

# Arithmetic Multifunction Circuits with Recursive Structure

Sébastien Roy  
courriel: [sebasroy@gel.ulaval.ca](mailto:sebasroy@gel.ulaval.ca)

Université Laval

June 17, 2009

# Plan

- 1 Introduction
- 2 Dyadic tree functions
  - Lead-digit detection
  - Comparator
  - Fast incrementer / decrementer
- 3 Fast addition
  - Fast carry generation
  - Parallel-prefix structures
  - Brent & Kung structure
- 4 4-way multifunction circuit
- 5 Conclusion

# Combinational synthesis

- Some combinational circuits are difficult to implement efficiently and / or with a short critical path.
- Such circuits are characterized by a **wide input vector** and outputs which **depend on all input bits**.
- The classical case is standard addition where carries at each position **can depend on all input bits of lesser weight**.
- It is known that the ripple carry adder does not scale well, and its delay grows linearly with its width.
- It was proposed (Babbage, 1851) to implement two-stage SOP or POS combinational circuits (using Karnaugh maps or the Quine-McClusky algorithm) to directly generate the carries at every position.
- This is the **carry-lookahead** concept, which yields a constant number of logical levels regardless of operand width.
- However, since high-weight positions yield gates with high fan-in, thus leading to significant slowdown in e.g. CMOS circuits.

## Combinational synthesis – 2

- Automated synthesis does not typically lead to good solutions with such problems.
- From a basic truth table description, an optimal two-level synthesis process would lead to the carry-lookahead structure, with its numerous drawbacks.
- It is not clear what multilevel synthesis would generate since there is no clear optimality criterion beyond two levels.
- The natural structure of the problem would not be recognized by any general synthesis algorithm.
- Many problems, including addition, have an inherent **hierarchy** which can lead to efficient multistage implementations.
- For addition, these are the so-called **parallel-prefix** class of architectures with critical paths of  $O(\log(N))$ .

## Combinational synthesis – 3

- Finding an optimal architecture for certain circuit functions through automated synthesis thus remains an open problem.
- However, Verma, Brisk, and Lenne have proposed a **progressive decomposition** approach to impose hierarchy in the automated synthesis.
- While this seems promising, it does not reach the level of optimality possible by applying hierarchy **prior** to automated synthesis.
- Aside from the difficulty in recognizing hierarchy, automated combinational synthesis suffers from two drawbacks :
  - If the number of logic levels is allowed to exceed 2 (typically the case), no unambiguous definition of optimality exists.
  - When multiple functions are implemented, it is not clear how to identify common intermediate signals / circuit modules which could potentially be shared.

# What questions does this raise ?

- Can a hierarchical structure be found that is general enough to be shared in the realization of multiple useful arithmetic functions ?
- Could this provide insights which could be applied in the automated synthesis of such shared circuits ?

## Leading-digit detectors

- Leading-digit detection is a useful operation in many contexts.
- It is crucial in floating-point processors to support **operand normalization**.
- This involves shifting the operand to the left until the most-significant '1' (leading digit) is in in the leftmost position.
- This requires a priori knowledge of the position of the leading digit, obtained through a **lead-digit detector** (LDD).
- The operand can then be instantly shifted using a **barrel shifter**.
- However, fully-combinational design of an LDD is problematic because of the dependance on all input bits (same situation as with addition).

## Leading-digit detectors — 2

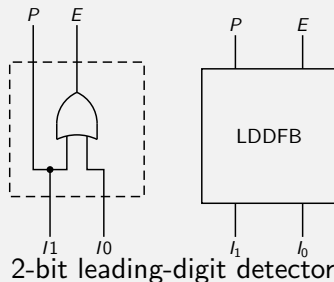
- Classical logic minimization (e.g. Karnaugh maps) would prove cumbersome with this problem.
- The similarity with addition suggests that hierarchical, **parallel-prefix-like** structures are possible.
- It will be seen that the optimal solution involves **joint** generation of all output bits.
- However, even software synthesis tools which have the capability of exploiting commonalities between many combinational functions will not recognize the **simple recursive structure** presented here.
- Okobzija and other authors have published solutions which exploit the hierarchical structure of the problem.
- However, none of these works recognize that the circuit is in fact **recursive**, self-similar across stages.

# Leading-digit detector

- To bring out the recursive structure of the leading-digit detection circuit, we start by examining the case of a 2-bit operand.

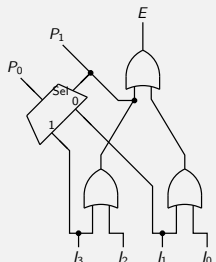
$I_1$	$I_0$	$P$	$E$
0	0	X	0
0	1	0	1
1	0	1	1
1	1	1	1

Truth table for 2-bit LDD



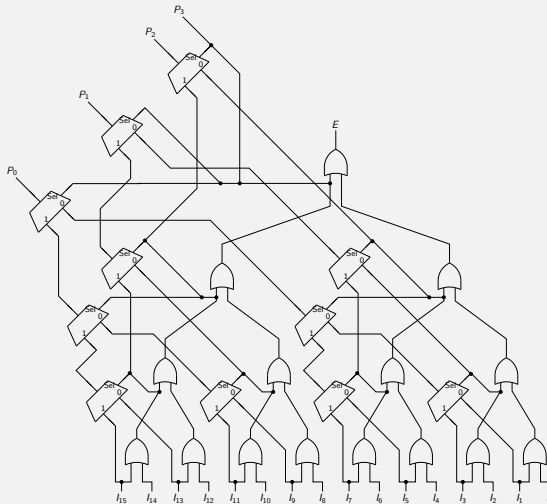
## 4-bit LDD

- Building a 4-bit LDD by replicating the 2-bit LDD and adding some glue logic.
- The glue logic amounts to one gate and one multiplexer.
- All outputs are generated with a 2-stage delay.



4-bit LDD circuit

## 16-bit LDD



16-bit LDD circuit

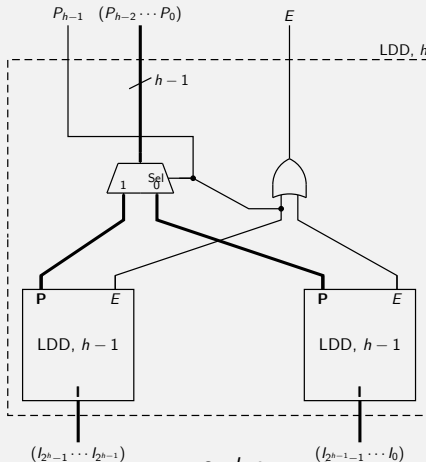
## 16-bit LDD – 2

- Outputs  $P_0$ ,  $P_1$  et  $P_2$  are generated by relatively independent multiplexer subcircuits.
- Said subcircuits are rooted in a shared OR-gate tree.
- The circuit can clearly be generalized to any depth  $h$  to handle operands of width  $2^h$ .
- All outputs are generated with a delay of  $h$  stages ; all elements have a fan-in of 2.
- Aside from the OR-gate tree, the recursive nature of the circuit is not immediately apparent.
- The OR-gate tree is a dyadic tree (comprised of two-identical height  $h - 1$  subtrees).

# Recursive structure

- The various multiplexer trees are also dyadic trees.
- However, complications arise because they take their inputs at various stages of the OR-gate tree.
- This conceptual difficulty can be lifted simply by representing the circuit as a single tree.

## Recursive structure – 2

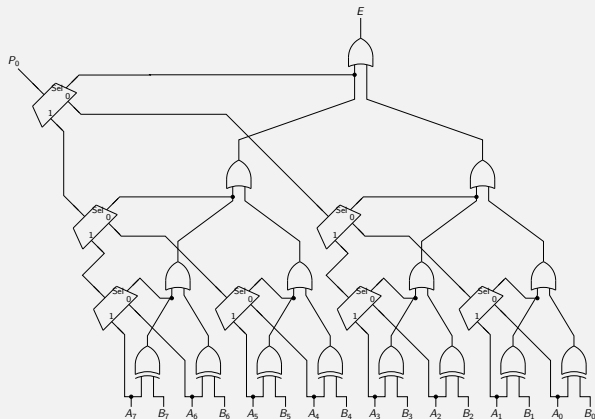


Recursive structure of  $2^h$ -bits,  $h$ -stages LDD.

# Comparator

- The same structure can act as a comparator by interlacing the two operands and keeping only the least-significant bit ( $P_0$ ) of the position output vector.
- The output  $P_0$  now indicates which operand is larger.
- The existence output  $E$  indicates whether the two operands are equal, in which case  $P_0$  becomes irrelevant.

# Comparator – 2



8-bit comparator circuit.

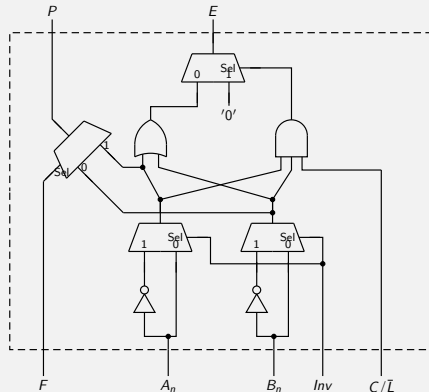
# Comparator

- Comparing this with the LDD circuit, the only difference is the foundation block and the absence of subtrees for  $P_1$  and  $P_2$ .
- The circuit works by extracting the position of the leading differing digit in the interlaced operands.
- Only the positions where the two operands differ are of interest, therefore the foundation block is a XOR gate.
- The existence output  $E$  will then indicate whether all bits are equal (in which case  $E = 0$ ).
- $P_0$  yields the least-significant bit of the position of the most significant differing digit.
- An odd position indicates that  $A > B$ , while an even position indicates that  $B > A$ .

# Generalized comparator / LDD

- A more sophisticated tree structure can be evolved by incorporating two modifiers called **flip** and **invert**.
- Both modifiers have advantages for both comparator and LDD functions.
- They will be described here in the context of a **bifunction** circuit.
- This duality in functionality affects only the foundation blocks.
- The remainder of the circuit is fully shared by both functions.

# Generalized comparator / LDD – 2

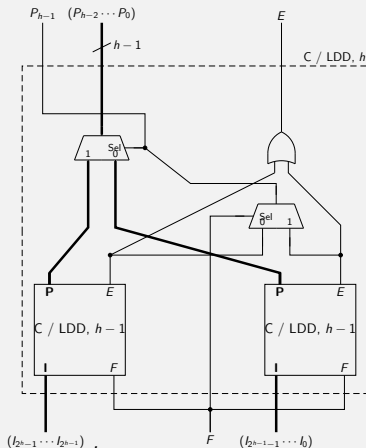


Foundation block for generalized comparator / LDD function

## Generalized comparator / LDD – 3

- Signal  $Inv$  inverts all inputs.
  - For the LDD function,  $Inv = 1$  implies that the leading zero is isolated instead of the leading one.
  - For the comparator,  $Inv = 1$  is useful to compare the magnitudes of two negative 2's complement operands.
- Signal  $F$  flips the logical weight ordering of the circuit.
- With  $F = 1$  and  $Inv = 1$ , the LDD function isolates the trailing zero position, a prerequisite for fast incrementation.
- With  $F = 1$  and  $Inv = 0$ , the LDD function isolates the trailing one position, useful for fast decrementation.

# Generalized comparator / LDD – 3

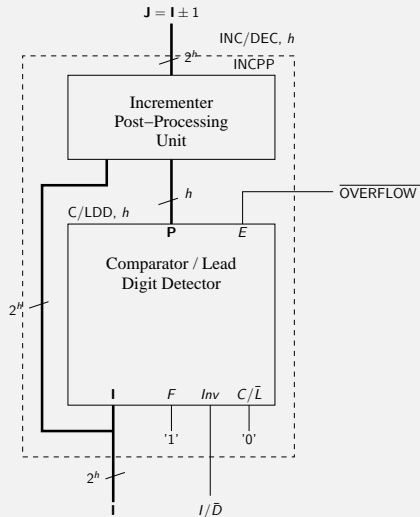


Recursive structure of  $2^h$  bits,  $h$ -stages generalized comparator / LDD circuit

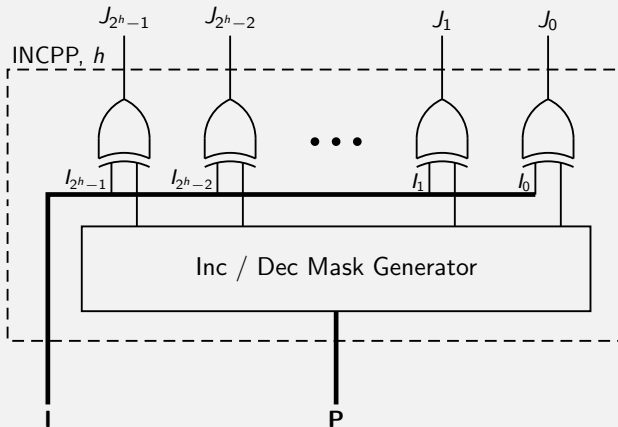
## Fast incrementer / decrementer

- In many contexts, incrementation and / or decrementation can be implemented with lower complexity than full addition.
- Indeed, incrementation is simply the inversion of a number of trailing digits :  $000 \rightarrow 001$ ,  $001 \rightarrow 010$ ,  $011 \rightarrow 100$ .
- Prerequisite : position of the trailing zero  $\rightarrow$  obtainable with generalized comparator / LDD circuit with  $C/\bar{L} = 0$ ,  $Inv = 1$ , and  $F = 1$ .
- If the operand is all '1's, existence output of LDD circuit is 0.
- This indicates overflow condition ; incrementation impossible without increasing width of operand.

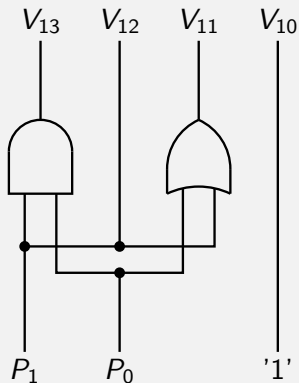
# Incrementer / decrementer circuit



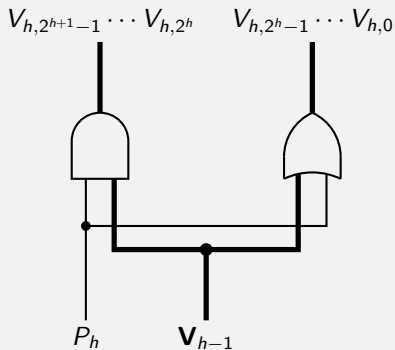
## Inc / Dec mask generator



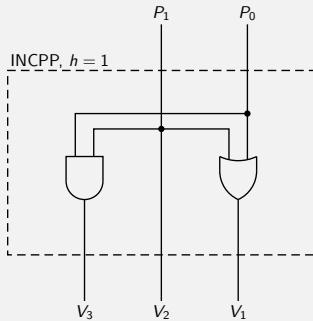
# Mask generator – hierarchical structure



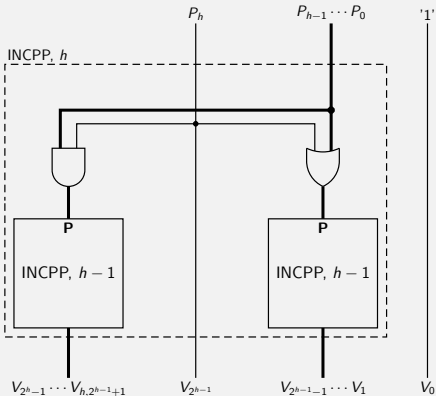
(a) First stage

(b) Stage  $h$ , where  $h > 1$

# Mask generator – alternate structure

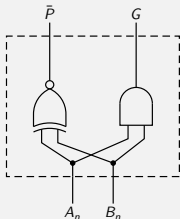


(a) First stage



(b) Stage  $h$ , where  $h > 1$

# Fast carry generation



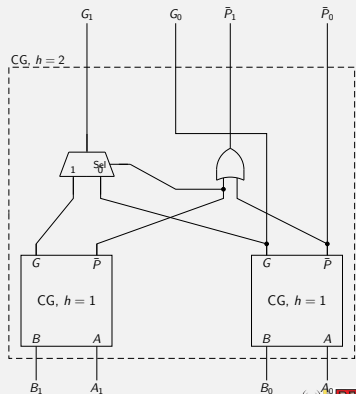
$$\bar{p}_{n1} = \overline{A_n \oplus B_n}, \quad g_{n1} = A_n \cdot B_n$$

$$g_{jk}(l) = g_{2j, k-1}(l), \quad l \in [0, 2^{k-2} - 1]$$

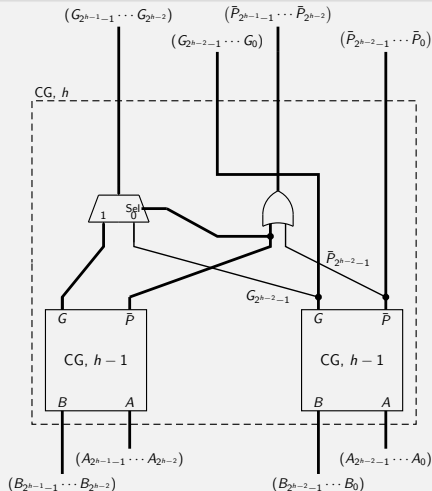
$$g_{jk}(l) = g_{2j+1, k-1}(l) \overline{p_{2j+1, k-1}(l)} + g_{2j, k-1}(l) p_{2j+1, k-1}(l), \\ l \in [2^{k-2}, 2^{k-1} - 1]$$

$$\overline{p_{jk}(l)} = \overline{p_{2j, k-1}(l)}, \quad l \in [0, 2^{k-2} - 1],$$

$$\overline{p_{jk}(l)} = \overline{p_{2j+1, k-1}(l)} + \overline{p_{2j, k-1}(l)}, \quad l \in [2^{k-2}, 2^{k-1} - 1],$$



# Fast carry generation



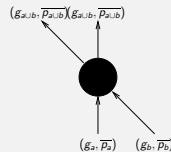
# Correspondance with parallel-prefix adders

## Classic parallel prefix

$$(g_{a \cup b}, p_{a \cup b}) = (g_a, p_a) \Delta (g_b, p_b)$$

$$g_{a \cup b} = g_a + p_a \cdot g_b$$

$$p_{a \cup b} = p_a \cdot p_b.$$



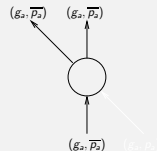
$\Delta'$  operator cell

## Proposed structures

$$(g_{a \cup b}, \overline{p_{a \cup b}}) = (g_a, \overline{p_a}) \Delta' (g_b, \overline{p_b})$$

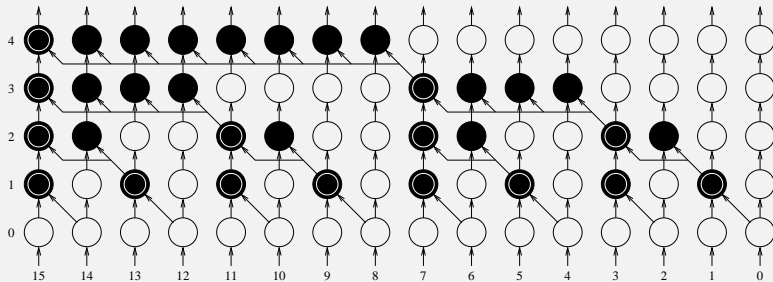
$$g_{a \cup b} = \overline{p_a} g_a + p_a \cdot g_b$$

$$\overline{p_{a \cup b}} = \overline{p_a} + \overline{p_b},$$



Flow-through cell

# Parallel-prefix structure

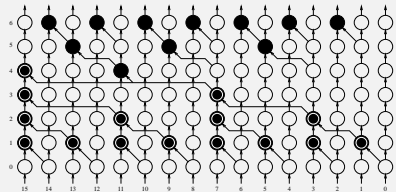


Signal-flow graph for previously described 16-bit carry generator

- This is clearly a Sklansky structure!
- Only difference is the use of  $\Delta'$  instead of  $\Delta$ .

## Alternate structure

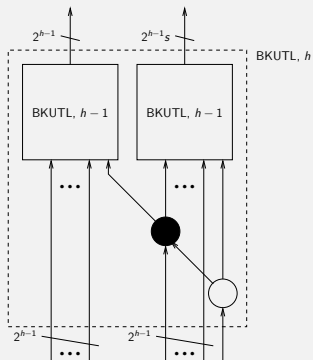
- Sklansky is parallel-prefix structure with least number of logical stages.
- However, it consumes a lot of logic and has high fan-out at the last stage.
- This leads to excessive delays and / or excessive die area in CMOS (similar to carry-lookahead)
- At the other extreme, Brent and Kung adder is PP structure with most stages;
- ... more resource-efficient, limited and regular fan-out (not a function of operand width).



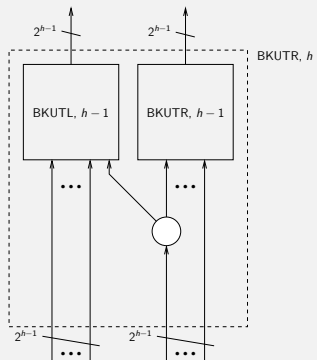
## Alternate structure – 2

- Sklansky structure was obtained by adding  $\Delta'$  nodes **between** the branches of the basic dyadic tree (common to all other functions).
- Hence, the carry generator retains same number of stages and unique recursive structure.
- Brent and Kung structure builds **on top** of dyadic tree by adding second distinct and inverted tree to generate the non power-of-two positions.
- This tree differs by 1- being inverted and 2- having different right and left subtrees.
- It can still be handled by automatic structural recursion in, e.g. VHDL.

# Brent and Kung upper tree



"Left" (most-significant) tree structure



"Right" (least-significant) tree structure

## Multifunction circuit

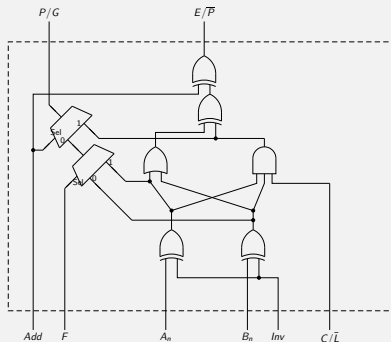
- The Generalized comparator / LDD circuit was already very versatile.

$F$	$Inv$	$C/L$	Mode
0	0	0	leading '1' detection
0	0	1	standard comparison (yields $P_0 = '1'$ if $A > B$ , $E = '0'$ if $A = B$ )
0	1	0	leading '0' detection
0	1	1	inverted comparison (yields $P_0 = '1'$ if $A < B$ )
1	0	0	trailing '1' detection
1	0	1	position of least-significant trailing digit between two operands
1	1	0	trailing '0' detection
1	1	1	position of least-significant trailing digit between two operands

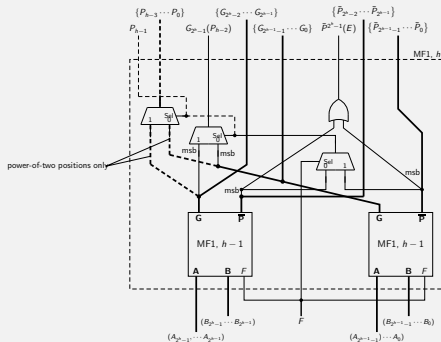
## Multifunction circuit – 2

- What if we want to add incrementation and addition?
- Is incrementation really useful in the presence of addition?
- Yes, because much simpler circuit (unused elements can be turned off for power savings).
- Adder for same operand width needs to be twice as wide.
- Incrementer has simpler post-processing unit.
- Both incrementer and adder need XOR / XNOR postprocessing at the very last stage.

# Multifunction circuit – 3

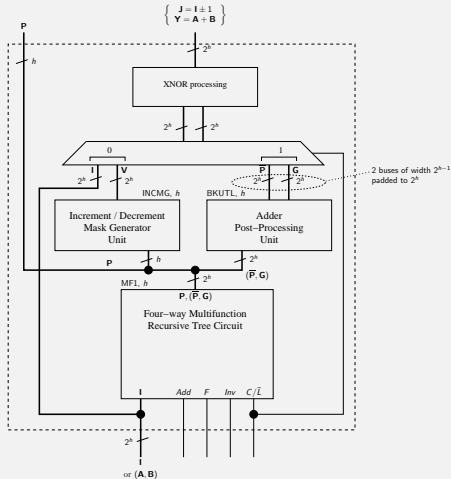


Foundation block of four-way multifunction circuit



Recursive structure of four-way multifunction circuit.

# Multifunction circuit – 4



## Conclusion

- A number of recursive circuits have been presented for fundamental arithmetic operations which are amenable to automatic scaling and synthesis through structural recursion in VHDL.
- This highlights the recursive structure inherent in many arithmetic operations which are considered to be problematic for automated synthesis.
- Said recursive structures exhibit desirable properties : scalability, minimum depth, equal depth for all outputs, constant and low fan-in and fan-out.
- The multifunction circuits show how several distinct functions can be combined advantageously in a single circuit.
- The full potential of such circuits, which have many secondary modes, has not been fully explored due to lack of space.
- The recursive approach has potential in automatic synthesis, as well as in automatic mask layout.