WAFER-SCALE INTEGRATION

THE FOUNDATIONS OF WAFER-SCALE INTEGRATION

Silicon digital integrated circuits (IC) are manufactured as small rectangular circuits on large-area silicon substrates (silicon wafers), as illustrated in Fig. 1(a). In general, each fabrication process simultaneously acts on the full area of the wafer, effectively allowing the "parallel" manufacture of a large number of ICs. Processing the largest possible silicon crystal substrate in this manner provides efficiency through which the cost of individual ICs is minimized. Implicit in Fig. 1(a) is the assumption that each IC represents only a small portion of the full circuitry appearing on the overall silicon wafer. *Wafer-scale integration* (WSI) (5–17) is a topic which reflects a wide range of issues affecting approaches to the development of ICs with circuit areas substantially larger than the normal "maximum" area of ICs. Early work (1–4) emphasized extending the monolithic circuitry during the era of only a few gates per IC, and subsequent studies reflected the continuing evolution of IC technologies.

The normal maximum area of a functional IC is limited by the nonvanishing density of defects (causing circuit faults) introduced during the manufacture of an integrated circuit (18,19). Figure 1(b)–(d) illustrates the basis for this maximum area. Here, some manufacturing defects (in this example, nine defects) are randomly distributed across the area of the silicon wafer. The shaded squares represent ICs which are defective because of the defects. If the area of the individual ICs within this full wafer of circuitry is very small [as in Fig. 1(b)], then the probability that a defect appears in any given IC is small, leading to only a few ICs with a microscopic defect damaging the circuitry. Those defective ICs are simply discarded (leading to a high yield of functional ICs). As the IC area increases [as in Fig. 1(c)], however, the probability that an IC contains one or more microscopic defects increases, leading to a more likely need to discard a fabricated IC (i.e., a lower yield of functional ICs). For very large ICs, as in Fig. 1(d), essentially all of the ICs are defective.

The yield Y is defined as the ratio of the number N_g of functional ICs to the total number N_t of ICs on a wafer, that is, $Y = N_g/N_t$. Letting A_{IC} be the area of an IC and $A_w = N_t \cdot A_{IC}$ be the area of the full wafer, then

$$Y = \frac{N_g}{N_t} = \frac{N_g A_{IC}}{N_t A_{IC}} = \frac{A_{good}}{A_w} \tag{1}$$

This merely illustrates that the yield also represents the fraction of the wafer area which produces functional ICs (i.e., from which a product can be sold to recover the cost of manufacturing). As the yield decreases with increasing area, the fraction of the manufactured wafer which can be sold as a functional product decreases, leading to an optimum area corresponding to the largest amount of circuitry which can be placed on an IC while maintaining yields sufficiently high (generally above 0.5 or 50%) to yield low prices per IC.

Figure 1(d) illustrated the case of a "large-area" IC, namely, an IC substantially larger than that complying with the defect density's bound on IC area. A defect rendering the entire IC defective is only a microscopic defect appearing as a localized defect somewhere in the IC's area. Considerable studies on the general topic of wafer-scale integration have sought to devise ways to avoid discarding a vast amount of functional circuitry because of a micron-sized defect affecting only a single (or a few) transistor. It is not possible to physically repair the defect directly. However, by applying techniques so that the defective portion of the circuitry is "cut

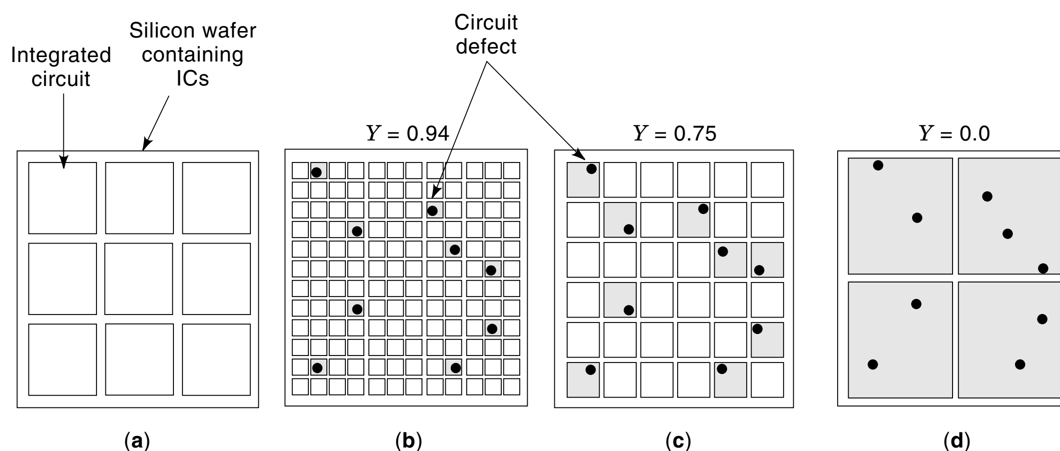

Figure 1. Integrated circuits of different sizes placed on a silicon substrate and containing circuit defects. Silicon wafers are round. The depiction here of a square substrate also represents a large area integrated circuit. (a) Nine ICs on a silicon wafer (actual numbers of ICs on wafers is in the hundreds). (b) Very small ICs, with a very high yield. Black dots indicate defects. Shaded squares indicate faulty ICs because of a defect. (c) Larger ICs with 75% yield. (d) Very large ICs with vanishing yield. Defect distributions on (b)–(d) are identical but yields differ greatly.

J. Webster (ed.), *Wiley Encyclopedia of Electrical and Electronics Engineering.* Copyright © 1999 John Wiley & Sons, Inc.

out" of the IC and "replaced" with a functional equivalent, then repair of the defective, large area IC can be considered.

Figure 2(a) shows the corresponding approach routinely used to repair printed circuit boards containing defective ICs. One simply pulls out the defective IC and replaces it with a functional IC. Figure 2(b) illustrates the application of this principle to repairing defective large area ICs and WSI components. Here, spare cells are provided on the circuit and are used to replace defective circuit cells. When a cell fails (as illustrated by the shaded cell), the connections to that cell are broken and connected instead to an available spare cell. To achieve this capability, the large area IC is divided into small circuit cells, each cell serving as the replaceable unit and with a cell area generally sufficiently small to achieve a high cell yield. In addition, spare identical cells are added to the circuit to serve as replacements for those cells which fail.

Such replacement of a defective portion of IC circuitry with a spare replica of that circuitry, in fact, is widely used today in "repairing" faulty Dynamic Random Access Memory (DRAM) ICs. A typical DRAM IC is a matrix of storage cells organized as rows and columns [Fig. 2(c)]. The overall DRAM address is divided into row and column addresses. The row address is fed into a decoder which generates a signal selecting the desired row of the memory cell array. All data stored in that row are read by the sense amplifiers and the desired data bit (corresponding to the column address) is selected according to the column address decoder. As shown in Fig. 2(c), the overall set of memory cells is organized as a large number of thin columns. Additional spare columns are provided in the event that a defect appears in one of the memory columns. If a memory cell is defective in the memory array, then the entire column containing that memory cell is "deleted." By adjusting the column output selector to read the spare column rather than the nominal column, the spare column "replaces" the defective column. This general approach to fabricating large capacity DRAM ICs has been used since the introduction of 64 Kb DRAMS several years ago (20–22). Various large area memory circuits have been explored (23–25), drawing on this fault tolerance capability.

The previous examples of large-area ICs extend even to larger area ICs by providing for repair of a larger number of defects, leading eventually to a single circuit implemented on a silicon wafer. In this general sense, wafer-scale integration is merely a special case of large-area, defect-tolerating ICs, and the area of the IC is increased to that of the full usable silicon substrate area. For this reason, the terms "large-area IC" and "wafer-scale integration" are not distinguished in this article.

The major themes appearing in investigating large-area ICs and WSI are generally as follows:

1. The area of the IC is so large that defect-free circuitry cannot be assumed. Instead, mechanisms for tolerating defects within a circuit must be provided.
2. A defective cell is replaced by a functional cell primarily by modifying the signal interconnections within the overall circuit. In some limited cases, selector switches

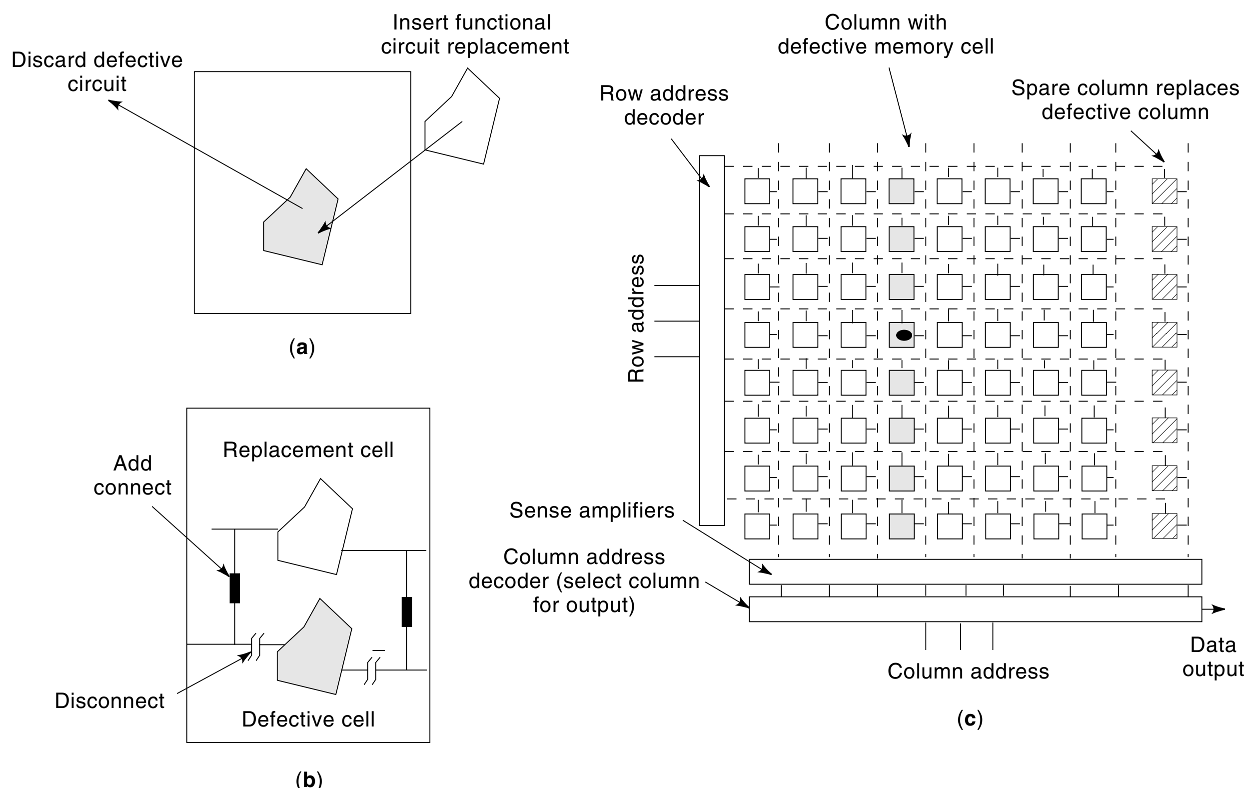

Figure 2. Replacement of a defective component with a spare. (a) Physical removal and replacement of defective circuitry (e.g., printed circuit board). (b) Use of a spare circuit to replace a defective circuit on an IC. (c) Example of replacing a column of memory cells having a defective cell with a spare column.

are combined with multiple copies of a circuit cell acting on the same inputs to choose a correct output.

3. The overall circuit is constructed from a number of small circuit cells of one or more types. Spares of each type are provided to replace defective cells of the same type.
4. A realistic model of defects circuitry within a silicon wafer is necessary to develop suitable architectures which tolerate defects. For example, how large can the individual cells be to achieve a reasonable yield of the replaceable cells? Given the areas of the cells, what fraction of cells will be defective (and what are the statistical variations in this fraction)?
5. Circuit defects must be detected and isolated to convert a circuit with defects into a functional circuit. Detection determines whether a conversion is needed. Isolation determines what is to be replaced. The defects themselves are not corrected. Rather, circuit blocks containing defects are replaced. Testing need not determine the type of defect causing an electrical fault (or the specific device which is faulty). Instead, all that is required is to determine whether, for whatever reason, a replaceable cell is not behaving correctly.

DEFECT AND FAULT MODELS

Studies of wafer-scale integration (and studies related to the optimum sizing of normal integrated circuits) have established a rich literature related to the statistical distribution functions of defects within circuits. The defect models range from simple models based on random and independent placements of point defects within the area of a silicon wafer to sophisticated models incorporating clustering and spatial nonuniformities as discussed later. Considerable evaluation of the manner in which physical defects affect the electronic performance of a circuit have led to a range of circuit fault models, ranging from simple models based on signal lines "stuck-at" logic values of 1 or 0 to fault models based on CMOS circuit failures leading to weak signal levels and to fault models including timing faults and parametric defects. The general topics are treated briefly in this section.

Physical Defects and Circuit Faults

A *physical defect* is a deviation of the physical structure of an IC circuit element from the nominal range of structures observed under correct processing (and including the unavoidable variations in processing conditions observed in a real manufacturing facility). Examples include opens in interconnection lines, shorts between adjacent lines (on the same interconnect layer or on adjacent layers of interconnection), resistive vias and contacts, leaky gate oxides preventing proper switching of an MOS transistor, gate contact shorted to the source or drain of a MOSFET, defects in the source/drain regions leading to leaky OFF states of a transistor, a wide variety of defects in patterned structures caused by particulates appearing during lithography (e.g., a particulate affecting the integrity of the patterned resist), contaminant and particulates in processing chemicals and gases, residual material from earlier processing steps affecting subsequent processing steps, defects within the crystalline substrate itself, etc. The term "manufacturing defect" is used in this article to represent such physical defects, and the term "defect" is used interchangeably with "circuit fault."

Faults are the manifestation of manufacturing defects on the behavior of the circuit in which the defect appears (26,27). In some cases, the fault is represented rather easily. Figure 3(a) illustrates various forms of "stuck-at" faults: an input connection to function B shorted to the supply voltage, leading to a "stuck-at-1" fault; an internal interconnection shorted to ground, that is, a "stuck-at-0" fault. Also shown in Fig. 3(a) is a defective circuit driving an input to A, which displays an output signal permanently stuck at either 0 or 1. Such static electrical faults are relatively easy to evaluate in terms of the functional change performed by a logic circuit.

Other faults lead to nonstandard voltage levels, complicating the evaluation of the effect of the fault because the logic functions assume logic 1 and 0 voltage levels. Figure 3(b) illustrates a basic CMOS circuit whose pull-down section has a defect which creates a permanent resistive path to ground. When the inputs are such that the pull-down circuit is normally turned ON, the correct static output level 0 is obtained. However, when the inputs are such that the pull-up circuit is turned ON (and therefore the pull-down circuit is normally in a high-impedance OFF state), the output voltage level divides according to the series combination of the pull-up circuit's ON-state resistance R_p and the resistance R_d of the defective pull-down circuit (i.e., the output settles to a voltage level $V_{dd}R_d/(R_p + R_d)$ which, if $R_n \approx R_p$, is midway between the expected logic voltage levels of 0 V and V_{dd}. Depending on the relative values of R_n and R_p, the output voltage may be a "weak" logic zero or a "weak" logic one. Quite different results from logic circuits receiving such nonstandard voltage levels occur with sensitivity to the ratio R_d/R_p.

The previous faults are "static faults," in the sense that they affect the static logic levels in the circuit. With clock rates of ICs increasing toward 1 GHz, timing faults are increasingly important. *Timing faults* may be the result of an unexpectedly large resistance appearing in the line transmitting a signal (e.g., a high-resistance via) and several other effects. Beyond the timing faults are *parametric faults,* where the device works generally as expected but has specific electrical parameters outside the nominal range of values which provide reliable circuit performance. The distinction among the various types of faults does not, in itself, affect the repair approaches used in large area IC and WSI components. However, the different fault types affect testing techniques considerably.

Defect Probability and Distribution Models

As noted earlier, clearly understanding defect/fault statistics (28,29) is an important element of WSI design, which affects the selection of areas of the replaceable cells used in the WSI circuit. The simplest defect model assumes independent, randomly distributed defects across the area of the wafer. Consider a silicon wafer of area A_w with an average defect density of D defects/cm^2. Then the average number N_d of defects on the wafer is $N_d = D \cdot A_w$. The wafer is fabricated with cells of area A_{cell}, giving a number $N_{cell} = A_w/A_{cell}$ of cells. Assume that A_{cell} is sufficiently small that $N_{cell} \gg N_d$, in which case the prob-

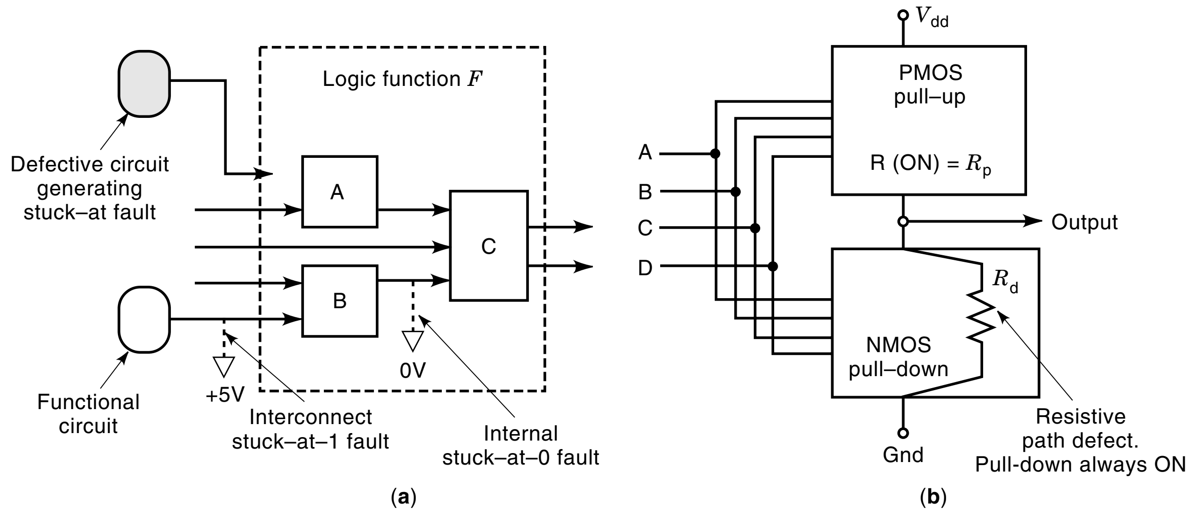

Figure 3. Examples of defects. (a) Stuck-at 0/1 faults including (i) input faults caused by a faulty preceding circuit and a faulty input interconnect and (ii) an internal fault. (b) Generation of incorrect output voltage due to a resistive path through a pull-down section of a CMOS circuit.

ability of multiple defects in a cell is negligible (and the number of defective cells equals the number of defects). In this case, the yield, defined as the ratio of good cells to total cells, is given by

$$Y = \frac{N_{\text{cell}} - N_{\text{d}}}{N_{\text{cell}}} = 1 - D \cdot A_{\text{cell}} \tag{2}$$

For $A_{\text{cell}} \ll D^{-1}$, the yield is close to unity. As $A_{\text{cell}} \to D^{-1}$, Eq. (2) indicates that $Y \to 0$. However, this approximation does not properly represent the case in which more than one defect appears in a cell. In particular, Eq. (2) assumes that each additional defect causes one additional cell to become defective, even if that additional defect appears in a cell already made defective by a previously placed defect. The basic yield equation, considering multiple defects in a single cell, is

$$Y = \exp(-D \cdot A_{\text{cell}}) \tag{3}$$

Figure 4 shows the variation of yield with the product $D \cdot A$. The linear plot illustrates the variations in the region $D \cdot A \approx 1$ and the logarithmic plot highlights the rapid decrease in yield as the area increases above D^{-1}. Detailed studies of yield on manufacturing lines suggest that Eq. (3) overestimates the rate at which the yield decreases with increasing area above D^{-1}, a detail not considered here.

Various straightforward refinements to the exponential model in Eq. (3) represent a variety of practical effects. For example, if the full area of a cell contains only a small area containing electronic devices and the remaining area is

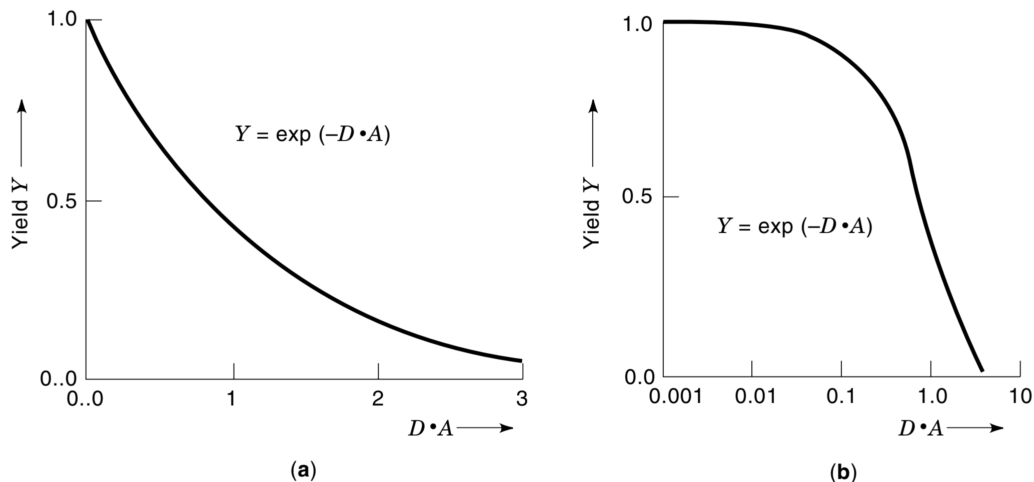

Figure 4. Decrease of IC yield with increasing area A of IC, given a defect density D. (a) Linear plot showing the effect of scaling at larger sizes. (b) Logarithmic plot showing rapid fall-off of yield for $D \cdot A > 0.2$.

empty, then the simple defect models in Eqs. (2) and (3) do not reflect the fact that only part of the wafer area should be considered. Letting $A_{\mathrm{cell}}^{(\mathrm{c})}$ be the critical area (area containing circuitry) within a cell, the equations corresponding to Eqs. (2) and (3) are $Y = 1 - D \cdot A_{\mathrm{cell}}^{(\mathrm{c})}$ and $Y = \exp(-DA_{\mathrm{cell}}^{(\mathrm{c})})$, respectively.

Often the probability of defects in areas containing only metal interconnections is substantially smaller than in areas containing a high density of transistor devices. Multiple defect densities D_i, where the subscript i represents a defect type (or a specific patterned feature), and the circuit area A_i susceptible to this defect type (or containing this specific feature) are used to refine the simple defect model in such cases. The overall yield is represented by a composite yield equation. For the model in Eq. (3), for example,

$$Y = \prod_{i=1}^{K_t} Y_i = \exp\left(\sum_{i=1}^{K_t} D_i A_i\right) \qquad (4)$$

The previous assumption of independent, random distributions of defects across the surface of the wafer is also relaxed. Two examples that significantly affect the development of a WSI circuit architecture are defect clustering and spatial nonuniformities in the defect distribution on a wafer.

Defect Clustering. Depending on the mechanism causing a defect, a defect in a region of a wafer increases the probability that another defect is produced nearby. The result is a clustering of defects, and some regions of the wafer have a higher density of defects than others. This becomes a serious issue in developing architectures that allow altering the circuitry to bypass defective elements. To avoid excessively long interconnections when connecting a spare cell to replace a defective cell, it is important to have nearby, functional spares available. However, significant clustering leads to a potential need to separate the spare spatially from the cell to be replaced and to use longer interconnections and accept slower speeds in the reconfigured circuit.

Spatial Nonuniformities in Defect Distribution. The defect density generally varies significantly across the surface of a wafer. Often, defect densities increase near the edges of the wafer, reflecting, in some cases, damage due to handling of the wafer and in other cases less accurate rotational alignment of masks at the edges, relative to the center. The transition to step-and-repeat lithography (in which patterns are defined by exposing different sections of the wafer sequentially, rather than at the same time as with earlier contact mask techniques) has lessened the latter problem.

Overall, defect and fault modeling have been studied extensively within the field of WSI circuits, yielding results of interest to WSI designers and also to designing and fabricating conventional ICs.

ARCHITECTURAL APPROACHES: LOCALLY CORRECTABLE CELLS

One of the richest contributions of WSI research to the literature is in the broad area of architectures suitable for realization on large area circuits containing defects. In some cases, the advances are driven by computational algorithms which lead to regular arrays of processors and regular data flow. In other cases, the architectural issues introduce interesting formal issues. This section and the next review several general approaches and rely often on examples.

The two primary techniques for correcting WSI circuits with circuit faults are illustrated in Fig. 5. Figure 5(a) illustrates placing repairable cells within a fixed and unchangeable network of cell-to-cell interconnections. In this case, capabilities are provided within each of the cells to ensure that the overall set of cells are viewed (from the perspective of the overall circuit) as fault-free cells by locally repairing the cell. This section reviews approaches for such locally correctable cells. As suggested in Fig. 5(a), this approach supports a heterogeneous set of circuit cells, with few global constraints on the architecture, such as found in the second

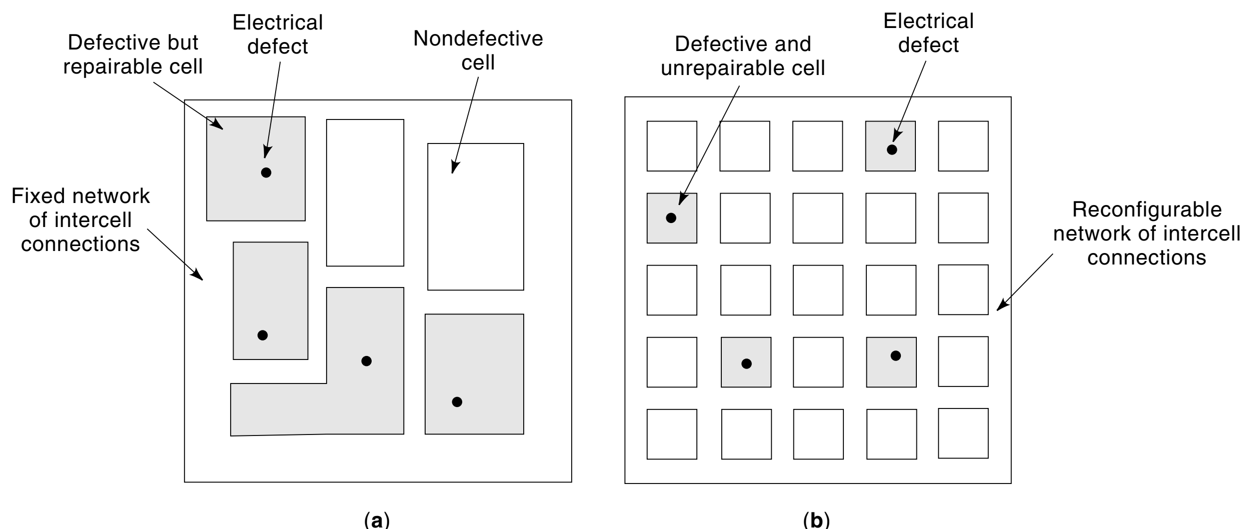

Figure 5. Two basic approaches for correcting faulty circuits. (a) Repairable cells (possibly of different functions) embedded in a fixed network of cell-to-cell interconnections. (b) Unrepairable cells (generally of same type to allow use as replacement for defective cell) embedded in a reconfigurable network of cell-to-cell interconnections.

approach, shown in Fig. 5(b). The primary approaches, considered below, are error detection/correction coding, modular redundancy, and local sparing/reconfiguration.

The second technique, discussed in the next section and illustrated in Fig. 5(b), uses unrepairable cells placed on a reconfigurable network of interconnections. In this case, the cell-to-cell interconnections in the overall circuit are modified to allow deleting a faulty cell and replacing it with a functional spare cell.

Error Detection/Correction Coding

For some arithmetic functions and simple data storage/routing functions, N_d-bit data are coded into longer data words (e.g., N_c-bit data) so that errors introduced in the data lead to data codewords that are not in the allowed set of correct codes. Error correction codes (30,31) are most often encountered in data transmission systems, where data errors introduced during transmission is corrected at the receiving end.

The two coding techniques most often considered for WSI circuits are based on parity and check sums. Parity techniques count (modulo 2) the number of binary 1 values in the data. A single error changes the parity, allowing detection of an erroneous bit (but normally not allowing identification and correction of the erroneous data bit). Check-sum techniques, similar in spirit, involve calculating the sum of all of the data values in a stream of data. When received, the check sum is recomputed and compared to the received check sum to determine whether an error has occurred. As in the case of the basic parity check, the basic check-sum approach does not allow locating and correcting the erroneous data value.

A straightforward extension of the parity (check-sum) approach adds the capability of identifying and correcting the specific error. Figure 6(a) illustrates the technique. Here, the binary digits (for parity check) or data values (for check-sums) are organized as an array. An extra column is added. Each element of the added column contains the parity (or check sum) of the corresponding row. In addition, an extra row is added, and each element of the added row contains the parity (or check sum) of the corresponding column. The bottom right-side added entry in Fig. 6(a) is the parity of the added column's elements above that entry (and also of the added row's elements to the left of that entry). When a single entry of the array of data becomes faulty, the corresponding entry in the added row and column do not agree with the recomputed values of the parity (check sum) of the row and column containing the faulty data (as indicated by the dark shaded entry in the added row and added column). The difference between the received value of the parity (check sum) and the recomputed value of the parity (check sum) provides the needed correction to restore the correct value of the erroneous data value. The technique detects and corrects any single error in the data array.

The check-sum approach previously described extends to an interesting class of computational functions, namely, sev-

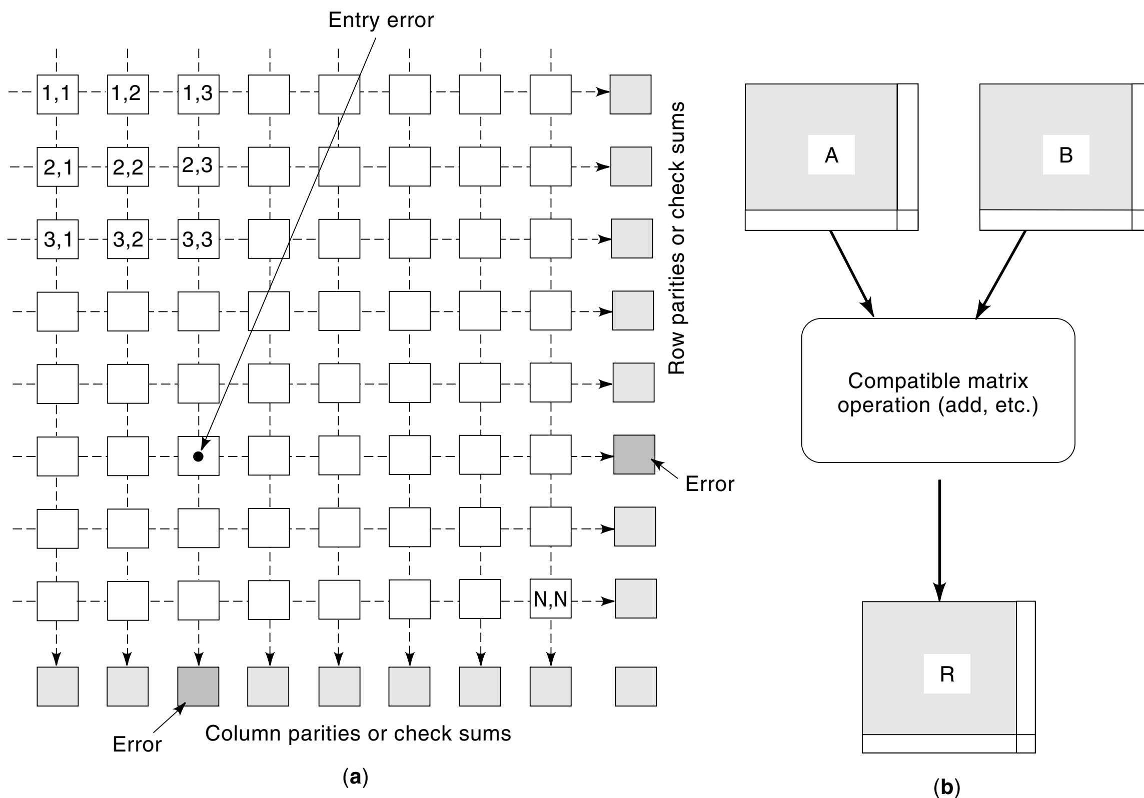

Figure 6. 2-D parity and check-sum error detection/correction. (a) Expansion of a set of data values (binary giving parity check or integers giving check-sum) into an $N \times N$ array of data values with an added column (row parity or check-sum) and added row (column parity or check-sum). (b) Augmenting data matrices with additional check-sum column and row to allow application of check-sum technique following matrix operation on data.

eral matrix operations. In this case, illustrated in Fig. 6(b), the matrix operation (e.g., sum of two matrices, product of two matrices, etc.), using the check-sum expanded data matrices as inputs, produces a new data matrix as a result, and the check-sum row and column correctly represent the resulting matrix. Matrix operations under which this condition holds have been reviewed (32,33), including consideration of techniques to handle roundoff errors in the matrix computations.

Error detection/correction coding for logic cells have seen only limited use in WSI studies, though such coding techniques are widely used in local area network (LAN) and other data networking protocols.

Triple Modular Redundancy for Self-Testing/Correcting Cells

Modular redundancy techniques typically use two or more copies of the same function, each receiving the same inputs, combined with a comparison of the outputs from those copies to determine whether different results are produced by the presumably identical copies. If different results are detected, then it is known that a copy is faulty. If three or more copies are used, the correct result is implied when the same result is produced by more than one of the copies. A wide range of modular redundancy techniques has been investigated. Here, only the well-known *triple modular redundancy* (TMR) approach is discussed. TMR is best known for its extensive application to highly reliable computers (34) but is also of interest for WSI because it provides self-testing of a circuit cell and automatic generation of the correct output if one of the replicated elements is faulty.

The basic TMR architecture is illustrated in Fig. 7(a) where the overall cell implements some digital function F. Three copies (A, B, and C) of the circuit that implements that function are provided, each receiving the same input data. The outputs of the three circuits are compared by the voter. The correct output is defined as that produced by the majority of circuit copies (i.e., produced by two or three of the circuit copies). If one of the circuit copies produces an output in conflict with the output determined by the majority, the voter provides a control signal to the 3 : 1 demultipexor (selector) to ensure that the majority determined from the circuit copies is routed to the overall output of the cell. The circuitry overhead for triple modular redundancy is at least a factor of 3 (i.e., rather than one cell function, three are required). The overhead is greater than three when the voter and selector circuitry are included. The issue of the failure of a voter or of the selector circuitry naturally arises, and approaches to designing these elements to ensure correct operation despite errors in the voter and/or selector have been studied extensively (e.g., self-checking checkers, etc.).

It is necessary to design the circuit functions [e.g., A, B, and C in Fig. 7(a)] to have a sufficiently high yield, so that no more than one of the circuit copies in *any* of the several TMR cells of the WSI circuit is faulty, because the TMR approach cannot determine the correct output if two of the cells are faulty. The result is the need for a very high yield for each of the circuit cell copies and a correspondingly small area. With such high yields for each of the circuit cell copies, most TMR cells have all three circuit copies working correctly, and the added factor of 2 overhead for triplication is used only occasionally among the TMR cells of the WSI component.

Modular Redundancy with Externally Applied Selection

The overhead of the TMR approach is considerably reduced if the capabilities for self-testing and self-correcting action are not needed. In the case of WSI components, pretesting cells identifies which of the cells are not functioning correctly, allowing the voter to be removed and only a single spare circuit used. Figure 6(b) illustrates a simple example of this approach, developed directly by simplifying the TMR approach in Fig. 6(a). Once externally tested, any bad cells are identified and the selector switch is set externally to put out the correct result. In this case, the use of three identical cells, necessary for error detection, is reduced to two identical cells if the probability that both cells are defective is sufficiently small. If this is not the case, then the number of replicated cells in the redundant cell is increased, allowing larger area basic cells (and an increased overhead in area and speed).

If each circuit cell is accompanied by a dedicated spare [as in Fig. 6(b)], then the overhead is approximately 100%, as opposed to the 200% overhead imposed by the TMR approach.

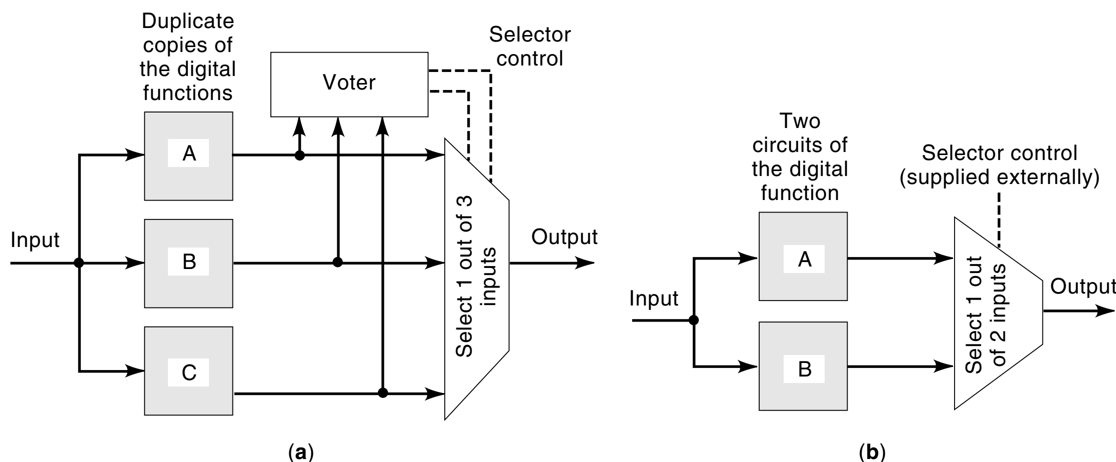

Figure 7. Modular redundancy, selecting good output with a selector. (a) Conventional triple modular redundancy (self-testing and self-correcting). (b) Dual redundancy using external testing and external setting of selector.

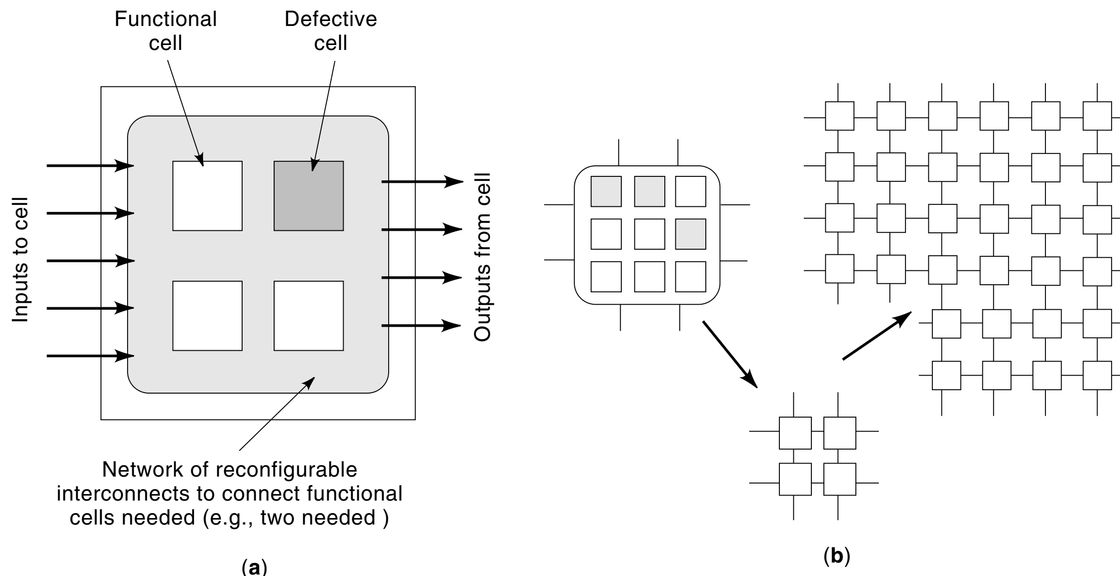

Figure 8. Repairable cells of a circuit with programmable interconnections and local spares of functions. (a) General approach. (b) Example using a 3 × 3 array of processors to construct a functional 2 × 2 element which can be inserted into a large array implemented in the overall circuit.

The overhead is reduced further by sharing a spare cell among more than one WSI component cell, using a selector with more than two inputs. A nice example of such techniques for reducing overhead is seen in the approaches for *k-out-of-m* redundancy for fault-tolerant binary trees (35).

Local Sparing with Reconfigurable Interconnections Within Each Cell

In the next section, techniques to perform a global reconfiguration of a WSI component's intercell connections are described. Here, a technique is presented for localizing the reconfiguration to within a circuit cell. The local reconfiguration approach is illustrated in Fig. 8(a). In this example, four cells are combined into a "supercell" in which connections among the four cells can be reconfigured. The network that provides connections between supercells is fixed (not reconfigurable), requiring that a faulty supercell be converted to a functional supercell. The overhead associated with use of such supercells depends critically on the specific WSI architecture implemented. For a linear array of processors, using the supercell in Fig. 8(a), all three functional cells in this example can be connected in a linear chain and used to construct the overall linear processor array of the WSI component. In this case, the overhead is simply the number of defective cells throughout the WSI component relative to the total number of functional cells.

However, in other cases the overhead is substantial. Figure 8(b) shows a WSI target architecture as a 2-D processor array. To construct this 2-D array, supercells containing nine processors are used. Each of the supercells is reconfigured using functional cells to implement a 2 × 2 array of processors. Then the reconfigured supercell is placed in the WSI component's overall 2-D processor array using the fixed interconnective network among supercells, as shown. In this example, at least four of the processors in the supercell must be functional. Otherwise, the overall 2-D array of the WSI component cannot be constructed. However, if more than four processors are available, then those additional processors cannot be used in the target WSI array. This produces the same issue noted earlier in the discussion of TMR. In particular, the processor cells must have sufficiently small area that the yield is not lower than 4/9 (i.e., four good processors of the nine in the supercell) for *any* of the supercells of the WSI component, which ensures that there are unused processors in the supercells in the vast majority of cases.

Reconfiguration with such internally reconfigurable supercells has a rich history, extending from several early formal studies [e.g., (36,37)] to more applied investigations.

GLOBAL RECONFIGURATIONAL APPROACHES

The overhead associated with the local approaches discussed in the previous section is largely caused by the need to provide spare cells which are not used (i.e., the spare used in modular redundancy imposes at least a 100% overhead even though only a few of the cells replicated are defective). Global reconfigurational approaches provide a few spare cells which are used to replace defective cells throughout the overall circuit. For example, if the overall circuit contains 100 cells and, with high probability, the overall circuit has no more than four defective cells, then only four spare cells are required to replace defective cells and achieve a high yield of repaired WSI components. This 4% overhead contrasts sharply with the 100% or greater overhead seen in the local redundancy approaches of the preceding section.

Arrays of identical cells provide a particularly suitable architecture for global reconfigurational techniques. In this case, each cell is surrounded by cells of the same function which can be used as replacements, minimizing interconnection line lengths to the replacement cell. However, as soon as

a neighboring cell replaces a defective cell, that neighboring cell is consumed and, for a regular array of interconnected cells, must be replaced itself by another cell. As a result, the replacement process initiated at a defective cell propagates outward like a wave, leading to the need to significantly alter interconnections throughout a large portion of the overall circuit. Figure 9 illustrates this effect for a mesh-connected array embedded in a square array of cells. Only a single cell (the cell shaded black) is defective. To embed a mesh-connected array, the connections normally made to the defective cell must be warped to connect to another cell, as shown in both examples in Fig. 9. In addition to the defective cell, the connections of the shaded cells (though functional) must be altered to complete the target array. Figure 9 also illustrates a typical result, namely that there is more than one reconfiguration of interconnections leading to the desired target array. Five of the 36 processors are unusable in Fig. 9 because a single defective processor reduces the maximum size of a target square array from 6×6 to 5×5.

The previous example of an array of processors has played a prominent role in the investigation of WSI circuits. In part this reflects the practical importance of computational engines for matrix computations, signal processing, image processing, and video processing. Many algorithms for such computational problems typically are well matched to parallel processing techniques, reduce to very regular arrays (linear and 2-D) of processors, and exhibit high regularity of data flow. Video processing, for example, illustrates these characteristics. Although the overall data rate in and out of the overall processor is quite high, the actual computational rate of individual processors in an array with a processor for each image pixel is quite modest. For video at 30 frame/s, the computational rate associated with each image pixel is only 30 Hz. Overall high performance is therefore achieved by implementing a single processor for each pixel, each of those single processors implemented as a very low-complexity (i.e., high-yield), serial-data, arithmetic unit. Although the computing power of each processor is modest, the net processing power of a vast number of such processors is high. In such architectures, the penalty for dividing a full wafer (e.g., an array of 1000×1000 processors for a 1000×1000 pixel image) into smaller chips which are packaged and then reassembled to produce the full processor is in the large number of interconnections among the small ICs. By placing all processors on a single monolithic circuit, these "chip-to-chip" interconnections are eliminated and realized instead by the high-density wiring capabilities of integrated circuit technologies.

The reconfigurational techniques discussed in this section allow changes in the organization of processors by reconfiguring interconnections among the processors. The principles developed through research on WSI architectures in this area is considerably relevant to an emerging topic, namely reconfigurable computing using field-programmable gate arrays (FPGA). FPGAs share with several of the WSI architectures an organization based on an array of identical cells (in some cases the array contains a small number of different cell types) combined with a programmable interconnective fabric for programming the cells and for interconnecting cells. The user-programmable interconnections of FPGAs are designed to allow the designer to customize a general purpose FPGA component for a specific digital system function. Originally, such programming was viewed as a one-time action. However, an important capability is provided if the programming of the FPGA is completed quickly. In particular, if the system function involves executing a predefined sequence of computational functions (as often arises in image processing, for example), then the FPGA is programmed to optimally execute the first function, reprogrammed to optimally execute the second function, and so on. In this manner, the electronic circuit implemented by the FPGA is dynamically changed to meet the needs of the algorithms being executed. In many cases, the same capabilities that allow reconfiguring WSI arrays to bypass faults provides the reconfigurational ability to change the WSI component's architecture. One strategy (39,40) for

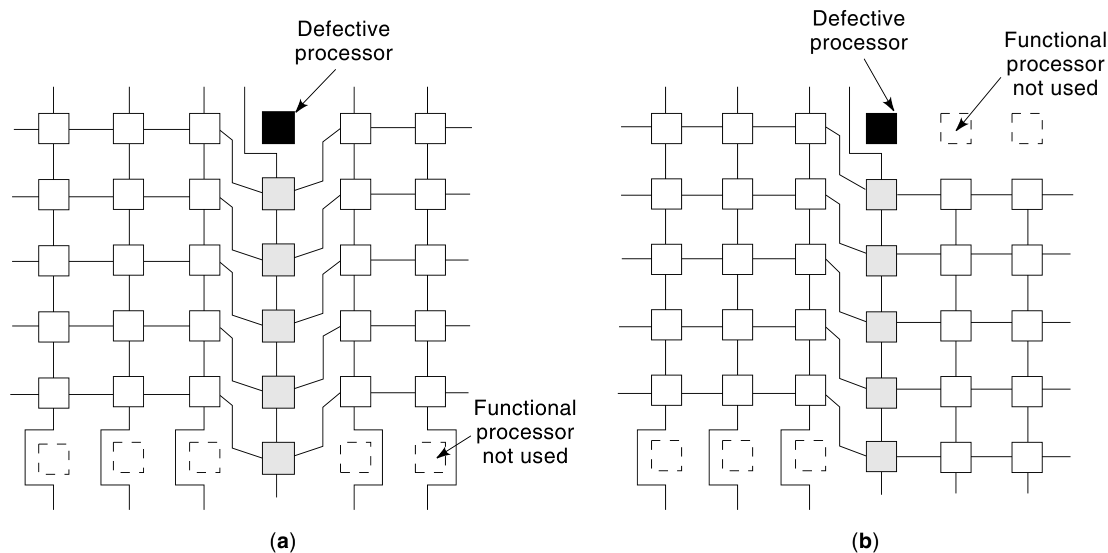

Figure 9. Illustration of global warping of a network in response to replacing a single defective processor. Cases (a) and (b) illustrate the multiplicity of reconfigurational solutions normally available.

fast switching of a WSI component's architecture involves storing a sequence of switch settings (representing the sequence of architectures) at each switch before starting the computations. Then the settings in this prestored sequence are locally "clocked" into the switches as the need for a different data flow organization occurs. This approach has reappeared in some of the discussions regarding fast reprogramming of FPGA's for versatile, reconfigurable computing.

The remainder of this section discusses several general approaches for global reconfiguration, in particular, the following:

- The CHiP computer architecture originally proposed by Snyder and his colleagues (39–41). This architecture is representative of the WSI cell and interconnective organizations used in several WSI studies.
- Reconfiguration using external testing to locate defective cells and external analysis to determine the settings for interconnective switches. The "divide and conquer" algorithm of Leighton and Leiserson (37) is used as an example.
- Reconfiguration using external testing to locate defective cells and using parallel data path wiring channels combined with autorouting techniques for configuring the array. The Diogenes approach of Rosenberg (42,43) is used to illustrate the approach.
- Reconfiguration using external testing to locate defective cells and embedding a search algorithm in the WSI array to evaluate the connectability to various parts of the array and to automatically determine a reconfiguration. The self-reconfiguration techniques developed at the Politecnico di Milano (44) are described.

Parallel Computer: Array of Processors and a Data Routing Network

A set of computers interconnected by a data network capable of routing data among processors has several qualitative features in common with WSI reconfigurable arrays. If a computer fails, data is routed past that computer and directed to another computer. High net data rates among the computers favor multiple, simultaneously active network paths, combined with switches capable of passing data from a source computer to a destination computer. The similarities to the problem of WSI reconfiguration become even more striking when the computers described are simplified to quite simple processors. This similarity is evident in the *configurable highly parallel* (CHiP) computer architecture (39–41) that continues to serve as a flexible model for combining an array of processing nodes and a fixed interconnection network over which routing channels for data routing are established.

The general CHiP architecture is illustrated in Fig. 10 for various versions of the architecture. Figure 10(a) illustrates the embedding of processors in a network consisting of single data paths running horizontally and vertically between processors (i.e., in "wiring channels") and degree-4 switches used to redirect the flow of data along those wiring channels. The shaded rectangles are processors of the array, and the open circles are the interconnective switches that allow data routing between the network of wiring channels and the processors. Figures 10(b) and (c) illustrate the same general approach but with different numbers of data paths per wiring channel and different degree switches. In general, a CHiP architecture is characterized by (1) the number of data paths per wiring channel and (2) the degree of the switches/processing nodes. This array/wiring channel model appears in several formal studies of reconfiguration. In addition, a WSI implementation of the CHiP architecture was pursued (41).

A wide variety of architectures based on switches completing data paths through a switched network have been studied, including investigations of efficient switching/routing circuits. Figure 11 illustrates representative switch designs.

- Figure 11(a) illustrates a two-state switch with "inputs" *A* and *B* and "outputs" *C* and *D*. Depending on the implementation of the switch, the ports are either bidirectional (e.g., implemented by pass transistors or fuses) or unidirectional (i.e., a specific direction is imposed by the circuitry implementing the switch). The two states shown are basically those of a cross-point switch.
- Figure 11(b) illustrates a switch intended to serve as a degree-4 switch (each port has a single, bidirectional data path). The three states shown arise assuming that no port is simultaneously connected to more than one other port. The bidirectional ports indicated are implemented by interconnect wires which are individually bidirectional.
- Figure 11(c) illustrates a four-port switch with separate input and output data interconnections at each port. This provides a substantially richer set of connective capabilities [not shown in Fig. 11(c)].

External Analysis and Programming of Reconfigured Mesh Array. The most common approach for determining the specific switch setting to perform a reconfiguration (1) first performs external testing of each cell to create a map of defective cells and then (2) uses a computer program to analyze the map and apply various algorithms to construct the map of

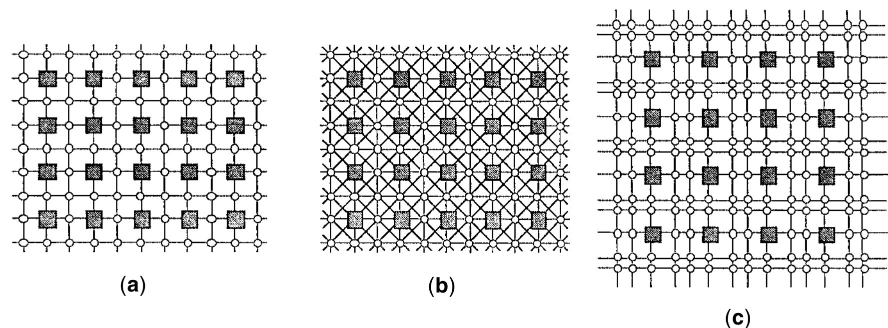

Figure 10. General architecture of the CHiP configurable computer. (a) Degree-4 switches and processors with single data path per interconnection channel. (b) Degree-8 switches and processors with single data path per interconnection channel. (c) Degree-4 switches and processors with two data paths per interconnection channel.

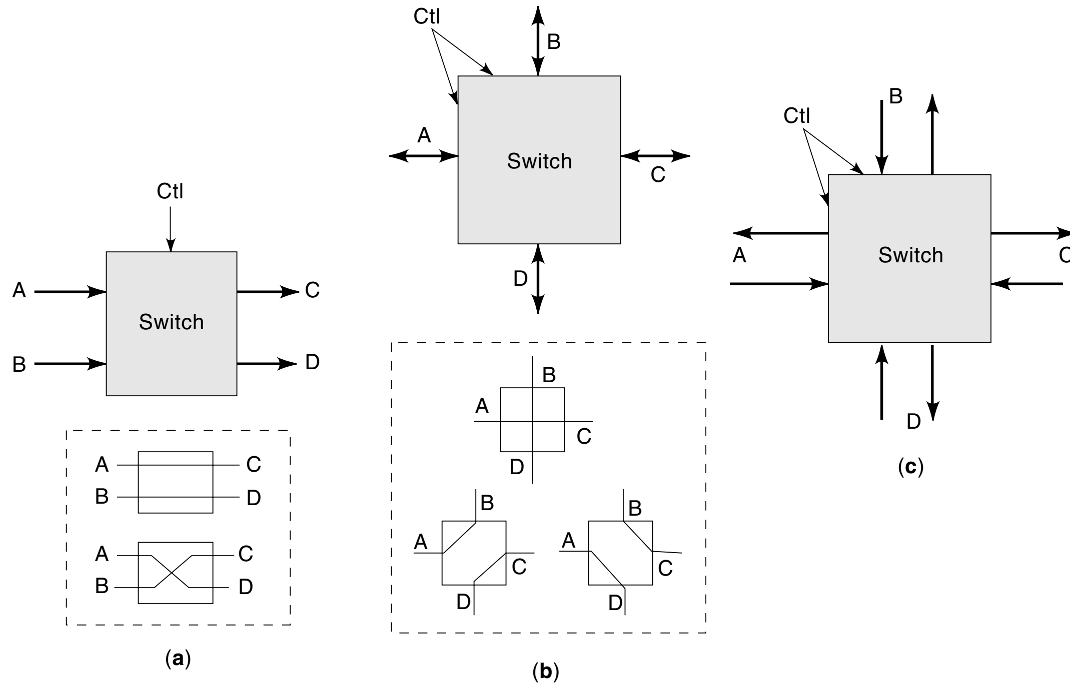

Figure 11. Examples of reconfigurational switches. (a) Two-state switch. (b) Three-state switch. (c) Bidirectional, four-port switch.

switch settings. The task is quite complex, particularly as the fraction of defective cells increases. However, there are some effective algorithms which are readily understood and are relatively straightforward to implement. The "divide and conquer" approach (37) (Fig. 12) illustrates a representative algorithm.

The target $K \times K$ array (with $k = 2^m$, m an integer) is defined by vertical and horizontal cuts performed hierarchically. The divide and conquer method in Fig. 12 bisects both the physical array and the ideal target array, distorting the cut in the target architectural array to reflect the available number of functional processors in each cut of the physical array. The algorithm proceeds in a top-down, hierarchical fashion.

Figures 12(a) and (b) illustrate the starting physical architecture (with identified defective processors) and starting tar-

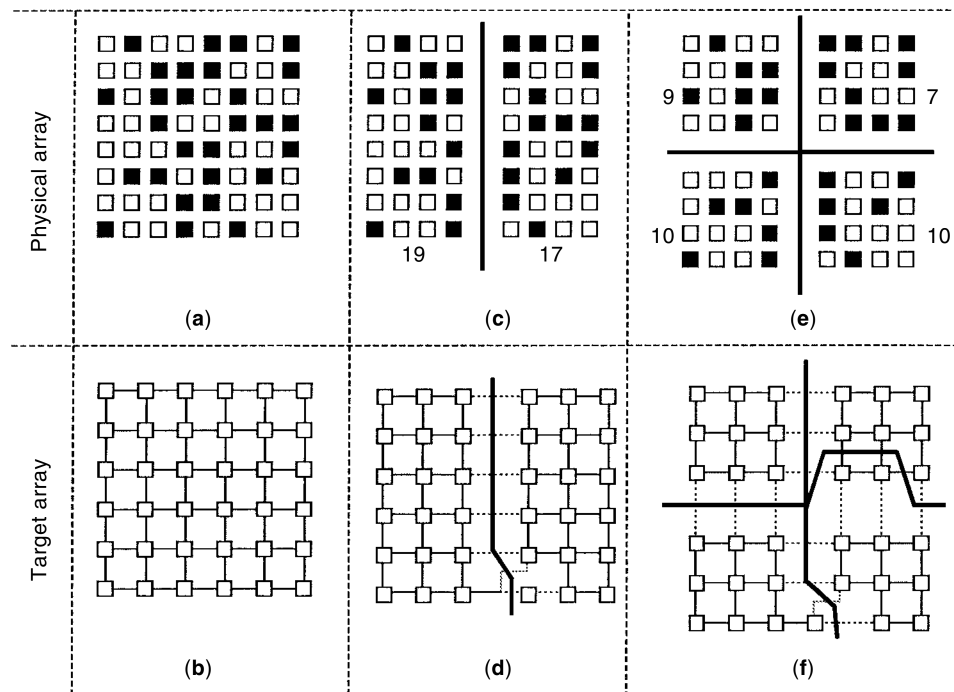

Figure 12. An example ("divide and conquer method") for reconfiguring a square mesh array on a physical array containing defective processors.

get array. In Fig. 12(c), the physical array is bisected vertically and the number of good processors on each side of the bisection is determined (in this case 19 to the left and 17 to the right. The 6×6 target architecture requires 18 processors on each side of the cut, requiring that the bisection of the target array be distorted [Fig. 12(d)] to reflect the 19 versus 17 processor counts in the physical array. Figure 12(e) shows the next bisection, in this case horizontal bisections of each half bisected in the previous step. The number of good processors in each quadrant of the bisected physical array is shown in the figure, ranging from seven in the top right quadrant to 10 in both of the bottom quadrants. Figure 12(f) illustrates the distortion of the horizontal cut of the target architecture to reflect the number of processors available in each quadrant of the physical array. This process continues to completion, leading to a fully defined specification of the mapping (and interconnection) of the good processors of the physical array to create the desired target array.

In addition to illustrating one of the important reconfiguration algorithms, this example also illustrates the importance of having a clear and provably correct reconfigurational algorithm. Convergence to a correct answer, if an answer exists, is of considerable importance. The literature on the formal methods of reconfiguration is rich in the creative application of techniques and principles from basic mathematics and computer science to embedding a desired regular array of processors into a physical array of good and bad processors.

Self-Reconfiguration Using Wiring Channels with Multiple Data Paths

Given an algorithm for mapping a target array architecture onto a physical array executed on a separate computer, it is interesting to consider the possibility of building that algorithm directly into the circuitry of a WSI component. If such self-reconfiguring components are possible, it is a modest step to adding built-in self-test (BIST) capabilities within the WSI component, leading to a complex component which tests itself and also reconfigures itself to avoid defective cells. The complexity of the additional circuitry to perform this self-reconfiguration is an obvious issue. However, if the additional circuitry is maintained at a sufficiently modest level, the combination of self-testing and self-reconfiguration would allow a WSI circuit to reconfigure itself to correct manufacturing defects and also to correct defects caused by failures in service (i.e., upon failure, to externally initiate self-test to identify the faulty cells and then initiate self-reconfiguration).

Here, only self-reconfiguration is considered, assuming that defective cells are located by externally applied tests and the *Diogenes approach* (42,43) is briefly described as an example of the general approach. The technique is based on using wiring channels containing several data paths and integrating a simple, local algorithm into the data path routing switches along the wiring channel.

Figure 13 illustrates the Diogenes approach. The need for multiple data paths in a wiring channel is indicated in Figs. 13(a) and (b). Figure 13(a) shows a binary tree for interconnecting eight processors. Figure 13(b) shows the embedding of that binary tree in a linear chain of tree switches. This simple example illustrates directly the appearance of multiple data paths at vertical cuts between the switches in Fig. 13(b). The physical layout of processors, switches, and parallel data paths used in the Diogenes approach is shown in Fig. 13(c). The basic principle here is that when an end-to-end connection is required to embed a network in an array, a physical data path is consumed between the origin of the data path and the destination of the data path, that is, a section of the data path in the wiring channel, of some length, depending on the positions of the source and destination, must be allocated for a specific connection. One approach is to dedicate a single path for such an interconnection, labeling the data path as "used" and requiring that a processor needing connection to a data path evaluate the 'used'/'unused' labels on the paths to locate an unused path. Alternatively, the path taken by data moving from a source to a destination could be moved from one path to another as processors along the way require new data path connections or terminate existing data path

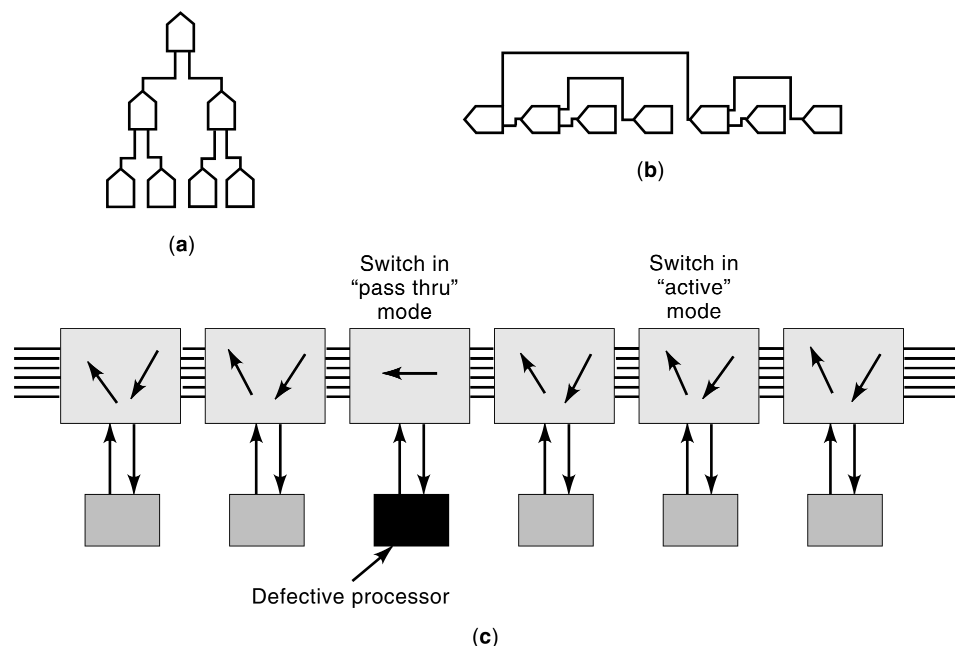

Figure 13. Illustration of the Diogenes approach. (a) Tree network for connection processors. (b) Mapping of network switches onto a serial array. (c) Use of switches which automatically change the data path on entering data in response to a line connected to the input of a processor or a line connected from the output of a processor.

connections. This latter approach motivates the Diogenes approach. As shown in Fig. 13(c), several data paths extend from the left to the right through routing switches. As data proceeds from switch to switch, it is moved to adjacent data paths to make the lowest data path available for the next processor requiring that path. If a processor makes a data connection into the wiring channel, all data presently propagating along the wiring channel move upward to the next higher data path. If a processor removes data from a data connection (that data does not propagate beyond that processor), then a path becomes free, and the data move through the switch shift downward to use that path now available. When a defective processor is encountered, the connections from the processor to the switch are disabled, and the data paths feed straight through the routing switch.

As described previously, it may appear that the movements of data between data paths, as the data propagates through routing switches, represents a very complex problem with no clear solution. However, quite simple algorithms are suitable for embedding a number of processor interconnective networks onto a WSI circuit. In particular, by drawing on an analogy to the use of "stacks" and "queues" when manipulating data in computer programs, rather simple algorithms depending only on the local action of processor (initiating a data path to another processor or terminating a data path from another processor) have been demonstrated. Stacks and queues are "last-in, first-out" and "first-in, first-out" data manipulation structures, respectively. Analogous to the operation of a data stack, the data inserted into the router by a processor might "push" each data flow through the router upward to the next higher data path, providing space on the lowest output line for the inserted data. Similarly, data extracted by a processor from the data paths allows each data path through the router to "fall" to the next lower data path.

Self-Reconfiguration with Routing to Nearby Functional Cells

The divide and conquer algorithm discussed earlier for embedding a 2-D processor array onto a physical array proceeds through a sequence of stages, starting at a high-level, global bisection and proceeding to finer levels of detail (i.e., smaller portions of the circuitry). The approach described here exploits a rather different direction. In particular, if one reconfigures a parallel array of processors used to perform parallel computing, is it possible to integrate a parallel processing algorithm directly into the WSI component to perform a global analysis of the overall WSI circuit, explore the range of possibilities for completing the interconnections needed to embed a target array in a physical array, and, on completion, establish the reconfiguration by setting the programming switches? This question was pursued by researchers at the Politecnico di Milano (44), and led to a rich literature regarding the general approach. The technical details of the approach are beyond the scope of this chapter. However, the general principles can be seen from the perspective of explorers.

Defective processor nodes are determined by testing before reconfiguration begins and are marked electronically. The case of embedding a 2-D mesh connected processor array in a physical array is considered here. Each processor site must establish four connections, the north, east, south and west connections. However, at the beginning, none of the processor sites know whether or how these connections can be made.

The reconfiguration process begins at one corner of the array. Electronic explorers (control signals) assigned to that corner port are sent out along each of the ports of the corner processor to locate a nearby processor which can be connected to that explorer's assigned port. On arriving at a nondefective processor site (the defective sites were identified and marked before the reconfiguration operation began), the arriving explorer spawns additional explorers at that site, and each of the spawned explorers starts its own search for a nearby processor to connect to its port. In this manner, the exploration initiated at the corner quickly expands into a vast number of explorers, each acquiring information regarding connectivity to a given processor's port. As the exploration reaches the far side of the physical array, that set of explorers has collectively acquired the global information needed to determine the reconfigurational structure. The wave to start the search begins with the explorers at one corner and flows to the opposite edges of the array. This directionality is preserved (i.e., no one can go home) until the search progresses to the far end. The explorers that reach the far end return to their processing sites and ports and launch a reconfigurational wave that moves toward the starting point at the beginning of the process. As explorers return to their home processing site, they set the switches that implement the connections to their ports of their processors, after which they allow the explorers who arrived during the search phase to return to their own processing sites and ports, where the returning explorers set their reconfigurational switches. There may be multiple options for connecting a given port to another processor. To resolve such cases, a priority decision scheme is built into the electronics implementing the search and configurational process.

An example of embedding a 5×7 array is shown in Fig. 14. Details of the algorithms are provided in the references cited above.

TECHNOLOGIES FOR ESTABLISHING INTERCONNECTIONS

The previous sections presented examples of reconfigurating a WSI or large area IC with possibly defective cells, to provide connections among the functional cells of the overall circuit. To a considerable extent, such reconfiguration involves efficiently implementing the switches required and programming the switches to the desired switch state. "Efficiency" relates to (1) minimizing the circuit area required for the switch and (2) minimizing the speed degradation due to the switch. Reconfigurational technologies have been explored extensively, and a rich and diverse set of techniques has been established. This section briefly summarizes the approaches with continued relevance for WSI and large area ICs and also for reconfigurable computing.

In general, the various technologies can be separated into generic categories:

- Physical alteration of an interconnection.
- Physical alteration of a fuse or antifuse.
- Electronic alteration of a fuse or antifuse.
- Electronic switch with
  physical alteration of control signal interconnections;

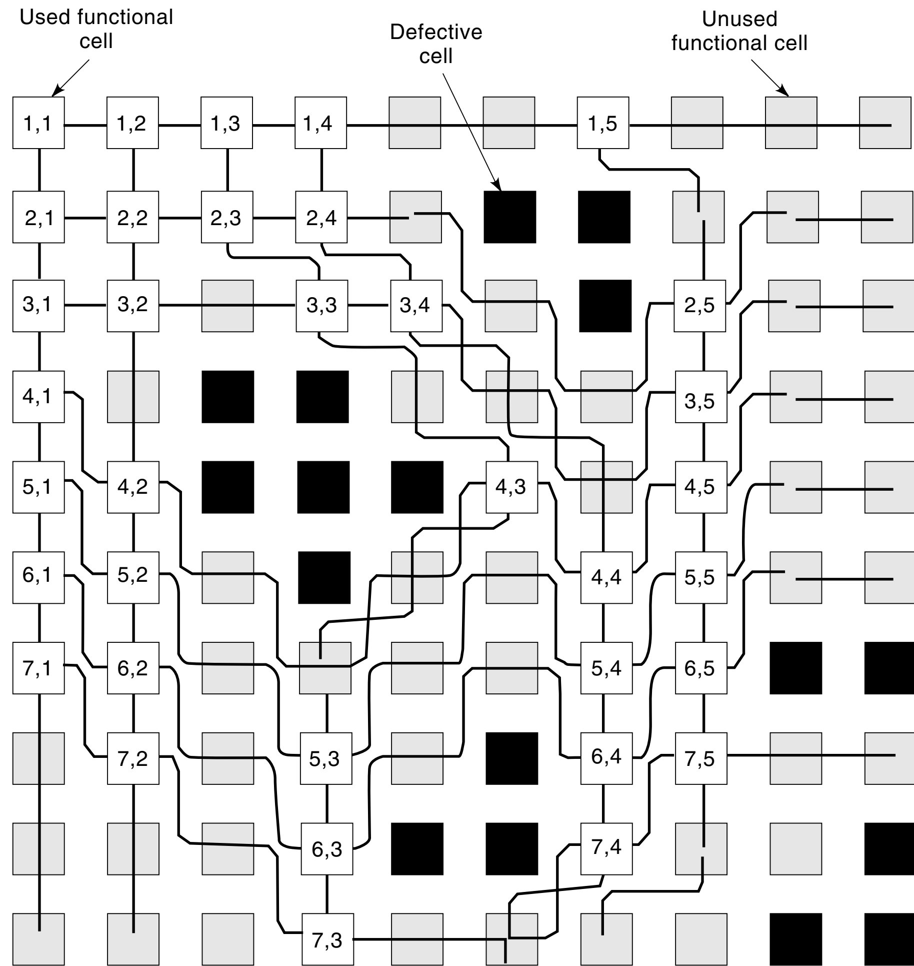

Figure 14. Example of reconfigured square array using the approach of (44).

setting of fuses/antifuses to establish control signals; and

electronic programming of control signals.

Representative examples of each of these approaches are briefly described below.

The most common example of physically altering an interconnection is provided by *laser ablation* with a sufficiently high-power laser beam, focused on a section of interconnection at the surface of the wafer, to "blast away" (ablate) the metal and create an open interconnection. Although seemingly primitive, high-speed laser reprogrammers have been commercially developed for reconfiguring DRAMs, and the approach is fast and low in cost. Other techniques involving selective deposition and etching of interconnective material have been reported, which relying on laser beams focused on the region where selective deposition of conducting material or selective etching of material is to be performed. The former case is important if selective connecting of interconnection segments in a plane is desired. However, fuse-based structures are probably more practical for this purpose.

A "fuse" ("antifuse") structure is a structure which normally has very low (high) resistance and is converted to a high (low)-resistance state. The "blowing" of a fuse is similar to laser ablation in the sense that an interconnection is broken. The "blowing" of an antifuse adds a connection where there was no previous connection. Specialized fuse (antifuse) structures have been developed and investigated, but compatibility with the underlying silicon CMOS process is an important capability. For this purpose, fuse and antifuse structures drawing on standard features of CMOS were investigated in a program at Lincoln Laboratories (45), demonstrating the practicality of such CMOS-compatible fuse structures. In the case of the CMOS-compatible antifuse, two heavily doped regions (diffusion regions) are separated by a narrow region of oppositely doped semiconductor, imposing a high resistance across the gap due to the back-biased diode. By heating the region with a pulsed laser, the dopants of the heavily doped regions diffuse across the separation, producing a continuous region of the same doping and a correspondingly low resistance. Lower resistance in the ON state is achieved by using wider diffused regions (requiring an increasing number of laser pulses to complete the connection as the width of the diffused region increases).

A different type of fuse is routinely used in EPROMs and EEPROMS, namely a MOSFET with a floating gate which can be electronically charged and discharged to cause a nonvolatile change in the threshold voltage of the MOSFET, establishing either an ON or OFF state. With a considerable amount of commercial development of such programmable gate transistors for conventional nonvolatile memory and

other electronically programmable circuits, these fuse/antifuse structures are attractive for fixed reconfiguration of a circuit. The technology adds a complication to the standard CMOS technology (providing the dual-gate MOS device). However, such transistors are already routinely used in mainstream CMOS processes (FPGAs, nonvolatile memory, etc.).

A given programmable switch architecture is usually implemented in a variety of ways. Figure 15(a) illustrates the states of an electronically controlled switch implemented with pass transistors shown in Fig. 15(b) and discussed in (46). The switch implements a four-port switch requiring only 1.5 transistors per port (and with a limited number of reconfiguration states). The role of the pass transistor is to implement an open or closed connection, depending on its control signal input. The control signal is externally entered into the WSI circuit and locally stored on flip-flops.

Figure 15(c) illustrates the same switch function shown in Fig. 15(b), but with either fuses or antifuses to set the connections of the switch. If fuses are used, fuses at locations corresponding to the OFF-state pass transistors of Fig. 15(b) are blown, creating an open in the connection. If antifuses are used, then the antifuses at locations corresponding to the ON-state pass transistors of Fig. 15(b) are blown. Figure 15(d) illustrates another possibility, namely placing segments (corresponding to the locations of the pass transistors in Fig. 15a) of the interconnections on the surface metal layer. Those segments corresponding to OFF-state pass transistors of Fig. 15(b) are opened by laser ablation.

The pass transistors used to implement the switch in Fig. 15(b) have one significant limitation, namely, the ON state resistance is quite high (significant because that resistance appears along an interconnection and would degrade the *RC* risetimes of signals propagating on that interconnection). The overall switch in Fig. 15(b) can be augmented with input receivers and output drivers to isolate the resistance of the pass transistor to the short connections within the switch. The input and output inverters regenerate the signal and also reestablish the polarity of the input signal at the output of the switch. Adding such receivers and drivers, however, converts potentially bidirectional paths through the switch into necessarily unidirectional paths.

CONCLUSION

This article has reviewed briefly the rich topic of wafer-scale integration. The term “wafer-scale integration” is of historical interest because use of a full wafer of contemporary deep submicron technology is less compelling than use of a full wafer circuitry using technologies of several years ago. However,

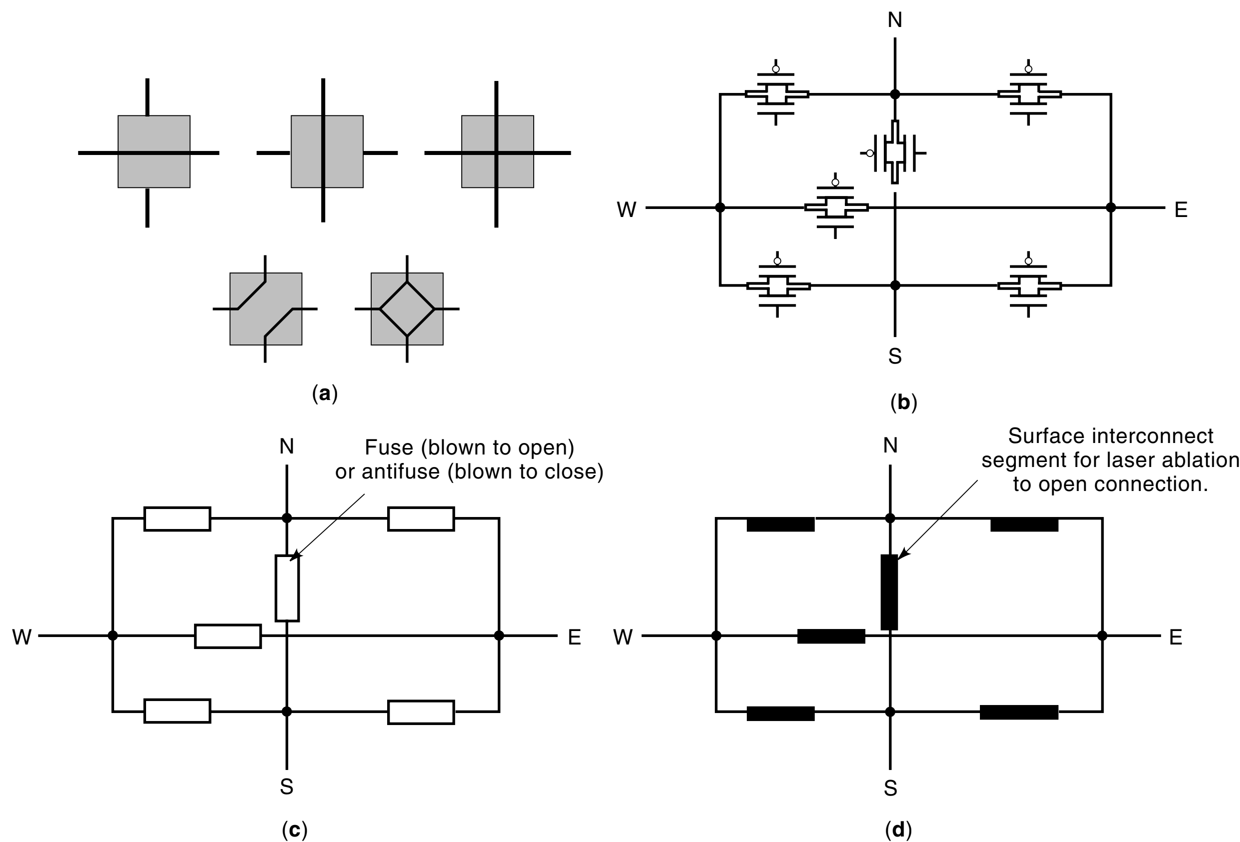

Figure 15. Example of a switch [adapted from (46)] illustrating implementation with various technologies. (a) Allowed switch states. (b) Pass transistor implementation allowing reprogramming [adapted from (46)]. (c) Fuse (state is established by opening connections) or antifuse (state is established by closing connections) for one-time or reprogrammable setting. (d) Placement of sections of the interconnections at the surface of the wafer, allowing laser ablation to break connections and establish the desired switch state.

the principles and methods of wafer-scale integration remain of considerable importance, not only for the migration of several techniques into mainstream IC design and manufacture but also for use in large-area ICs. With today's emphasis on "system-on-a-chip" capabilities, the maximum IC area imposed to achieve high yield in the presence of manufacturing defects is a serious limiting constraint. By applying appropriate techniques to correct a large-area IC with a microscopic defect, that area-bounding constraint is relaxed. In this sense, defect-tolerant, large-area ICs return to the objectives driving the early initiation of WSI projects, namely to eliminate the IC area boundary in cases where a higher performance and more compact system can be achieved by larger than normal IC area.

BIBLIOGRAPHY

1. E. A. Sack, R. C. Lyman, and G. Y. Chang, Evolution of the concept of the computer on a slice, *Proc. IEEE,* **52**: 1713–1720, 1964.
2. J. Lathrop et al., A discretionary wiring system as the interface between design automation and semiconductor manufacture, *Proc. IEEE,* **55**: 1988–1997, 1967.
3. D. F. Calhoun and L. D. McNamee, A means of reducing custom LSI interconnection requirements, *IEEE J. Solid-State Circuits,* **7**: 395–404, 1972.
4. R. C. Aubusson and I. Catt, Wafer-scale integration—a fault-tolerant procedure, *IEEE J. Solid-State Circuits,* **SC-13**: 339-344, 1978.
5. S. K. Tewksbury, *Wafer Level System Integration: Implementation Issues,* Norwell, MA: Kluwer, 1989.
6. E. E. Swartzlander, *Wafer Scale Integration,* Norwell, MA: Kluwer, 1989.
7. Saucier and Trihle (eds.), *Wafer-Scale Integration,* Amsterdam: Elsevier, 1986, pp. 89–97.
8. C. Jesshope and W. Moore (eds.), *Wafer-Scale Integration,* Bristol, UK: Adam Hilger, 1987, pp. 204–215.
9. R. M. Lea (ed.), *Wafer-Scale Integration II,* Amsterdam: North Holland, 1988.
10. *Proc. Int. Conf. on Wafer-Scale Integration,* Washington, DC: IEEE Computer Society Press, 1989.
11. M. Sami and F. Distante (eds.), *Wafer-Scale Integration III,* Amsterdam: North Holland, Elsevier, 1990.
12. J. Brewer and M. Little (eds.), *Int. Conf. on Wafer-Scale Integration,* Washington, DC: IEEE Computer Society Press, 1990.
13. M. J. Little and V. K. Jain (eds.), *Wafer-Scale Integration,* Washington, DC: IEEE Computer Society Press, 1991, pp. 83–89.
14. S. K. Tewksbury and R. M. Lea (eds.), *Wafer-Scale Integration,* Washington, DC: IEEE Computer Society Press, 1994.
15. G. C. Chapman and S. K. Tewksbury (eds.), *Wafer-Scale Integration,* Washington, DC: IEEE Computer Society Press, 1995.
16. S. K. Tewksbury, D. Sciuto, and G. Chapman, *Proc. IEEE 1996 Int'l Conference on Innovative Systems in Silicon,* Piscataway, NJ: IEEE Press, 1996.
17. S. K. Tewksbury, Architectural Fault Tolerance, in P. Gyvez and D. Pradhan (eds.), *IC Manufacturability: The Art of Process and Design Integration,* Piscataway, NJ: IEEE Press, in press.
18. C. H. Stapper, Modeling of integrated circuit defect sensitivities, *IBM J. Res. Dev.,* **27**: 549–557, 1983.
19. A. V. Ferris-Prabhu, Defects, faults and semiconductor device yield, in I. Koren (ed.), *Defect and Fault Tolerance in VLSI Systems,* New York: Plenum, 1989, pp. 33–46.
20. R. P. Cenker et al., A fault-tolerant 64K dynamic random access memory, *IEEE Trans. Electron Devices,* **ED-26**: 853–860, 1979.
21. B. F. Fitzgerald and E. P. Thoma, Circuit implementation of fusible redundant addresses of RAMs for productivity enhancement, *IBM J. Res. Develop.,* **24**: 291–298, 1980.
22. R. T. Smith et al., Laser programmable redundancy and yield improvement in a 64K DRAM, *IEEE J. Solid-State Circuits,* **SC-16**: 506–514, 1981.
23. Y. Kitano et al., A 4-Mbit full wafer ROM, *IEEE J. Solid-State Circuits,* **SC-15**: 686–693, 1980.
24. Y. Egawa et al., A 1-Mbit full wafer MOSRAM, *IEEE J. Solid-State Circuits,* **SC-15**: 677–686, 1980.
25. N. MacDonald et al., 200Mb Wafer Memory, *Digest: 1989 IEEE Int. Solid-State Circuits Conf.,* San Francisco, 1989, pp. 240–241.
26. E. J. McCluskey, *Logic Design Principles with Emphasis on Testable Semicustom Circuits,* Bedford, MA: Digital Press, 1982.
27. J. A. Abraham and W. K. Fuchs, Fault and error models for VLSI, *Proc. IEEE,* **74**: 639–654, 1986.
28. B. T. Murphy, Cost-size optima of monolithic integrated circuits, *Proc. IEEE,* **52**: 1537–1545, 1964.
29. C. H. Stapper, On yield, fault distributions and clustering of particles, *IBM J. Res. Dev.,* **30**: 326–338, 1986.
30. S. Lin and D. J. Costello, *Error Control Coding: Fundamentals and Applications,* Englewood Cliffs, NJ: Prentice-Hall, 1982.
31. J. F. MacWilliams and N. J. A. Sloane, *The Theory of Error Correcting Codes,* Amsterdam: North-Holland, 1977.
32. K.-H. Huang and J. A. Abraham, Algorithm-based fault tolerance for matrix operations, *IEEE Trans. Comput.,* **C-33**: 518–528, 1984.
33. J.–Y. Jou and J. A. Abraham, Fault-tolerant matrix arithmetic and signal processing on highly concurrent computing structures, *Proc. IEEE,* **74**: 732–741, 1986.
34. D. P. Siewiorek and R. S. Swarz, *The Theory and Practice of Reliable System Design,* Bedford, MA: Digital Press, 1982.
35. N. Tsuda, Rotary spare replacement redundancy for tree architecture WSIs, in M. J. Little and V. K. Jain (eds.), *Wafer-Scale Integration,* pp. 83–89, Washington, DC: IEEE Computer Society Press, 1991.
36. J. W. Greene and A. El Gamal, Configuration of VLSI arrays in the presence of defects. *J. ACM,* **31**: 694–717, 1984.
37. T. Leighton and C. E. Leiserson, Wafer-scale integration of systolic arrays, *IEEE Trans. Comput.,* **C-34**: 448–461, 1985.
38. L. Snyder, Overview of the CHiP computer, in John P. Gray (ed.), *VLSI 81.* Boston: Academic Press, 1981, pp. 237–246.
39. L. Snyder, Introduction to the configurable highly parallel computer, *IEEE Computer,* **15**: 47–56, 1982.
40. L. Snyder, Parallel programming and the poker programming environment, *IEEE Computer,* **17**: 27–36, 1984.
41. K. S. Hedlund, WASP—a Wafer-scale Systolic Processor, *Proc. IEEE Int. Conf. Comp. Design,* 1985, pp. 665–671.
42. A. L. Rosenberg, The Diogenes approach to testable fault-tolerant arrays of processors, *IEEE Trans. Comput.,* **C-32**: 902–910, 1983.
43. A. L. Rosenberg, Graph-theoretic approaches to fault-tolerant WSI, in C. Jesshope and W. Moore (eds.), *Wafer-Scale Integration,* London: Adam Hilger, 1986, pp. 10–23.
44. R. Negrini and R. Stefanelli, Comparative evaluation of space- and time-redundancy approaches for WSI processing arrays, in G. Saucier and J. Trihle (eds.), *Wafer-Scale Integration,* Amsterdam: Elsevier, 1986, pp. 207–222.
45. J. M. Canter, G. H. Chapman, B. Mathur, M. L. Naiman, and J. I. Raffel, A laser-induced ohmic link for wafer-scale integration, *IEEE Trans. Electron Devices,* **ED-33**: 1861, 1986.

46. V. N. Donaints, V. G. Lazarev, M. G. Sami, and R. Stefanelli, Reconfiguration of VLSI arrays: A technique for increased flexibility and reliability, *Microprocessing and Microprogramming,* **16**: 101–106, 1985.

S. K. TEWKSBURY
West Virginia University