Recursive Synthesis of Digital Circuits

Behrooz Parhami
Department of Electrical and Computer Engineering
University of California, Santa Barbara, USA
parhami@ece.ucsb.edu

Presentation on Behalf of IEEE Computer Society’s Distinguished Visitors Program
Outline

• Intro: HW Design and Recursion
• Iterative Refinement (SW & HW)
• Algor/SW/HW Recursion Example
• Example 1: Fast Fourier Transform
• Example 2: Recursive Multipliers
• Example 3: Counting Networks
  – Parallel Counters
  – Weight-Checkers
  – Threshold Counters
  – Variations in Counting
• Conclusion and Future Work
Introduction: Hardware Design

- Time- and work-intensive; expensive; error-prone
- Choice of algorithm, technology, design methodology
- Quick proof-of-concept, followed by fine-tuning
- Fine-tuning adds complexity and thus errors
- In some cases, correctness / reliability more important
- Modularity, packageability, reusability are key attributes
- Ditto for testability, serviceability, availability
- We may opt for simpler designs, older technologies
- $O(\sqrt{n})$ & $O(\log n)$ complexities about the same for small $n$
The Concept of Recursion

To iterate is human, to recurse divine!

**Textbook definition:**
Recursion is a method where the solution to a problem depends on solutions to smaller instances of the same problem.

Problem

\[
\begin{cases}
\text{Base case} \\
\text{Subproblem(s)}
\end{cases}
\]

\[
\begin{align*}
n! &= \begin{cases}
& \text{if } n \leq 1 \text{ then } 1 \\
& \text{else } n \times (n - 1)!
\end{cases} \\
\text{fib}(n) &= \begin{cases}
& \text{if } n \leq 2 \text{ then } 1 \\
& \text{else } \text{fib}(n - 1) + \text{fib}(n - 2)
\end{cases}
\end{align*}
\]
Recursive Synthesis of Digital Circuits

Recursive Dolls
Recursive Pizzas
Iterative Refinement

- Compute $f(z)$ by iteratively refining an approximation
- Example $x = \sqrt{z}$:
  $$x^{(i+1)} = 0.5(x^{(i)} + z/x^{(i)})$$

$$\sqrt{z} \approx x^{(0)} = 1.5$$
$$\sqrt{z} \approx 1 + z/4$$
$$\sqrt{z} \approx 7/8 + z/4$$
$$\sqrt{z} \approx 17/24 + z/3$$

Best linear approx.
Recursive Synthesis of Digital Circuits

Unrolling and Pipelining

- Compute \( f(z) \) by iteratively refining an approximation
- Example \( x = \sqrt{z} \):
  \[
  x^{(i+1)} = 0.5(x^{(i)} + z/x^{(i)})
  \]

- Pipelining: Five different square-rootings in progress
- No unrolling: One orange box, used four times
- Partial unrolling: Two orange boxes, used twice
**Algorithm/SW/HW Example: Selection Sort**

Algorithm:

```
selectionsort(x_0, x_1, … x_{n-1})
if n = 1 then exit
find x_j = max(x_0, x_1, … x_{n-1})
swap x_j and x_{n-1}
selectionsort(x_0, x_1, … x_{n-2})
```
Recursion Drawbacks and Benefits

- Many procedure calls, with associated overheads
- Overhead not as bad on modern hardware
- Tail-recursion: Recursive call is in last statement
- Unroll the recursion into a loop (smart compiler?)
- Partial unrolling: $n! = n \times (n - 1) \times (n - 2) \times (n - 3)!$

- May be preferred, even if not the most efficient
- In the case of hardware, the design is recursive
- The circuit-level realization is often fully unrolled
- Recursive design provides analyzability & reliability
Discrete Fourier Transform: FFT Network

DFT: \( y_i = \sum_{j=0}^{n-1} \omega_n^{ij} x_j \)

Naïve algorithm: \( T(n) = O(n^2) \)

FFT:
\[
\begin{align*}
    y_i &= u_i + \omega_n^i v_i \quad (0 \leq i < n/2) \\
    y_{i+n/2} &= u_i - \omega_n^i v_i 
\end{align*}
\]

Seq: \( T(n) = 2T(n/2) + n = O(n \log n) \)
Par: \( T(n) = T(n/2) + 1 = O(\log n) \)

Inverse DFT is almost exactly the same computation:

IDFT: \( x_i = \frac{1}{n} \sum_{j=0}^{n-1} \omega_n^{-ij} y_j \)
Recursive Synthesis of Digital Circuits

Bit-reversal permutation

Butterfly operation:
\[
\begin{align*}
  a & \rightarrow a + b \omega^j \\
  b \rightarrow a - b \omega^j
\end{align*}
\]
Regularized Butterfly: Shuffle-Exchange

Rearranged nodes, same connectivity

Removal / sharing of some links

B. Parhami (UCSB)
Recursive Synthesis of Digital Circuits

More Economical FFT Hardware
Recursive Multipliers: Concept

Building wide multiplier from narrower ones

Form the 4 half-products
Add the resulting 4 (3) numbers
Add much faster than multiply

Rearranged partial products in 2b-by-2b multiplication

\[ D(n) = D(n/2) + O(\log n) = O(\log^2 n) \]
\[ C(n) = 4C(n/2) + O(n) = O(n^2) \]
Recursive Multipliers: Example

Multipliers can be built recursively from square or non-square component multipliers

An $8 \times 8$ multiplier built from $4 \times 4$ component multipliers
Recursive Multipliers: Circuit

A 32-bit array multiplier built of four 16-bit array multipliers

The same design is valid for non-array multipliers of the additive type (square or rectangular)

2 x 4 additive multiplier
Karatsuba Multiplication

$2b \times 2b$ multiplication requires four $b \times b$ multiplications:

$$(2^b a_H + a_L) \times (2^b x_H + x_L) = 2^{2b} a_H x_H + 2^b (a_H x_L + a_L x_H) + a_L x_L$$

Karatsuba noted that one of the four multiplications can be removed at the expense of a few extra additions:

$$(2^b a_H + a_L) \times (2^b x_H + x_L) =$$

$$2^{2b} a_H x_H + 2^b [(a_H + a_L) \times (x_H + x_L) - a_H x_H - a_L x_L] + a_L x_L$$

Form the 3 half-products; do additions:

$$C(n) = 3C(n/2) + O(n) = O(n^{1.58})$$

Benefit is quite significant for extremely wide operands

$$ (4/3)^5 = 4.2 \quad (4/3)^{10} = 17.8 \quad (4/3)^{15} = 74.8 \quad (4/3)^{20} = 315.3 $$
Special Case of Squaring

Non-square building blocks not beneficial for squarers because we won’t be able to use squarers

An $8 \times 8$ squarer built from $4 \times 4$ component multipliers/squarers
Counting Networks

- **Circuits that compute (symmetric) logic functions based on the number of 1s among the inputs**
  - $?/n$ How many 1s are there? (Parallel counters)
  - $=k/n$ Are there exactly $k$ 1s?
  - $\geq l/n$ Are there at least $l$ 1s?
  - $< m/n$ Are there fewer than $m$ 1s? (or $\leq (m-1)/n$)
  - $\in [l,m)/n$ Are there at least $l$ and fewer than $m$ 1s?
  - $\in [l,m-1]/n$ Are there at least $l$ and at most $m-1$ 1s?
  - $\{j_1,j_2,...,j_k\}/n$ Is the number of 1s in the set $\{j_1,j_2,...,j_k\}$?
  - Also, Hamming-weight-comparators, not discussed here
Recursive Design of Parallel Counters

\[ D(\text{LSB } ?/n) = 1 + D(\text{LSB } ?/\lfloor n/2 \rfloor) = \lfloor \log_2 n \rfloor \]

\[ D(?/n) = D(\text{LSB } ?/n) + \lfloor \log_2 n \rfloor - 1 \]
\[ = 2 \lfloor \log_2 n \rfloor - 1 \]

\[ C(?/n) = 2C(?/\lfloor n/2 \rfloor) + \lfloor \log_2 n \rfloor \approx 2n \]
Mux-Based Hardware Realizations

Shannon expansion or decomposition

\[ f(x_1, x_2, \ldots, x_{n-1}, x_n) = x_n' f(x_1, x_2, \ldots, x_{n-1}, 0) \lor x_n f(x_1, x_2, \ldots, x_{n-1}, 1) \]

Example: Majority voter

\[ f(x_1, x_2, x_3) = x_1 x_2 \lor x_2 x_3 \lor x_3 x_1 \]
\[ = x_3'(x_1 x_2) \lor x_3(x_1 \lor x_2) \]
Recursive Design of Weight-Checkers

\[ B(=k/n) = 1 + B(=k/(n - 1)) + B(=(k - 1)/(n - 1)) \]

\[ B(=k/n) = 1 + B(=k/(n-1)) + B(=(k-1)/(n-1)) - B(=(k-1)/(n-2)) \]

\[ = k(n - k) - 1 \]

\[ C(=k/n) = B(=k/n) + \text{peripheral gates} \]

\[ D(=k/n) = n - 2 + \max(D_{\text{AND}}(2), D_{\text{NOR}}(2), D_{\text{XOR}}) \]

\[ B(=3/6) = 1 + B(=3/5) + B(=2/5) - B(=2/4) \]

\[ = 1 + 5 + 5 - 3 = 8 \]
Recursive Design of Threshold Counters

\[
B(\geq l/n) = 1 + B(\geq l/(n-1)) + B(\geq (l-1)/(n-1)) - B(\geq (k-1)/(n-1)) = (n - l)(l - 1) - 1
\]

\[
C(\geq k/n) = B(\geq k/n) + \text{peripheral gates}
\]

\[
D(\geq l/n) = 1 + \max[D(\geq l/(n-1)), D(\geq (l-1)/(n-1))] = n - 3 + \text{small constant}
\]
Example (Inverse) Threshold Counters

\[
B(\geq l/n) = 1 + B(\geq l/(n-1)) + B(\geq (l-1)/(n-1)) - B(\geq (k-1)/(n-1)) = (n-l)(l-1) - 1
\]

\[
C(\geq k/n) = B(\geq k/n) + \text{peripheral gates}
\]

\[
D(\geq l/n) = 1 + \max[D(\geq l/(n-1)), D(\geq (l-1)/(n-1))] = n - 3 + \text{small constant}
\]

Direct realization of a <m/n TPCN is beneficial for 2 ≤ m ≤ n/2
Example CMOS Implementation

Design

High-level implementation

Three possible mux designs:
Ordinary CMOS; bypass transistor; transmission gate

Average improvements:
~10% delay
~65% power
~55% transistor count
More results forthcoming
Between-Limits Threshold Counters

Example application:

\[
C(4, 9) + C(5, 9) = 126 + 126 = 252
\]

Codewords of length 9 bits and weights 4 or 5

\[
C(\in [l, m]/n) = \text{Open problem}
\]

\[
D(\in [l, m]/n) = n - 2 + \text{a small constant}
\]
Membership Checkers

\(\{3,4,6\}/8\) membership checker

Negative terms and terms larger than \(n\) are dropped
Even- or Odd-Parity Checker

Parity checking: XOR-tree networks
First attempt at recursive formulation

\[
\text{odd-parity}(x_1, x_2, \ldots, x_{n-1}, x_n) = \text{odd-parity}(x_1, \ldots, x_m) \oplus \text{odd-parity}(x_{m+1}, \ldots, x_n)
\]
Advantages and Drawbacks

• Recursion not applicable to all of our needs
• May not lead to theoretically-optimal design
• But ... Optimal designs tend to be complex
  → Long design times and many design errors
• Recursive designs: Analyzable and verifiable
• Stop recursion upon hitting a known design
• Commonly-used parts can be fully optimized
• Good for prototyping, if not for final circuit
Recursion Synthesis of Digital Circuits

Speed vs. Regularity

- Tree: Fast, but irregular
- Array: Slow, but regular

- All partial products
- Large tree of carry-save adders
- Adder
- Product
- Log-depth
- Log-depth

- Tree: Fast, but irregular
- Array: Slow, but regular
Conclusion and Future Work

- Recursive hardware design is feasible and beneficial
- I covered three examples: FFT; Multiplier; Counter
- Counting-network designs are new
- There are other examples: e.g., sorting networks
- Benefits: Ease of analysis and correctness proof
- May be preferred, even if not the most efficient
- All designs can be pipelined for higher throughput
- Latency, cost, power for implemented networks
Questions?

parhami@ece.ucsb.edu
PDF files of B. Parhami’s publications are available at:
www.ece.ucsb.edu/~parhami/publications.htm
Back-up Slides

parhami@ece.ucsb.edu
www.ece.ucsb.edu/~parhami
Insertion Sort and Selection Sort

Parallel insertion sort = Parallel selection sort = Parallel bubble sort!

\[ C(n) = \frac{n(n - 1)}{2} \]
\[ D(n) = 2n - 3 \]
Cost × Delay
\[ = \Theta(n^3) \]
Batcher’s Even-Odd Merge Sorting

Batcher’s \((m, m)\) even-odd merger, for \(m\) a power of 2:

\[
C(m) = 2C(m/2) + m - 1 = (m - 1) + 2(m/2 - 1) + 4(m/4 - 1) + \ldots = m \log_2 m + 1
\]

\[
D(m) = D(m/2) + 1 = \log_2 m + 1
\]

Cost × Delay = \(\Theta(m \log^2 m)\)

Batcher sorting networks based on the even-odd merge technique:

\[
C(n) = 2C(n/2) + (n/2)(\log_2(n/2)) + 1 
\approx n(\log_2 n)^2 / 2
\]

\[
D(n) = D(n/2) + \log_2(n/2) + 1 = D(n/2) + \log_2 n = \log_2 n (\log_2 n + 1)/2
\]

Cost × Delay = \(\Theta(n \log^4 n)\)
Example Batcher’s Even-Odd 8-Sorter

Batcher’s even-odd merge sorting network for eight inputs
(Non-)Square Recursive Multipliers

Non-square components may lead to matrix height reduction

Square components

Non-square components
Example with No Height Reduction

$12 \times 8$ multiplication
$3 \times 4$ components
Examples of Matrix Height Reduction

14 × 4 multiplication
2 × 2 components

12 × 6 multiplication
2 × 3 components