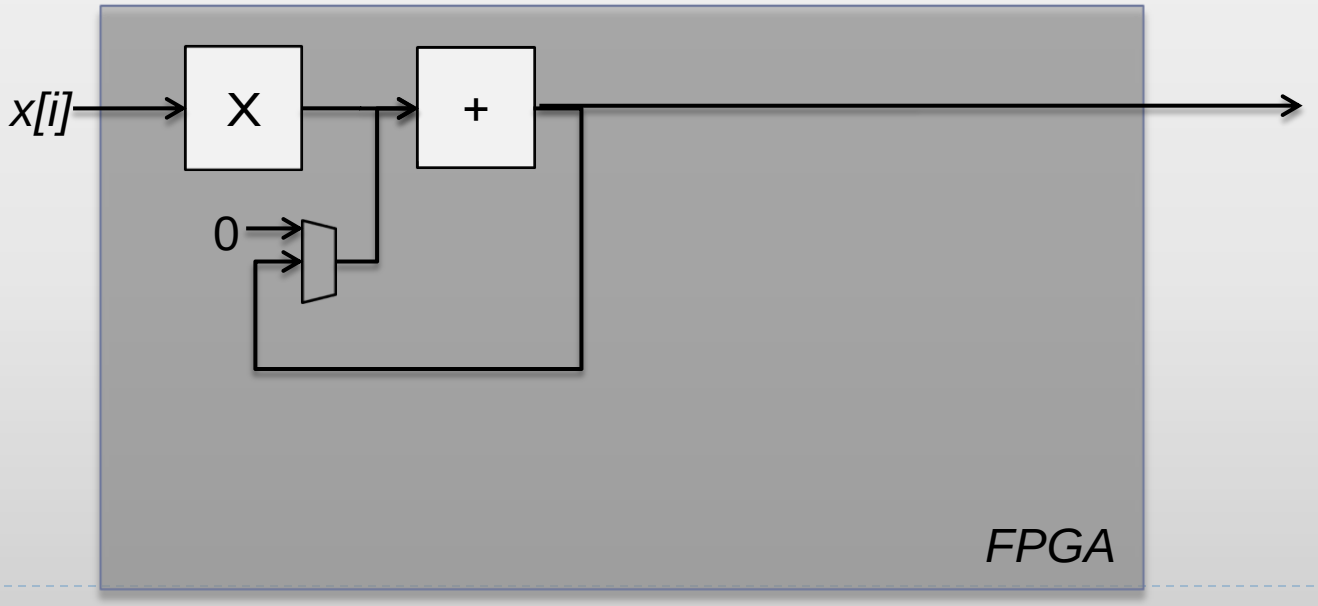
Word-length optimization beyond straight-line code

David Boland
and George A. Constantinides

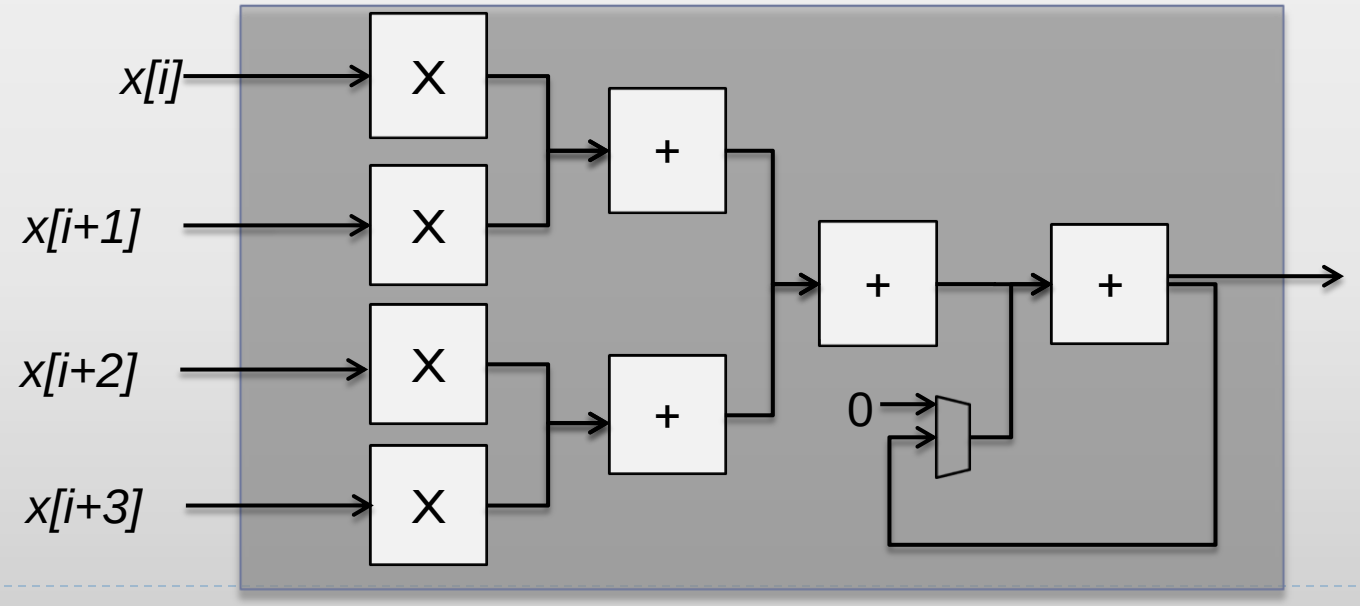
What is word-length optimization?

```
Function dotProduct(x)
output = 0;
for i = 1; i <= 10000; i++ do
    prod = x[i] × x[i];
    output = output+prod;
end for
return(output)
```



What is word-length optimization?

```
Function dotProduct(x)
output = 0;
for i = 1; i <= 10000; i++ do
    prod = x[i] × x[i];
    output = output+prod;
end for
return(output)
```

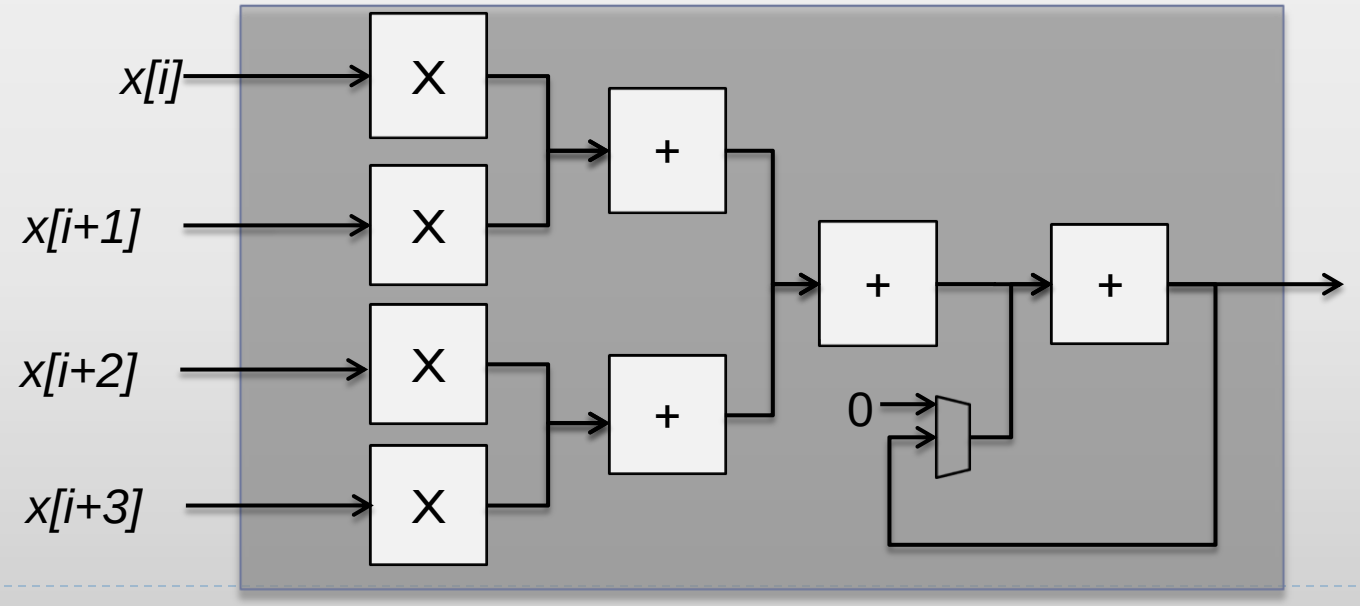


What is word-length optimization?

Known input bounds: $|x| < 1$

Design criteria: Relative error < $1e^{-6}$

Initial design in IEEE double precision. Is this overkill?



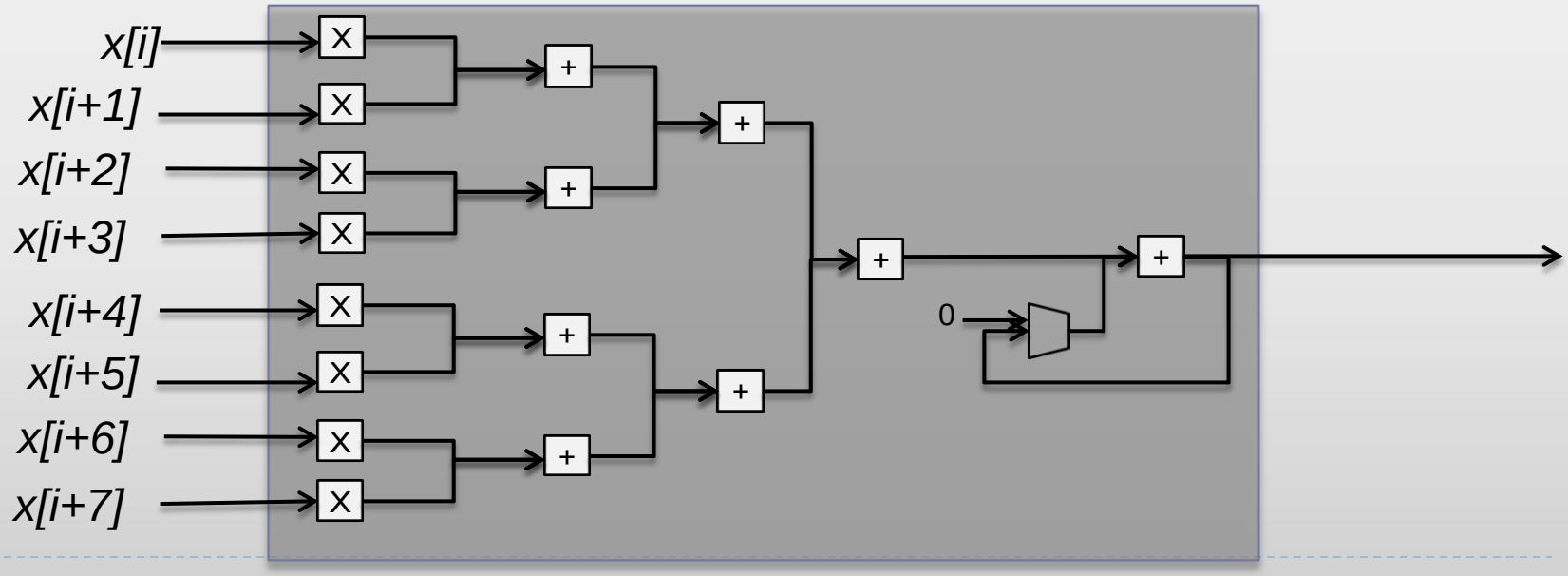
What is word-length optimization?

Known input bounds: $|x| < 1$

Design criteria: Relative error < $1e^{-6}$

Initial design in IEEE double precision. Is this overkill?

Word-length optimization tools find the minimum precision necessary to guarantee design criteria is met



A year ago at FPGA 2012

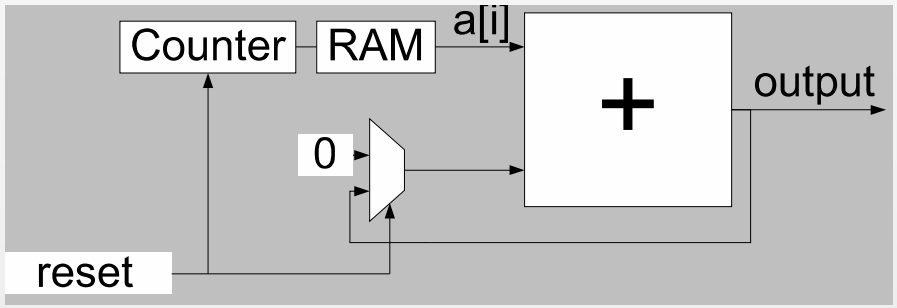
- We presented our latest word-length optimization framework*:
  - Execution time linear in number of operations
  - Bounds approaching quality of algorithms whose execution time grows exponentially in the number of operations.
- A question was raised:

  "Can your approach deal with loops?"

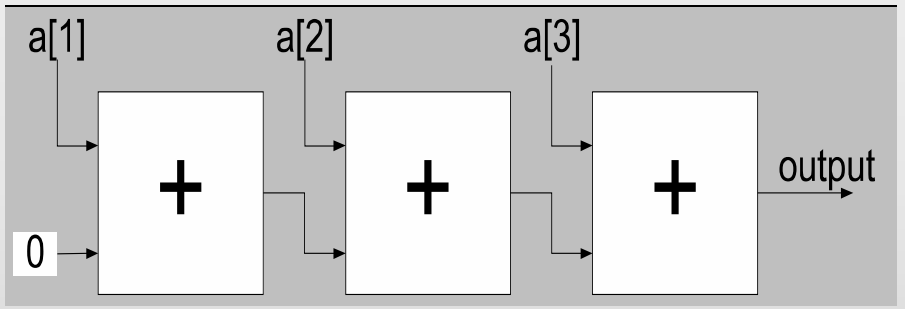
*David Boland and George A. Constantinides, *A Scalable Approach for Automated Precision Analysis*, Proc. ACM/SIGDA Int. Conf. on Field-Programmable Gate Arrays, pp. 185-194, 2012.

Can your approach deal with loops?

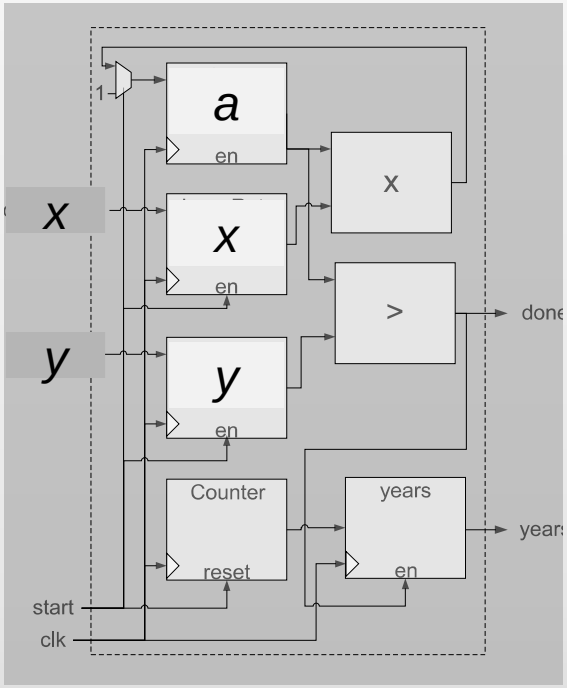
```
Vector_sum(a);
output = 0;
for i = 1; i <= 3; i++ do
    output = output+a[i];
end for
```



- Unroll "for" loop:
  - The error seen at the output of these architectures are equivalent.
- But how do you deal with "while" loops?
  - How much do you unroll?



A toy example




```
Function RadioActiveDecay (fixed4bit x, fixed4bit y)
// x, y ∈ [1/16; 15/16]
int8bit years;
a = 1;
while a > y do
    a = a × x
    years++;
end while
return(years)
```

How many bits are needed for the variable *a*?

Um…5 bits?

- If x=0.9375, y=0.125

```
Function RadioActiveDecay (fixed4bit x, fixed4bit y)
// x, y ∈ [1/16; 15/16]
int8bit years;
a = 1;
while a > y do
    a = a × x
    years++;
end while
return(years)
```

Loop Iteration	Years	a (before rounding)	a (after rounding)
0	0	1	1
1	1	0.9375	0.9375
2	2	0.87890625	0.875
3	3	0.8203125	0.8125
4	4	0.76171875	0.75
5	5	0.703125	0.71875
⋮	⋮	⋮	⋮
19	19	0.29296875	0.28125
20	20	0.263671875	0.25
21	21	0.234375	0.25

Rounded up to same value ⇔ Loop will run forever

Um…5 bits?

- If x=0.9375, y=0.125

```
Function RadioActiveDecay (fixed4bit x, fixed4bit y)
// x, y ∈ [1/16; 15/16]
int8bit years;
a = 1;
```

For the accelerator to be useful, it must terminate at some point:

Choose the fewest number of bits required to ensure termination

4	4	0.76171875	0.75
5	5	0.703125	0.71875
⋮	⋮	⋮	⋮
19	19	0.29296875	0.28125
20	20	0.263671875	0.25
21	21	0.234375	0.25

Rounded up to same value

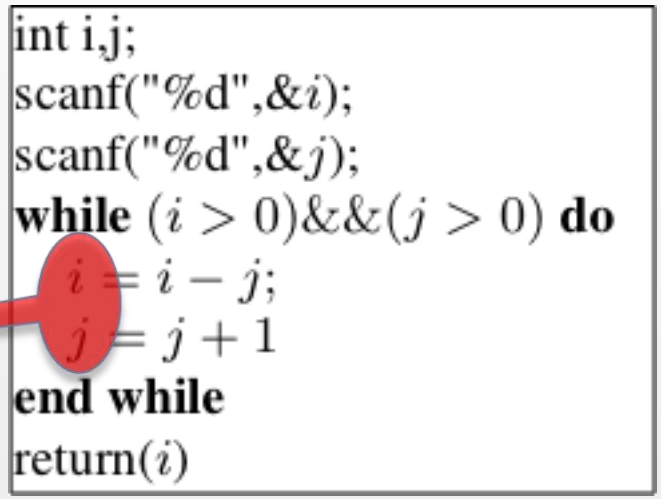
⇔

Loop will run forever

So how do you deal with "while" loops?

- Borrowed some ideas from the software verification community regarding loop termination
- Extended these ideas to:
  - Incorporate finite precision arithmetic affects
  - Deal with non-linear operations
  - Demonstrate its use for hardware design

Proving loop termination



Loop variables are:
i,j

Loop variables after a loop iteration are:
i'=i-j
j'=j+1

- We analyse loop variables before and after a **single** loop iteration
- Loop updates must ensure the loop variables always move towards the loop exit condition

Proving loop termination

1. Create a *ranking function* $f(i, j, \dots)$ that maps every potential state within the loop to a positive real number.
2. Show that after every loop iteration, this function always decreases by more than some fixed amount $\epsilon > 0$, *i.e:*

$$f(i', j', \dots) < f(i, j, \dots) - \epsilon$$

Eventually the loop transition statement will cause:

$$f(i', j', \dots) < 0$$

this corresponds to the loop terminating.

Proving loop termination example

1. Create a *ranking function* that maps every potential state within the loop to a positive real number.

$$f(i) = 100 - i$$

2. Show:

$$f(i', j', \dots) < f(i, j, \dots) - \epsilon$$

where $\epsilon > 0$

$$f(i) = 100 - i$$

$$f(i') = 100 - (i + 1) = 99 - i$$

then $99 - i < 100 - i - \epsilon$ if

$$0 < \epsilon < 1$$

```
int i;
scanf("%d",&i);
while (i < 100) do
    i = i + 1;
end while
return(i)
```

Remaining questions?

- How to choose the ranking function $f(i, j, \ldots)$?
  - We describe a search procedure for several candidate ranking functions in the paper.
- If $f(i, j, \ldots)$ can be any polynomial function, is proving that:

  $$f(i', j', \ldots) < f(i, j, \ldots) - \epsilon$$

  nontrivial?
  - Yes…but if we re-write equation as

    $$0 < f(i, j, \ldots) - f(i', j', \ldots) - \epsilon$$

    previous work* can be used to bound the right-hand polynomial
  - If the lower bound of right-hand polynomial > 0, then we have proved the ranking function holds
- How to prove termination in finite precision arithmetic?

*David Boland and George A. Constantinides, *A Scalable Approach for Automated Precision Analysis*, Proc. ACM/SIGDA Int. Conf. on Field-Programmable Gate Arrays, pp.185-194, 2012.

How to prove termination in finite precision arithmetic?

- Create polynomial bounding the worst-case output range after any operation $\odot \in \{+, -, \times, /\}$
- For n-bit fixed point:

  $$\textbf{fix}(x \odot y) = x \odot y + \delta \qquad |\delta| \leq 2^{-n}$$

- For floating point with n-bit mantissa:

  $$\textbf{float}(x \odot y) = (x \odot y)(1 + \delta)$$

- Show that even after worst-case rounding,

  $$f(\textbf{fix}(i'), \textbf{fix}(j'), \dots) < f(i, j, \dots) - \epsilon \qquad \text{or}$$

  $$f(\textbf{float}(i'), \textbf{float}(j'), \dots) < f(i, j, \dots) - \epsilon$$

  still holds

Proving loop termination for RadioActiveDecay

```
Function RadioActiveDecay (fixed4bit x, fixed4bit y)
// x, y ∈ [1/16; 15/16]
int8bit years;
a = 1;
while a > y do
   a = a × x
   years + +;
end while
return(years)
```

1. Create a *ranking function* that maps every potential state within the loop to a positive real number.

$$f(a, years) = a$$

2. Show: $f(\mathbf{fix}(a'), \mathbf{fix}(years')) < f(a, years) - \epsilon$

   where $\epsilon > 0$

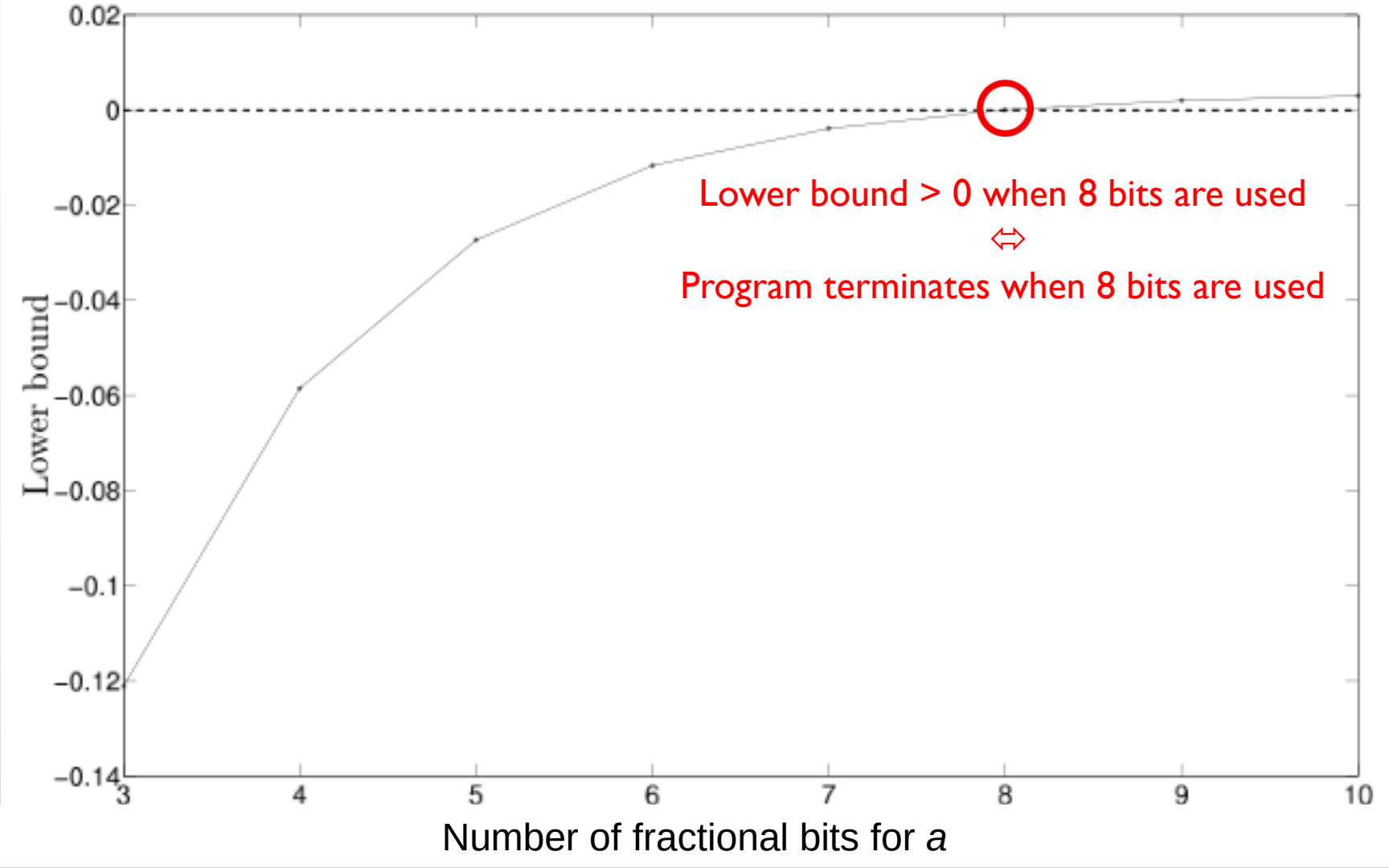
   since $f(\mathbf{fix}(a'), \mathbf{fix}(years')) = a \times x + \delta$

   $f(a, years) = a$

   Terminates if $a \times x + \delta < a - \epsilon$

   or $0 < a - (a \times x + \delta) - \epsilon$

Lower bound for $a - (a \times x + \delta) - \epsilon$



Potential benefits for FPGA-based accelerators.

RadioActiveDecay example

```
int8 years = 0;
fixed?bit currAmount = 1;
while currAmount > limit do
    years = years+1;
    currAmount = currAmount × decayRate;
end while
return(years);
```

Newton's method to compute the square root of a number

2: // where i lies in the interval $[0;\ 100]$ and $\eta = 1 \times 10^{-7}$
3: Function Newton_SQRT(i)
4: $i_0 = 1, i_1 = i; k = 0$
5: **while** $|i_{k+1} - i_k| > \eta$ **do**
6: $\quad i_{k+1} = \frac{1}{2}\left(i_k + \frac{i}{i_k}\right)$
7: $\quad k = k + 1;$
8: **end while**
9: return(i)

Euclid's method to compute GCD

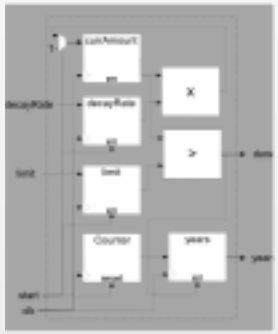
2: // where a lies in the interval $[0.1;\ 1000]$, b lies in the interval $[0.1;\ 100]$ and $a > b$.
3: Function GCD(a, b)
4: **while** $b > 0.1$ **do**
5: $\quad c = b;$
6: $\quad b = a - \lfloor a/b \rfloor b$
7: $\quad a = c;$
8: **end while**
9: return(a)

Euler's method to solve differential equations

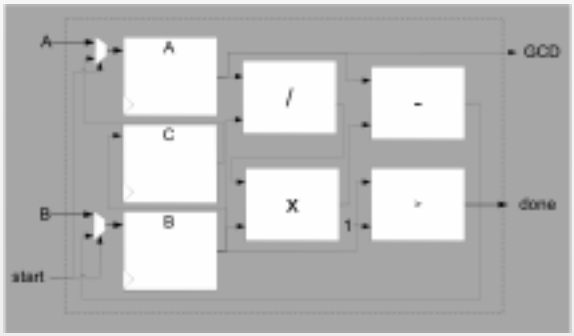
$a \in [0;\ 1]$ and $b \in [0;\ 1]$, $c \in [90;\ 100]$, $d \in [0.4;\ 0.5]$, $\eta_1 \in [0.1;\ 1]$ and $\eta_2 \in [10^{-6};\ 1]$*/
1: Function Euler($a, b, c, d, y_{in}, \eta_1, \eta_2$)
2: $y_{curr} = y_{in}$
3: **while** $a < b$ **do**
4: $\quad h = 0.5$
5: $\quad$ **do**
6: $\quad\quad y_0 = y_{curr} + h \times (c + d \times y_{curr})$
7: $\quad\quad mid = y_{curr} + \frac{1}{2}h \times (c + d \times y_{curr})$
8: $\quad\quad f = c + \frac{1}{2}h + d \times mid$
9: $\quad\quad y_1 = y_{curr} + \frac{1}{2}h \times ((c + d \times y_{curr}) + f)$
10: $\quad\quad \tau = y_1 - y_0$
11: $\quad\quad h = \frac{1}{2}h$
12: $\quad$ **while** $(\tau > \eta_1)\&\&(h > \eta_2)$
13: $\quad y_{curr} = y_0;$
14: $\quad a = a + 2 \times h;$
15: **end while**
16: return(a)

Potential benefits for FPGA-based accelerators.

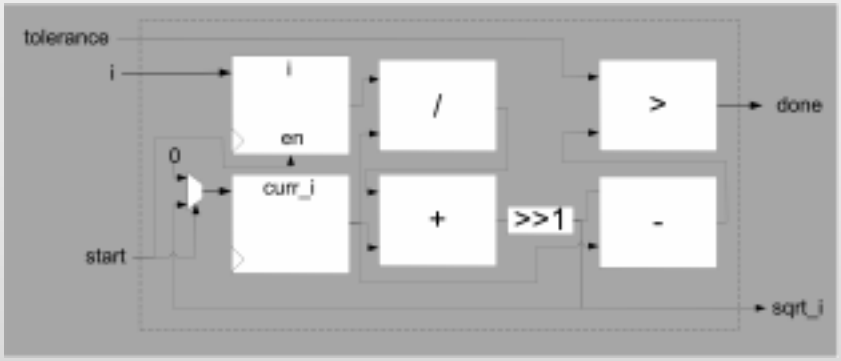
RadioActiveDecay example



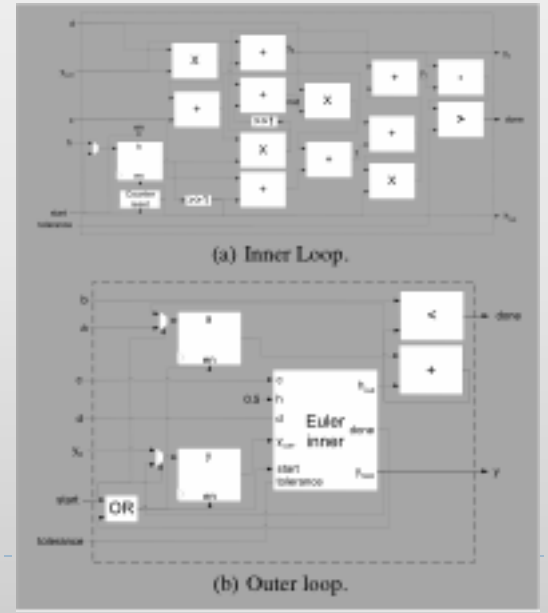
Newton's method to compute the square root of a number



Euclid's method to compute GCD



Euler's method to solve differential equations



Potential benefits for FPGA-based accelerators.

RadioActiveDecay example

# bits for *currAmount*	Slices	DSPs	Frequency (MHz)
8	14	1	275
12	15	1	275
16	16	1	275
20	17	2	210

Euclid's method to compute GCD

# fractional bits	Slices	DSPs	Frequency (MHz)
5	141	1	230
6	183	1	225
7	203	1	220
8	262	1	190
9	289	1	195
10	274	2	180

Newton's method to compute the square root of a number

Method	Precision (# bits)	Slices	Frequency (MHz)
Our Approach	27	1,236	490
IEEE Single Precision	24	999	500
IEEE Double Precision	53	2,667	330

For fixed point examples, our approach can pick smallest, fastest architecture & guarantee it will terminate

Method	Precision (# bits)	Slices	Frequency (MHz)
Our Approach	9,17	418	550
IEEE Single Precision	24	926	490
IEEE Double Precision	53	2230	480

Potential benefits for FPGA-based accelerators.

RadioActiveDecay example

# bits for *currAmount*	Slices	DSPs	Frequency (MHz)
8	14	1	275
12	15	1	275
16	16	1	275

For floating point examples, our approach can pick smallest, fastest architecture & guarantee it will terminate

# fractional bits	Slices	DSPs	Frequency (MHz)
5	141	1	230
6	183	1	225
7	203	1	220
8	262	1	190
9	289	1	195
10	274	2	180

Newton's method to compute the square root of a number

Method	Precision (# bits)	Slices	Frequency (MHz)
Our Approach	27	1,236	490
IEEE Single Precision	24	999	500
IEEE Double Precision	53	2,667	330

Euler's method to solve differential equations

Method	Precision (# bits)	Slices	Frequency (MHz)
Our Approach	9,17	418	550
IEEE Single Precision	24	926	490
IEEE Double Precision	53	2230	480

Potential benefits for FPGA-based accelerators.

Note a single precision architecture wouldn't terminate -> it wouldn't work!

# bits for *currAmount*	Slices	DSPs	Frequency (MHz)
8	14	1	275
12	15	1	275
16	16	1	275

For floating point examples, our approach can pick smallest, fastest architecture & guarantee it will terminate

# fractional bits	Slices	DSPs	Frequency (MHz)
5	141	1	230
6	183	1	225
7	203	1	220
8	262	1	190
9	289	1	195
10	274	2	180

Newton's method to compute the square root of a number

Method	Precision (# bits)	Slices	Frequency (MHz)
Our Approach	27	1,236	490
IEEE Single Precision	24	999	500
IEEE Double Precision	53	2,667	330

Euler's method to solve differential equations

Method	Precision (# bits)	Slices	Frequency (MHz)
Our Approach	9,17	418	550
IEEE Single Precision	24	926	490
IEEE Double Precision	53	2230	480

Potential benefits for FPGA-based accelerators.

Note a single precision architecture wouldn't terminate -> it wouldn't work!

# bits for *currAmount*	Slices	DSPs	Frequency (MHz)
8	14	1	275
12	15	1	275
16	16	1	275

For floating point examples, our approach can pick smallest, fastest architecture & guarantee it will terminate

# fractional bits	Slices	DSPs	Frequency (MHz)
5	141	1	230
6	183	1	225
7	203	1	220
10	274	2	180

Savings of 50 % or 80 % over IEEE double precision arithmetic

Newton's method to compute the square root of a number

Method	Precision (# bits)	Slices	Frequency (MHz)
Our Approach	27	1,236	490
IEEE Single Precision	24	999	500
IEEE Double Precision	53	2,667	330

Euler's method to solve differential equations

Method	Precision (# bits)	Slices	Frequency (MHz)
Our Approach	9,17	418	550
IEEE Single Precision	24	926	490
IEEE Double Precision	53	2230	480

Can your approach deal with loops?

- Loop termination is provably undecidable in the general case
- Nevertheless
  - Described a technique to choose precision for code containing while loops
    - Incorporate finite precision models into a termination argument.
    - Use precision analysis techniques to validate such a termination argument.
    - Scalable
      - Only analyses the loop body for a single iteration of the loop
  - Our technique can prove loop termination in finite precision for examples where:
    - Loop body consists any arithmetic operations
    - Loop exit conditions are linear and conjunctive
    - Beyond the capabilities of current tools from the software verification community
  - Demonstrated its use for word-length optimization of basic hardware accelerators