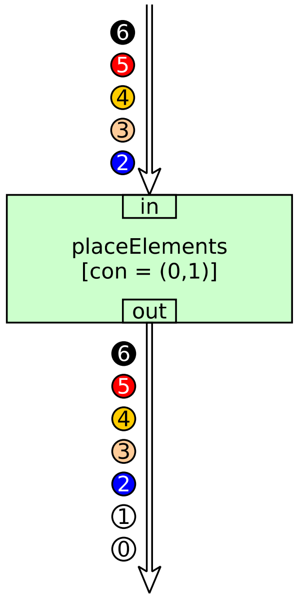
Code using a single adder

```
def myDualIncrement(in : DFBits) : DFBits = {
  val adder = Reusable[Adder](in.width, in.width)
  return Seq(in, adder(in, 1), adder(in, 2)).merge(in.width)
}
```

Code using a single incrementor

```
def myDualIncrement(in : DFBits) : DFBits = {
  val incr = Reusable[Incrementor](in.width, 1)
  val temp = incr(in)
  return Seq(in, temp, incr(temp)).merge(in.width)
}
```

4.8 SISO Example: Place first elements



Requirement

Given an input and a constant sequence, the function will first output the constant sequence and then continue with the input's sequence.

Example

$placeElements\,(in = (2,3,4,5,6), con = (0,1)) \quad => (0,1,2,3,4,5,6)$

Definition

$$out_t = placeElements\left(S_{in}^{N}, S_{con}^{L}\right) = \begin{cases} con_t & t < L \\ in_{t-L} & L \le t < N+L \end{cases}$$

Code

```
1  def placeElements(in : DFBits, con : Seq[BigInt])
2  : DFBits = {
3    val out = DFBits(in.width)
4    val cnt = DFBits(Log2(con.length+1), init=0)
5
6    ifdf (cnt == con.length) {
7      out := in
8    } else {
9      in.dontConsume()
10     out := con(cnt)
11     cnt := cnt + 1
12   }
13   return out
14 }
```

Notes

1. Line 6: Implicit assignment *cnt := cnt.prev* allows us to treat *cnt* as a state which holds its previous value and use *cnt*, instead of the verbose *cnt.prev*. Note that using *cnt.prev* would have achieved the same result.
2. Line 9: Notice the use of *dontConsume()* to prevent a token from *in* variable to be consumed when the ifdf statement condition is false.
3. Lines 11 and 12 must be placed in this order, since assignments take effect immediately within the scope. If we would have used *cnt.prev + 1*, then the result would have been the same in any order.
4. Initialization of *cnt* using its constructor was necessary. Excluding it would have resulted in a compilation error, since without an initialization to *cnt* the function would deadlock due to a missing initial token.

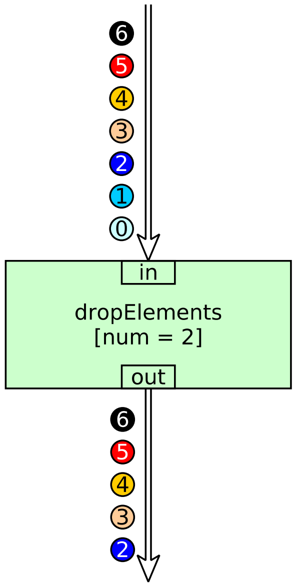
Code alternative

```
def placeElements(in : DFBits, con : Seq[BigInt]) : DFBits = {
  return in.prevInit(con.length, con)
}
```

4.9 SISO Example: Drop first elements

Requirement

Given an input and an L number of elements to drop, the function will consume the first L elements from the input sequence and without producing outputs. Once the consumed number of input elements has reached L the function will produce outputs as the inputs as they are consumed.



Example

$dropElements\,(in = (0,1,2,3,4,5,6), num = 2) \quad => (2,3,4,5,6)$

Definition

$$out_t = dropElements\left(S_{in}^{N}, L\right) = \left\{ in_t \quad L \leq t < N \right.$$

Code

```
def dropElements(in : DFBits, num : Integer)
: DFBits = {
  val out = DFBits(in.width)
  val cnt = DFBits(Log2(num+1), init=0)

  ifdf (cnt == num) {
    out := in
  } else {
    out.dontProduce
    cnt := cnt + 1
  }
  return out
}
```

4.10 SISO Example: Tokens counter

Requirement

Outputs a count of the consumed tokens.

Example

$tokensCounter(5,2,1,5,10,11,2) => (0,1,2,3,4,5,6)$

Definition

$$out_t = tokensCounter\left(S_{in}^N\right) = \begin{cases} t & t < N \end{cases}$$

Code

```
def tokensCounter(in : DFBits) : DFBits = {
  val cnt = DFBits(32, init=0)

  ifdf (in.isNotEmpty()) {
    cnt := cnt + 1
  } elsedf { //If we don't care about repeating count output, we can drop the else
    cnt.dontProduce() //cnt.prev preserves latest valid value
  }
  return cnt
}
```

4.11 SISO Example: Unstuck repeater

Requirement

Any token consumed may be repeatedly produced any number of times until the next token is consumed. The consumer which reads from this function's output controls the production rate.

Example

$unstuckRepeater(0,1,2,3) => (0,0,0,1,1,2,3,3,3,3,3,3,...)$

Definition

$$out_t = unstuckRepeater\left(S_{in}^N\right) = ?????$$

Code

```
def unstuckRepeater(in : DFBits) : DFBits = {
  val out = DFBits(in.width)

  ifdf (in.isNotEmpty()) {
    out := in
  }
  return out
}
```

4.12 MISO Example: Round robin selector

Requirement

Consumes token from $in0$ and outputs it and then consumes token from $in1$ and outputs it and vice versa. This function may hang indefinitely with tokens at its inputs, if one of the inputs is empty.

Example

$rrSel\,((0,2,4,6,8,10),(1,3,5)) => (0,1,2,3,4,5,6)$

Definition

$$out_t = rrSel\left(S_{in}^{N}\right) = ?????$$

Code

```
def rrSel(in0 : DFBits, in1 : DFBits) : DFBits = {
  val out = DFBits(max(in0.width, in1.width))
  val sel = DFBits(1, init=0)

  ifdf (sel == 0) {
    out := in0
    in1.dontConsume()
  } elsedf {
    out := in1
    in0.dontConsume()
  }
  sel := !sel
  return out
}
```

4.13 MISO Example: Round robin greedy balanced selector

Requirement

Consumes token from $in0$ and outputs it and then consumes token from $in1$ and outputs it and vice versa. If a token does not exist on the current input then skip to next input.

Example

$rrGBSel((0,2,4,6,8,10),(1,3,5)) \quad => \quad MANY\_OPTIONS$
This is a time-variant function which depends not only on order of token, but their time as well.

Definition

$$out_t = rrGBSel\left(S_{in}^{N}\right) = ?????$$

Code

```
def rrGBSel(in0 : DFBits, in1 : DFBits) : DFBits = {
  val out = DFBits(max(in0.width, in1.width))
  val sel = DFBits(1, init=0)

  ifdf (sel == 0 && in0.isNotEmpty()) {
    out := in0
    in1.dontConsume()
  } elseifdf (sel == 1 && in1.isNotEmpty()) {
    out := in1
    in0.dontConsume()
  } elsedf {
    out.dontProduce()
    in0.dontConsume()
    in1.dontConsume()
  }
  sel := !sel
  return out
}
```

4.14 MISO Example: Round robin greedy priority selector

Requirement

If $in0$ has a token then consume it and output it. Otherwise, if a token exists at $in1$ then consume it and outputs it.

Example

$rrGPSel\,((0,2,4,6,8,10),(1,3,5)) \quad => \quad MANY\_OPTIONS$

This is a time-variant function which depends not only on order of token, but their time as well.

Definition

$$out_t = rrGPSel\left(S_{in}^{N}\right) = ?????$$

Code

```
def rrGPSel[N0, N1](in0 : DFBits[N0], in1 : DFBits[N1])
    : DFBits[Max[N0, N1]] = {
  val out = DFBits[Max[N0, N1]]

  ifdf (in0.isNotEmpty()) {
    out := in0
    in1.dontConsume()
  } elseifdf (in1.isNotEmpty()) {
    out := in1
    in0.dontConsume()
  } elsedf {
    out.dontProduce()
    in0.dontConsume()
    in1.dontConsume()
  }
  return out
}
```

4.15 NISO Example: Fibonacci series generator

Requirement

Generates Fibonacci series 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Example

$myFib() => (0,1,1,2,3,5,8,13,21,34,...)$

Definition

$$out_t = myFib\,() = \begin{cases} 0 & t = 0 \\ 1 & t = 1 \\ out_{t-1} + out_{t-2} & t \geq 2 \end{cases}$$

Code

```
def myFib() : DFBits[32] = {
  val out = DFBits[32].init(Seq(0,1))
  out := out.prev + out.prev(2)
  return out
}
```

4.16 NISO Example: Toggling bit generator

Requirement

Generates toggling bit 0, 1, 0, 1, 0, 1, ...

Example

$myToggle() => (0,1,0,1,0,1,...)$

Definition

$$out_t = myToggle\,() = \begin{cases} 0 & t = 0 \\ !out_{t-1} & t \geq 1 \end{cases}$$

Code

```
def myToggle() : DFBool = {
  val out = DFBool.init(false)
  out := !out.prev
  return out.prev
}
```

Code alternative

```
def myToggle() : DFBool = {
  val out = DFBool.init(false)
  out := !out.prev
  return out
}
```

5. Dynamic Scheduling

5.1 SISO Example: Odd numbers filter

Requirement

Filters out the odd numbers.

Example

$myOddFilter(0,1,2,3,4,5,6) => (0,2,4,6)$

Definition

$$out_t = myOddFilter\left(S_{in}^{N}\right) = \begin{cases} in_t & ??? \end{cases}$$

Code

```
def myOddFilter(in : DFBits) : DFBits = {
  val out = DFBits(in.width)
  ifdf (in.bit(0) == 0) { //Even
    out := in
  } elsedf {
    out.dontProduce()
  }
  return out
}
```

5.2 SISO Example: Repeating numbers filter

Requirement

Filters out duplicates of a number if occurs more than once consecutively.

Example

$myRepeatFilter(0,1,1,2,3,4,4,5,6,2) => (0,1,2,3,4,5,6,2)$

Definition

$$out_t = myRepeatFilter\left(S_{in}^N\right) = \begin{cases} in_t & ??? \end{cases}$$

Code

```
def myRepeatFilter(in : DFBits) : DFBits = {
  val out = DFBits(in.width)
  val first = DFBool(init = true)
  ifdf (first) {
    out := in
    first := false
  } elsedf {
    ifdf (in == in.prev) {
      out.dontProduce()
    } elsedf {
      out := in
    }
  }
  return out
}
```

Open questions

Do we expect the code to produce a single output if only a single input is consumed? Is the given syntax satisfy or do we need to add for the first condition something like in.prev.dontConsume()?

IV

Timers

V

IO Interface

VI

VHDL/Verilog Interface

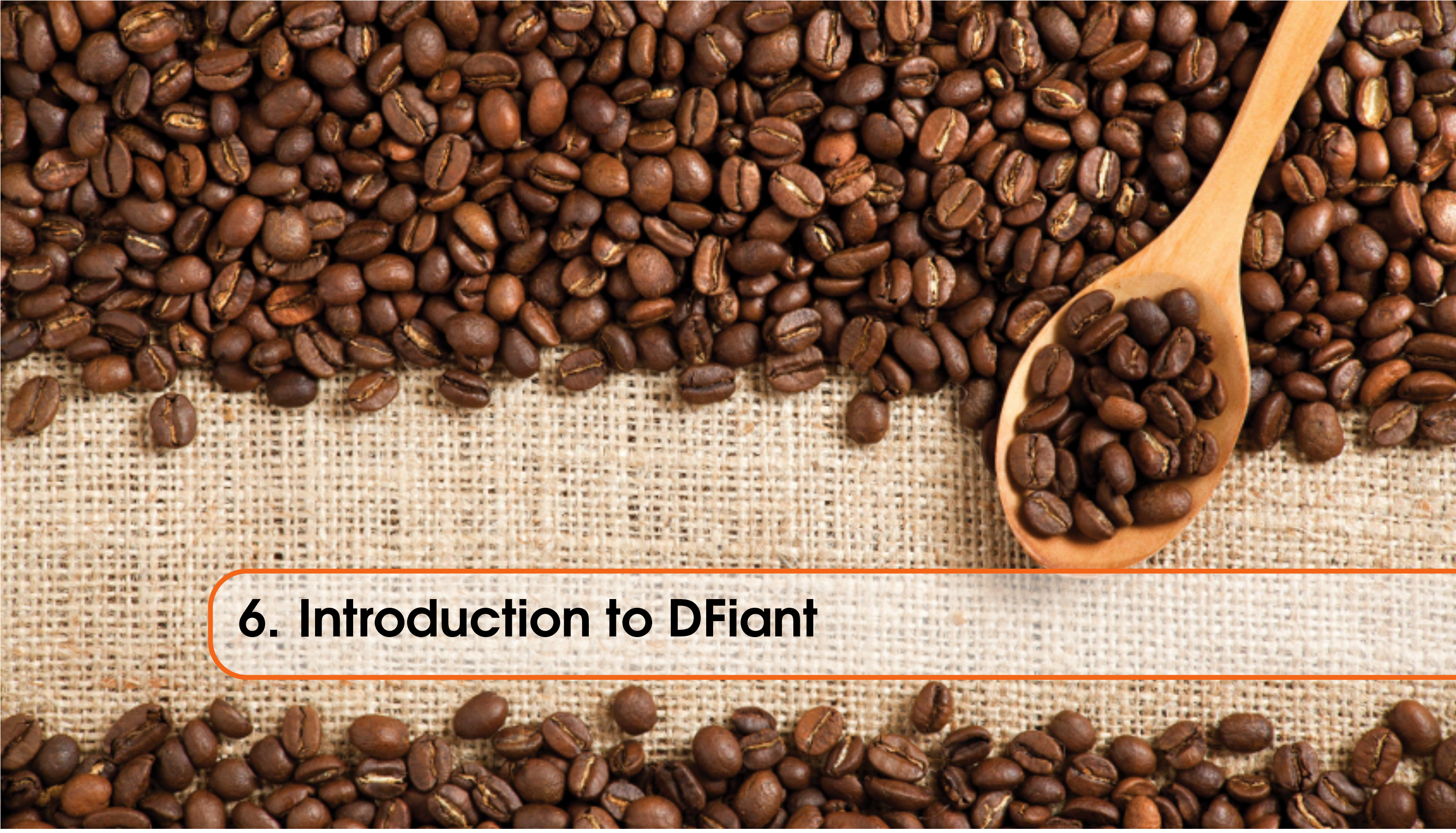
VII

Vendor Libraries

VIII Latex References

6 Introduction to DFiant 43
6.1 Background
6.2 Citation
6.3 Lists

7 Latex template reference 45
7.1 Theorems
7.2 Definitions
7.3 Notations
7.4 Remarks
7.5 Corollaries
7.6 Propositions
7.7 Examples
7.8 Exercises
7.9 Problems
7.10 Vocabulary



6. Introduction to DFiant

6.1 Background

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim.

Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

6.2 Citation

This statement requires citation [**book_key**]; this one is more specific [**article_key**].

6.3 Lists

Lists are useful to present information in a concise and/or ordered way[1].

6.3.1 Numbered List

1. The first item
2. The second item
3. The third item

6.3.2 Bullet Points

- The first item
- The second item
- The third item

6.3.3 Descriptions and Definitions

Name Description
Word Definition
Comment Elaboration

[1] Footnote example...

7. Latex template reference

7.1 Theorems

This is an example of theorems.

7.1.1 Several equations

This is a theorem consisting of several equations.

Theorem 7.1.1 — Name of the theorem. In $E = \mathbb{R}^n$ all norms are equivalent. It has the properties:

$$\big| ||\mathbf{x}|| - ||\mathbf{y}|| \big| \leq ||\mathbf{x} - \mathbf{y}|| \tag{7.1}$$

$$||\sum_{i=1}^{n} \mathbf{x}_i|| \leq \sum_{i=1}^{n} ||\mathbf{x}_i|| \quad \text{where } n \text{ is a finite integer} \tag{7.2}$$

7.1.2 Single Line

This is a theorem consisting of just one line.

Theorem 7.1.2 A set $\mathscr{D}(G)$ in dense in $L^2(G)$, $|\cdot|_0$.

7.2 Definitions

This is an example of a definition. A definition could be mathematical or it could define a concept.

Definition 7.2.1 — Definition name. Given a vector space E, a norm on E is an application, denoted $||\cdot||$, E in $\mathbb{R}^+ = [0, +\infty[$ such that:

$$||\mathbf{x}|| = 0 \Rightarrow \mathbf{x} = \mathbf{0} \tag{7.3}$$

$$||\lambda \mathbf{x}|| = |\lambda| \cdot ||\mathbf{x}|| \tag{7.4}$$

$$||\mathbf{x} + \mathbf{y}|| \leq ||\mathbf{x}|| + ||\mathbf{y}|| \tag{7.5}$$

7.3 Notations

Notation 7.1. *Given an open subset G of $\mathbb{R}^n$, the set of functions φ are:*

1. *Bounded support G;*
2. *Infinitely differentiable;*

a vector space is denoted by $\mathscr{D}(G)$.

7.4 Remarks

This is an example of a remark.

R The concepts presented here are now in conventional employment in mathematics. Vector spaces are taken over the field $\mathbb{K}=\mathbb{R}$, however, established properties are easily extended to $\mathbb{K}=\mathbb{C}$.

7.5 Corollaries

This is an example of a corollary.

Corollary 7.5.1 — Corollary name. The concepts presented here are now in conventional employment in mathematics. Vector spaces are taken over the field $\mathbb{K}=\mathbb{R}$, however, established properties are easily extended to $\mathbb{K}=\mathbb{C}$.

7.6 Propositions

This is an example of propositions.

7.6.1 Several equations

Proposition 7.6.1 — Proposition name. It has the properties:

$$\big| ||\mathbf{x}|| - ||\mathbf{y}|| \big| \leq ||\mathbf{x}-\mathbf{y}|| \tag{7.6}$$

$$||\sum_{i=1}^{n} \mathbf{x}_i|| \leq \sum_{i=1}^{n} ||\mathbf{x}_i|| \quad \text{where } n \text{ is a finite integer} \tag{7.7}$$

7.6.2 Single Line

Proposition 7.6.2 Let $f,g \in L^2(G)$; if $\forall \varphi \in \mathscr{D}(G)$, $(f,\varphi)_0 = (g,\varphi)_0$ then $f=g$.

7.7 Examples

This is an example of examples.

7.7.1 Equation and Text

■ **Example 7.1** Let $G=\{x\in\mathbb{R}^2 : |x|<3\}$ and denoted by: $x^0=(1,1)$; consider the function:

$$f(x) = \begin{cases} e^{|x|} & \text{si } |x-x^0| \leq 1/2 \\ 0 & \text{si } |x-x^0| > 1/2 \end{cases} \tag{7.8}$$

The function f has bounded support, we can take $A=\{x\in\mathbb{R}^2 : |x-x^0| \leq 1/2+\varepsilon\}$ for all $\varepsilon \in]0;5/2-\sqrt{2}[$. ■

7.7.2 Paragraph of Text

■ **Example 7.2 — Example name.** Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris. ■

7.8 Exercises

This is an example of an exercise.

Exercise 7.1 This is a good place to ask a question to test learning progress or further cement ideas into students' minds. ■

7.9 Problems

Problem 7.1 What is the average airspeed velocity of an unladen swallow?

7.10 Vocabulary

Define a word to improve a students' vocabulary.
Vocabulary 7.1 — Word. Definition of word.