Programmable Numerical Function Generators Based on Quadratic Approximation: Architecture and Synthesis Method

Shinobu Nagayama[1] Tsutomu Sasao[2] Jon T. Butler[3]

[1]Hiroshima City University, Japan

[2]Kyushu Institute of Technology, Japan

[3]Naval Postgraduate School, U.S.A.

Numerical Function Generators (NFGs)

x

$y \simeq f(x)$

NFG computes an approximated value *y* for a numerical function *f*(*x*) with some <u>given acceptable error</u>.

e.g. Trigonometric, logarithm functions

Background

- **Numerical functions are extensively used in:**
  - Digital signal processing
  - Communication systems
  - Robotics
  - Graphics applications
  - Astrophysics
  - Fluid physics
  - Etc.
- **Fast NFGs are required.**

Algorithms for NFGs

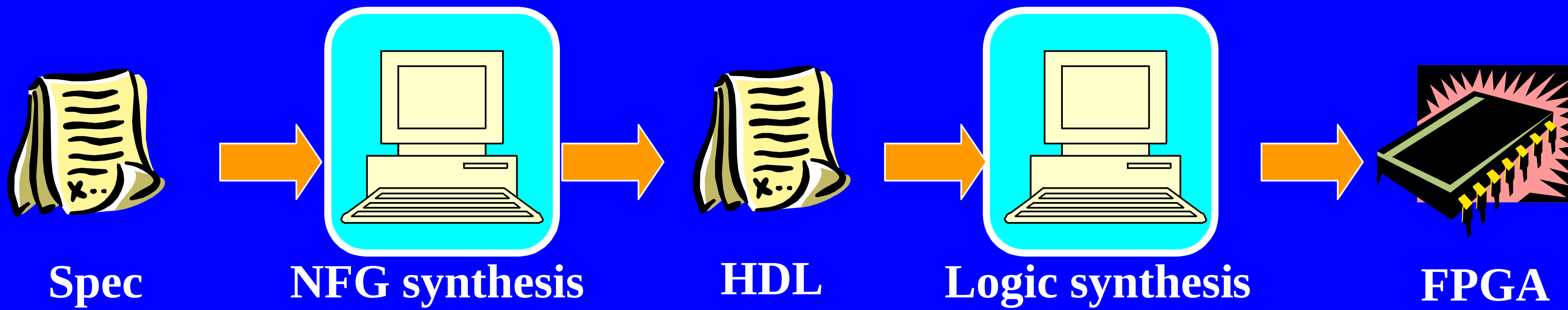
Algorithms	Speed	Size	Precision
Single lookup table (ROM)	Very Fast	Huge	Low
Interpolation (linear interpolation)	Fast	Middle	Middle
Iterative algorithm (CORDIC)	Slow	Small	High

Research Objectives

- **We propose an architecture for fast and compact programmable NFGs.**
  - **Uses an LUT cascade.**
  - **Is based on interpolation by non-uniform segmentation.**
  - **Is based on quadratic approximation.**
  - **Realizes high precision NFGs with small memory size.**
  - **Can be implemented with a compact and low-cost FPGA.**

- **We develop an automatic synthesis system for NFGs.**
  - **It converts MATLAB-like specification into HDL.**
  - **Users need no knowledge of LSI.**

Synthesis Flow for NFGs

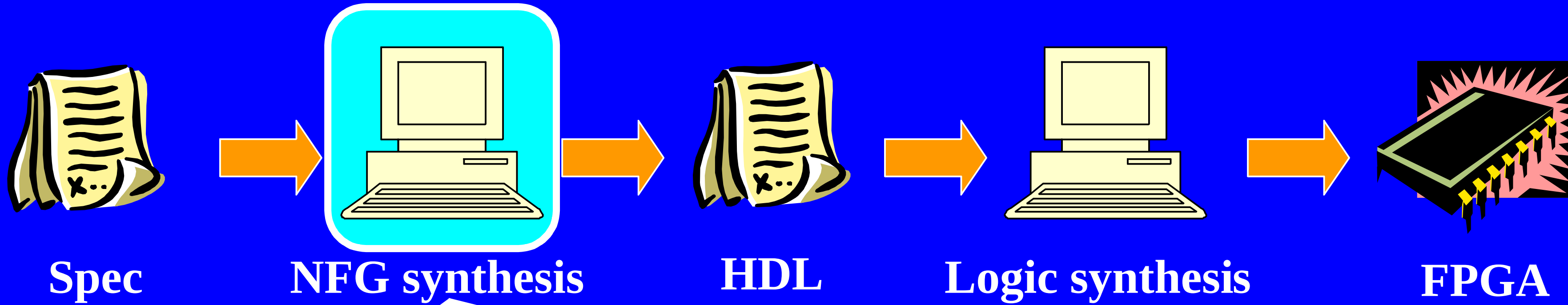


- Function $f(x)$
- Domain $[a, b]$
- Precision
  - \# input bits
  - \# output bits

Features of this synthesis

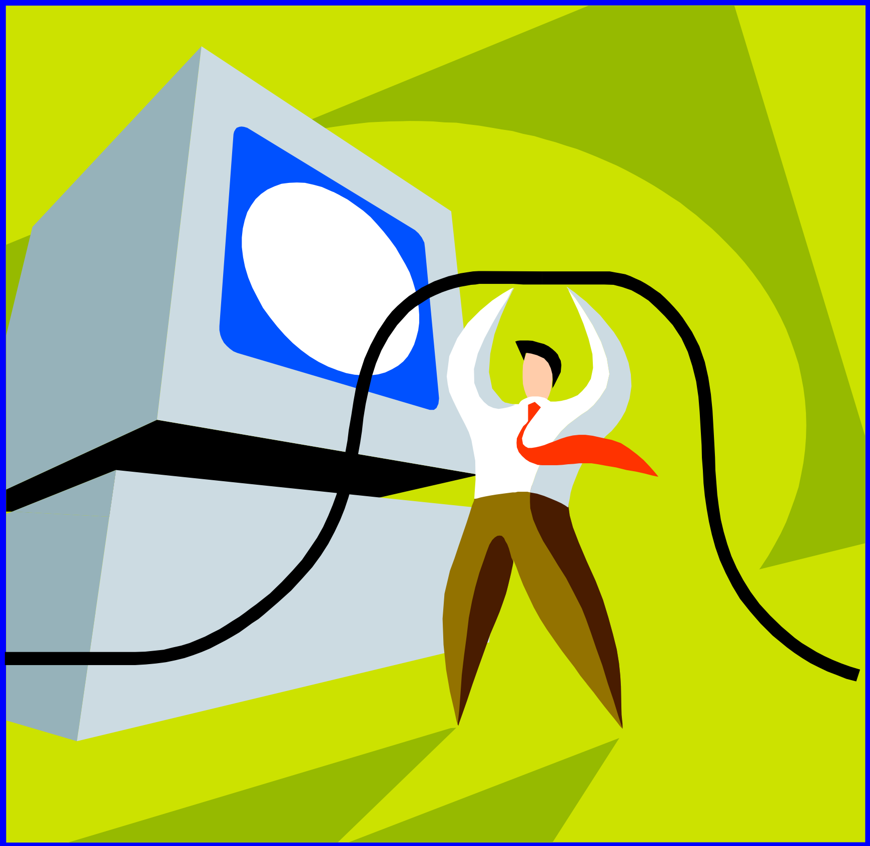
- Users provide only the spec using numerical software (e.g. MATLAB).
- No need to design by HDL or block diagram.
- The use of LUT cascades facilitates automatic synthesis.

Outline of NFG Synthesis



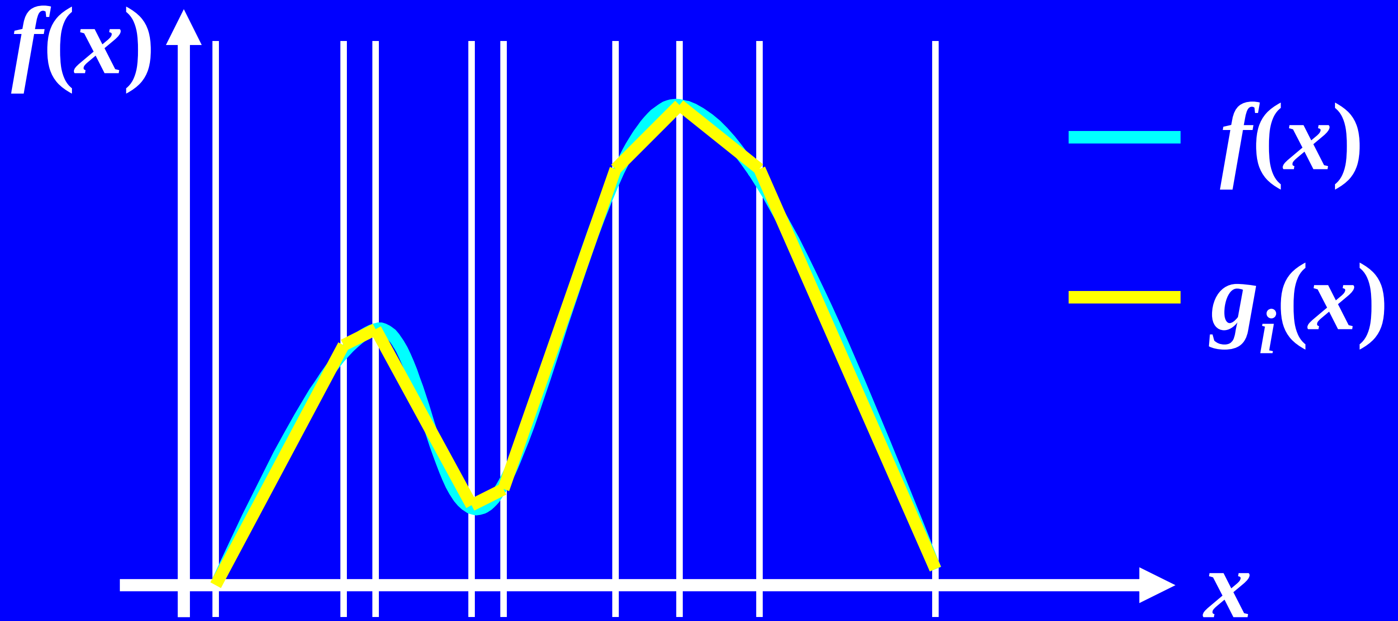
1. **Function approximation (Segmentation algorithm)**
2. **Error analysis for NFG**
3. **Computation of bit-sizes**
4. **HDL code generation**

Function Approximation



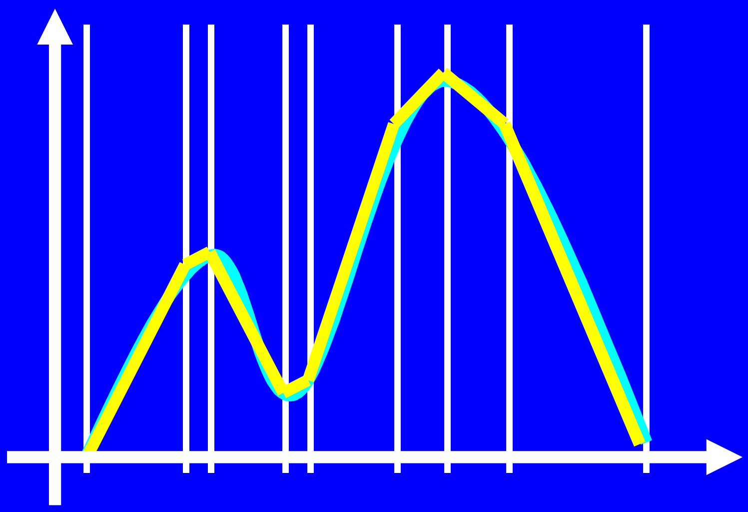
Approximation Method to $f(x)$

- Partition the domain into **non-uniform segments**.
- Approximate $f(x)$ by a polynomial function $g_i(x)$ for each segment.

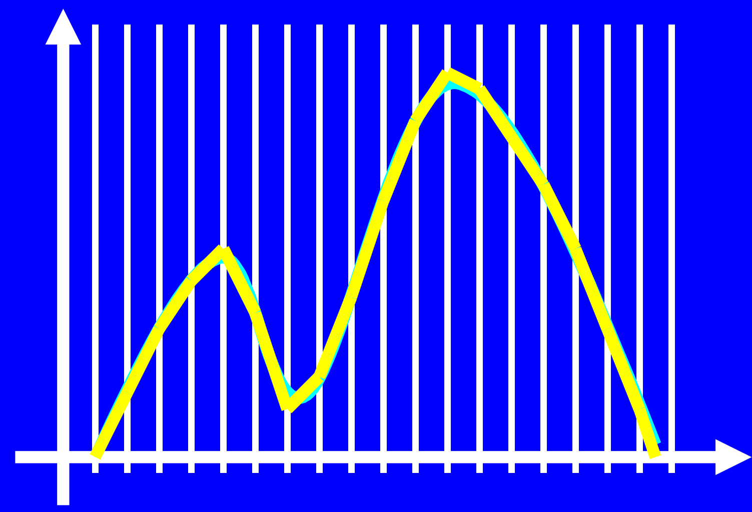


Non-uniform Segmentation

- **Approximates the function with fewer segments.**
  - **Less memory size is required.**



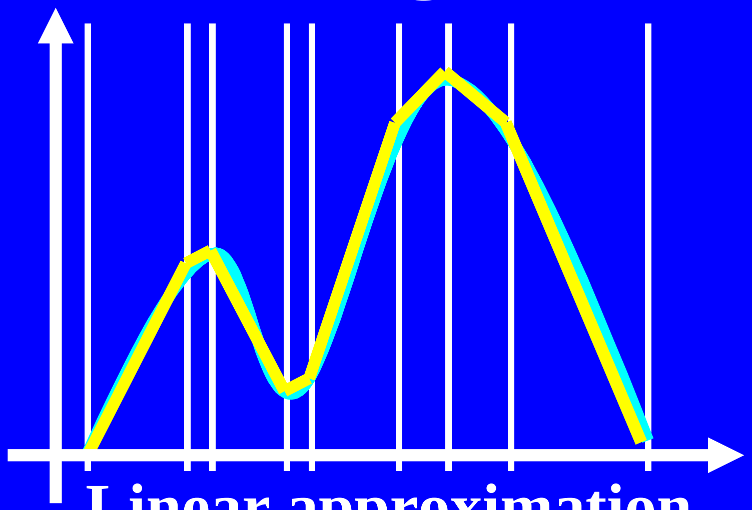
Non-uniform segmentation



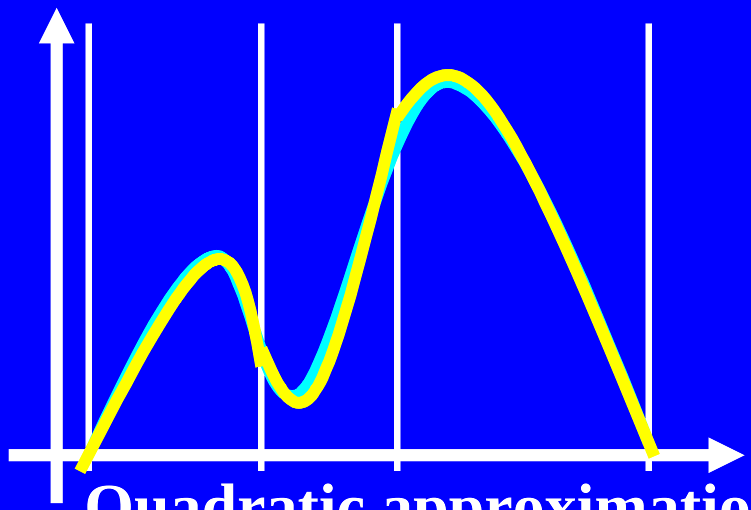
Uniform segmentation

Number of Non-uniform Segments

- Depends on approximation polynomial.
  - Accurate approximation polynomial requires fewer segments.
- 2nd-order Chebyshev polynomial is used in each segment.



Linear approximation



Quadratic approximation

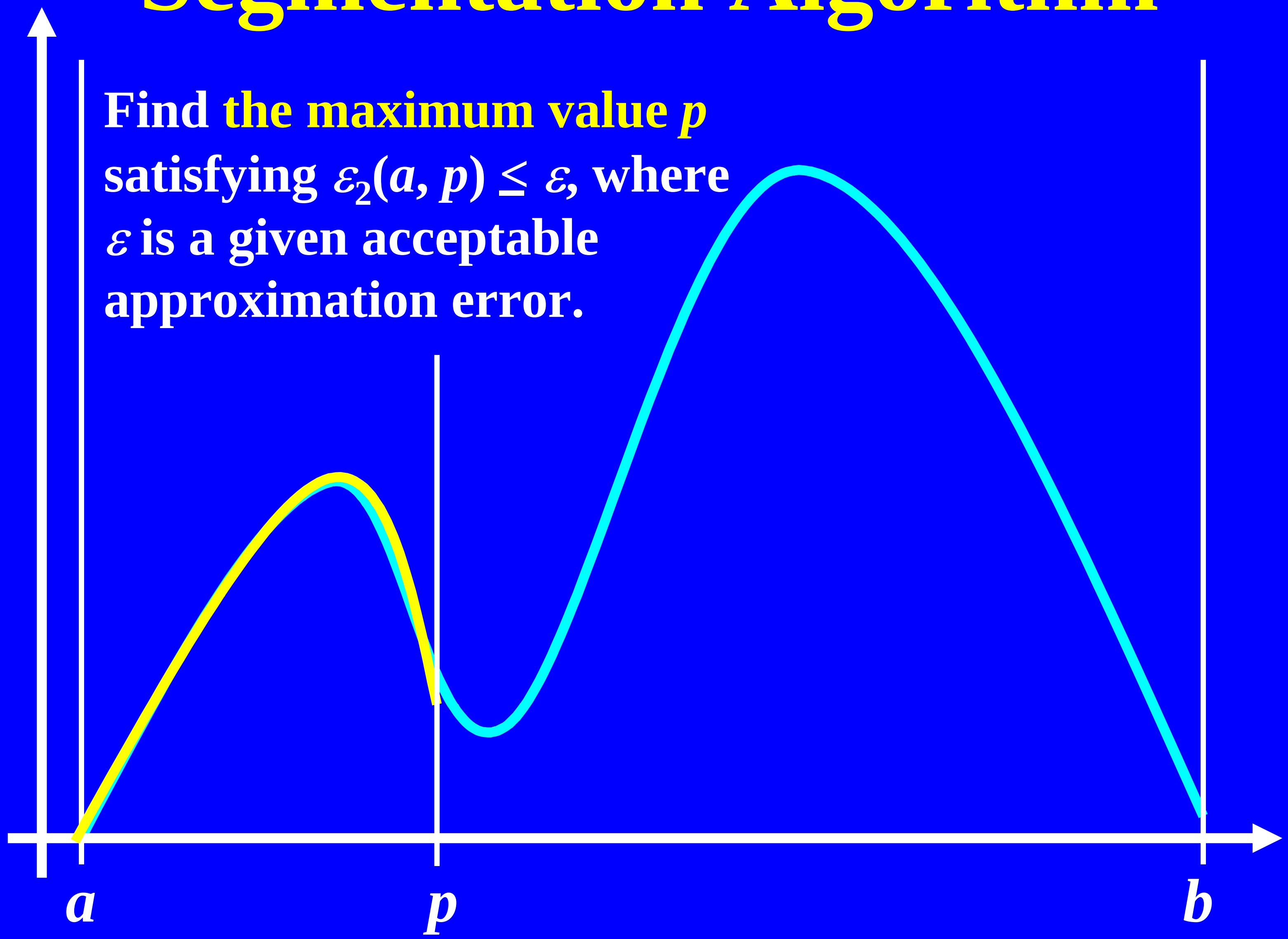
Maximum Approximation Error of 2nd-order Chebyshev Polynomial on a Segment [*a*, *b*]

$$\varepsilon_2(a, b) = \frac{(b-a)^3}{192} \max_{a \le x \le b} (| f^{(3)}(x) |)$$

$\varepsilon_2(a, b)$ is a monotone increasing function of segment width $b - a$.

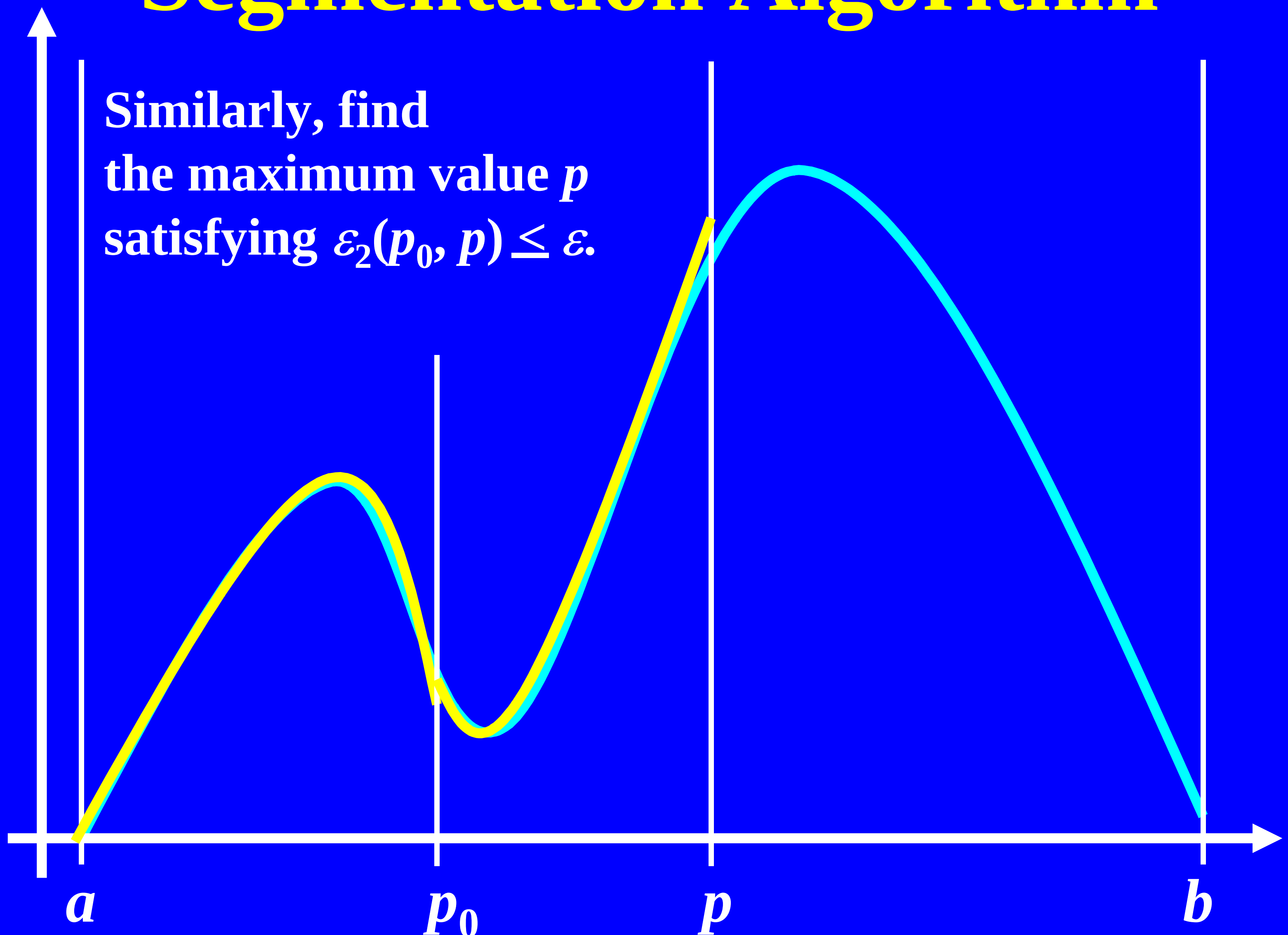
Segmentation Algorithm

Find **the maximum value** ***p*** satisfying $\varepsilon_2(a, p) \leq \varepsilon$, where ε is a given acceptable approximation error.



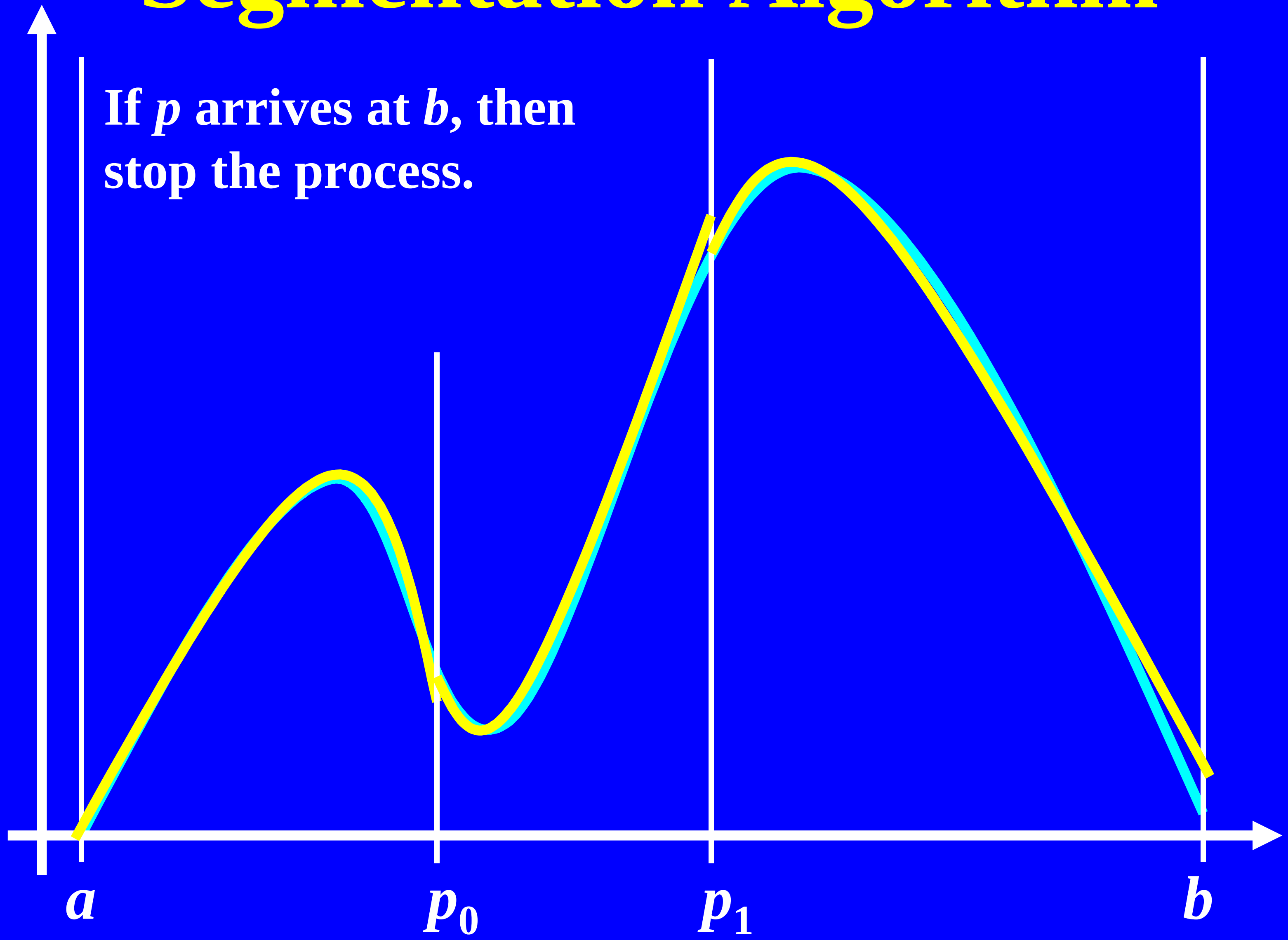
Segmentation Algorithm

Similarly, find the maximum value p satisfying $\varepsilon_2(p_0, p) \leq \varepsilon$.



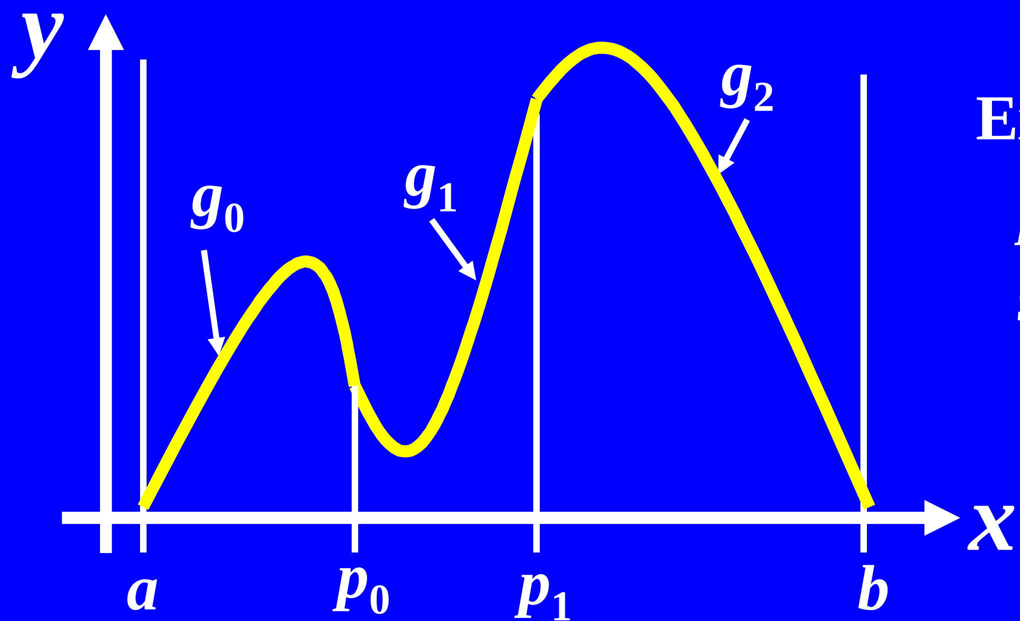
Segmentation Algorithm

If *p* arrives at *b*, then stop the process.



Computation of Approximated Value

- We select a quadratic function $g_i(x)$ depending on input x.
- $g_i(x) = c_{2\_i}\, x^2 + c_{1\_i}\, x + c_{0\_i}$ computes the approximated value y.



Example:

$p_0 < x \leq p_1$

$g_1(x) = c_{2\_1}\, x^2 + c_{1\_1}\, x + c_{0\_1}$

Transformation of Quadratic Polynomial

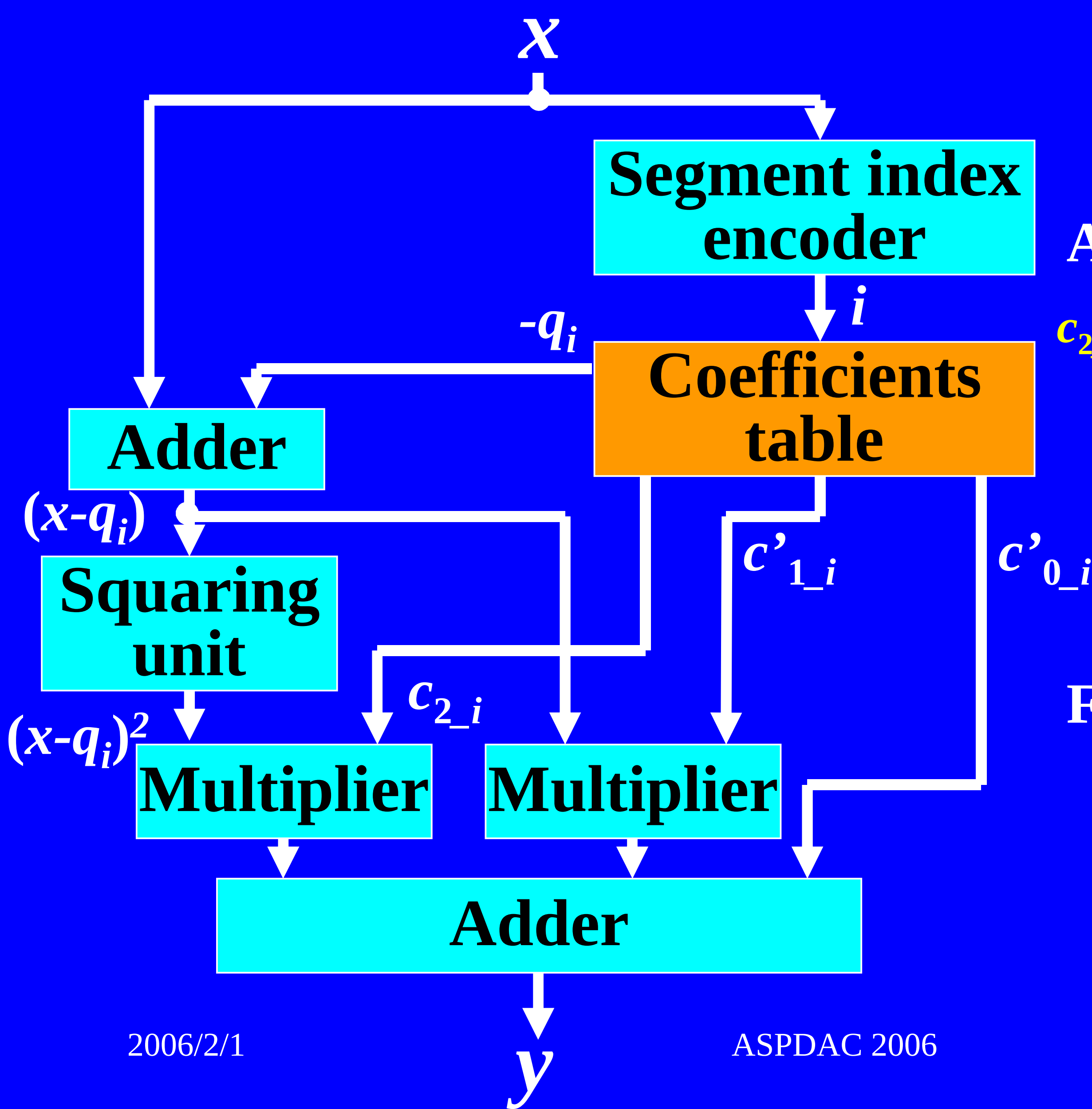
$$c_{2\_i}\, x^2 + c_{1\_i}\, x + c_{0\_i}$$

Substitute $(x - q_i + q_i)$ for x, where q_i is any value.

$$c_{2\_i}\,(x - q_i)^2 + \underbrace{(c_{1\_i} + 2c_{2\_i}\, q_i)}_{c'_{1\_i}}(x - q_i) + \underbrace{c_{0\_i} + c_{1\_i}\, q_i + c_{2\_i}\, q_i^2}_{c'_{0\_i}}$$

$$c_{2\_i}\,(x - q_i)^2 + c'_{1\_i}\,(x - q_i) + c'_{0\_i}$$

Architecture for the NFG



Architecture is based on

$$c_{2\_i}(x - q_i)^2 + c'_{1\_i}(x - q_i) + c'_{0\_i}$$

For each segment $[s_i, e_i]$

$$q_i = \frac{s_i + e_i}{2}$$

Uniform vs. Non-uniform Segmentations

Uniform

Merit

- No segment index encoder.
  - MSBs of *x* can specify an approximation function.

Demerit

- Larger coefficients table.

Non-uniform

Merit

- Smaller coefficients table.

Demerit

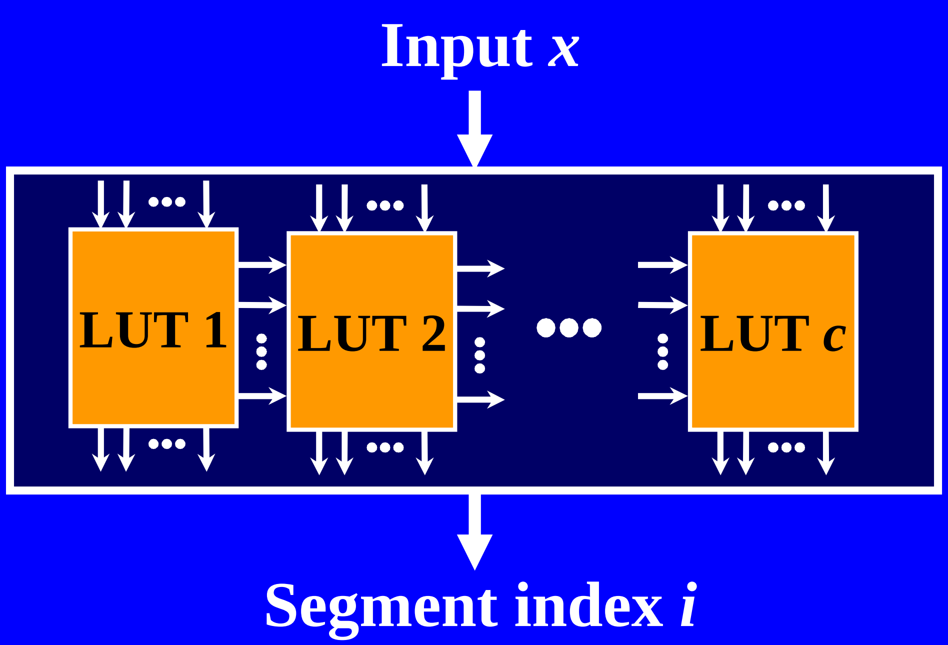
- Large segment index encoder.

Segment Index Encoder

Interval	Index
$a \le x \le p_0$	0
$p_0 < x \le p_1$	1
$p_1 < x \le p_2$	2
⋮	⋮
$p_{t-2} < x \le b$	t - 1

Segment index function

- It converts input x into segment index i.
- Single lookup table is large.
- We use **LUT cascade**.



Advantages of LUT Cascades for Segment Index Encoder

- **LUT cascade can realize any *segment index function* compactly.**
- **It allows flexible segmentation.**
  - **We can partition the domain anywhere.**
  - **This property facilitates automatic non-uniform segmentation.**
- **It is suitable for pipeline processing.**

Experimental Results

Numbers of Segments for Approximations and Segmentations

Function $f(x)$	Domain	Number of Segments		
		Linear	Quadratic	
		Non-uni.	Uniform	Non-uni.
2^x	[0, 1]	2,048	65	44
$\ln(x)$	[1, 2)	1,437	128	50
$\cos(\pi x)$	[0, ½]	2,027	129	74
$\sqrt{x}$	[1/32, 2)	3,082	2,016	138
$\sqrt{-\ln(x)}$	[1/32, 1)	5,933	8,126,464	331

Acceptable approximation error: 2^{-25}

Memory Sizes of Linear NFGs and Quadratic NFGs (Non-uniform)

Function $f(x)$	Domain	Memory size [bits]		Ratio [%]
		Linear*	Quad.	
2^x	[0, 1]	696,320	19,072	3
$\ln(x)$	[1, 2)	700,416	19,136	3
$\cos(\pi x)$	[0, ½]	663,552	38,784	6
$\sqrt{x}$	[1/32, 2)	1,425,408	86,784	6
$\sqrt{-\ln(x)}$	[1/32, 1)	2,662,400	173,056	7

* T. Sasao, S. Nagayama, J. Butler, "Programmable numerical function generators: architectures and synthesis method," in *FPL 2005*.

Acceptable error: 2^{-23}

Performances [MHz] of Linear NFGs and Quadratic NFGs (Non-uniform)

Function $f(x)$	16-bit precision		24-bit precision	
	Linear	Quad.	Linear	Quad.
2^x	195	185	--	131
$\ln(x)$	197	185	--	131
$\cos(\pi x)$	237	179	--	131
$\sqrt{x}$	237	179	--	124
$\sqrt{-\ln(x)}$	215	135	--	130

FPGA device: Altera Stratix (EP1S10F484C5)

--: Insufficient memory blocks in the FPGA.

Number of Logic Elements of Linear NFGs and Quadratic NFGs

Function $f(x)$	16-bit precision				24-bit precision			
	Linear		Quad.		Linear		Quad.	
	LE	DSP	LE	DSP	LE*	DSP	LE	DSP
2^x	167	2	482	4	604	2	758	10
$\ln(x)$	170	2	379	4	416	2	863	10
$\cos(\pi x)$	172	2	354	4	412	8	647	10
$\sqrt{x}$	270	2	496	4	1211	2	822	16
$\sqrt{-\ln(x)}$	304	2	623	10	854	8	942	16

FPGA device: Altera Stratix (EP1S10F484C5)

Total LEs: 10570, Total DSPs: 48

Comparison of Linear NFGs and Quadratic NFGs

- **Linear NFGs**
  - **are faster, and require fewer logic elements and DSPs.**
  - **require much larger memory size.**
  - **need large-scale FPGA for high-precision.**

- **Quadratic NFGs**
  - **require much smaller memory size.**
  - **are relatively fast, and require reasonable logic elements and DSPs.**
  - **realize high-precision NFGs with compact and low-cost FPGA.**

Conclusion

- **We proposed architecture of NFG based on quadratic approximation.**
  - **LUT cascade can realize any *segment index function* compactly.**
  - **Our architecture efficiently implements NFGs for a wide range of functions.**
- **Our automatic synthesis**
  - **is facilitated by the use of LUT cascade.**
  - **produces NFGs with smaller memory size than the existing method.**
  - **produces high-precision NFGs that can be implemented with a low-cost FPGA.**

Thank you for your attention!