# EVOLVABLE HARDWARE

# EVOLVABLE HARDWARE

- **Evolvable hardware for electrical circuits**
  - **Digital**
  - **Analog**

- **Reconfigurable antenna**

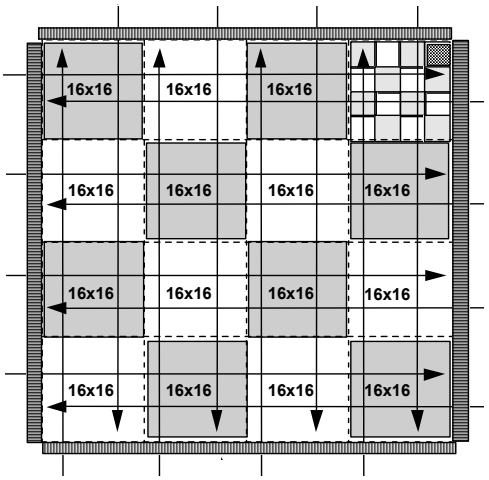
# DIGITAL FIELD-PROGRAMMABLE GATE ARRAYS (FPGAs)

- **Massively parallel computational devices**
- **Originally seen as commercial alternative to ASICs — especially for new designs (that may have bugs)**
- **Commercially introduced by Xilinix in early 1980s**
- **Some FPGAs are the *anti-fuse* type (i.e., are irreversibly programmed by the user in the field by the one-time application of a high voltage)**
- **Other types can be reprogrammed, but only a few hundred times. For example, some FPGAs must be physically removed from their operating environment and erased with ultraviolet light before they can be electrically reprogrammed.**

# RAPIDLY RECONFIGURABLE FIELD-PROGRAMMABLE GATE ARRAYS (FPGAs)

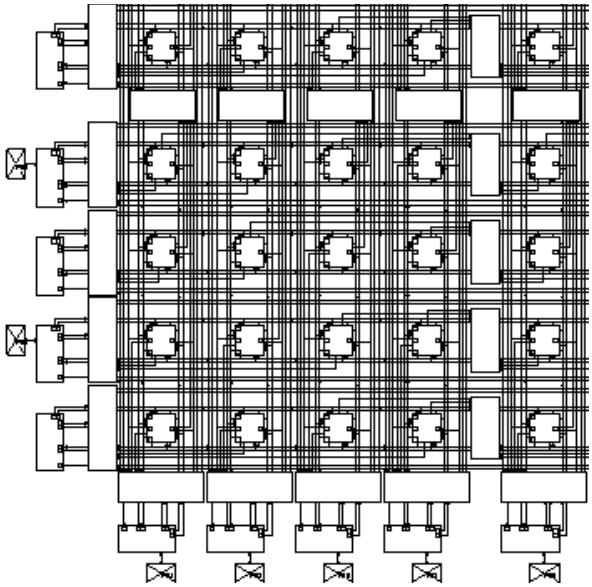
- **Typically there is a regular "sea" of some basic function unit**
- **There are input and output units along the edge of the chip**
- **There are interconnection resources — often a mixture of short, medium, and long lines connecting the basic function units to one another and to the input and output units**
- **The basic function units are sometimes look-up tables (LUTs) and sometimes multiplexers (MUXs).**
- **The basic function units often contain flip-flop(s) and other devices**

# HIERARCHICAL VIEW OF XILINX XC6216 FPGA




- **The Xilinix XC6216 has 64×64 cells**
- **4×4 arrangement of regions, with each region containing a total of 16×16 = 256 individual cells.**
- **At the next lower level of the hierarchy, there is a 4×4 arrangement of subregions, with each subregion containing 4×4 = 16 individual cells.**

# 5×5 CORNER OF XILINX XC6216 FPGA

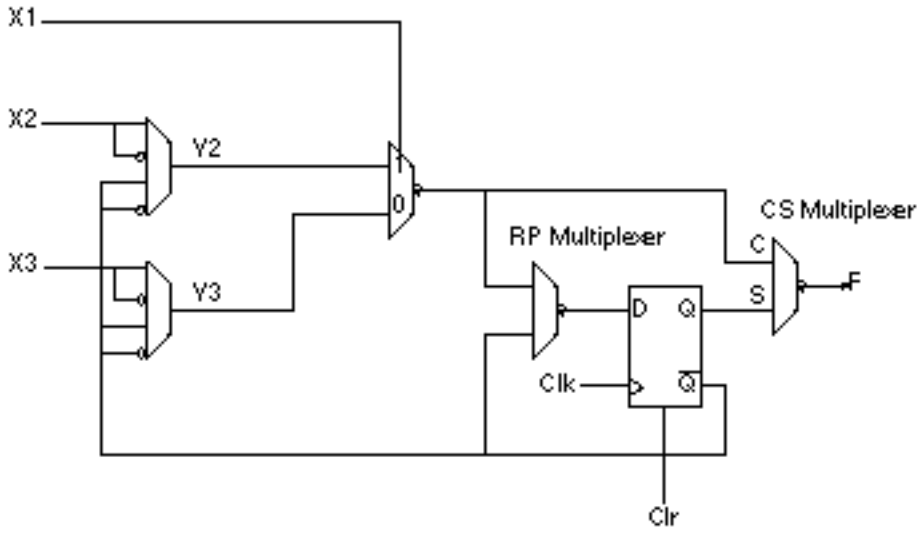


- **Shows**
  - **One 4×4 group of function units**
  - **10 of 256 input-output blocks (IOBs) along the periphery of the chip**
  - **Short line interconnections within the 4×4 group of function units**
  - **Several intermediate and long lines for interconnections**

# EVOLVABLE HARDWARE

# RAPIDLY RECONFIGURABLE FIELD-PROGRAMMABLE GATE ARRAYS (FPGAs)

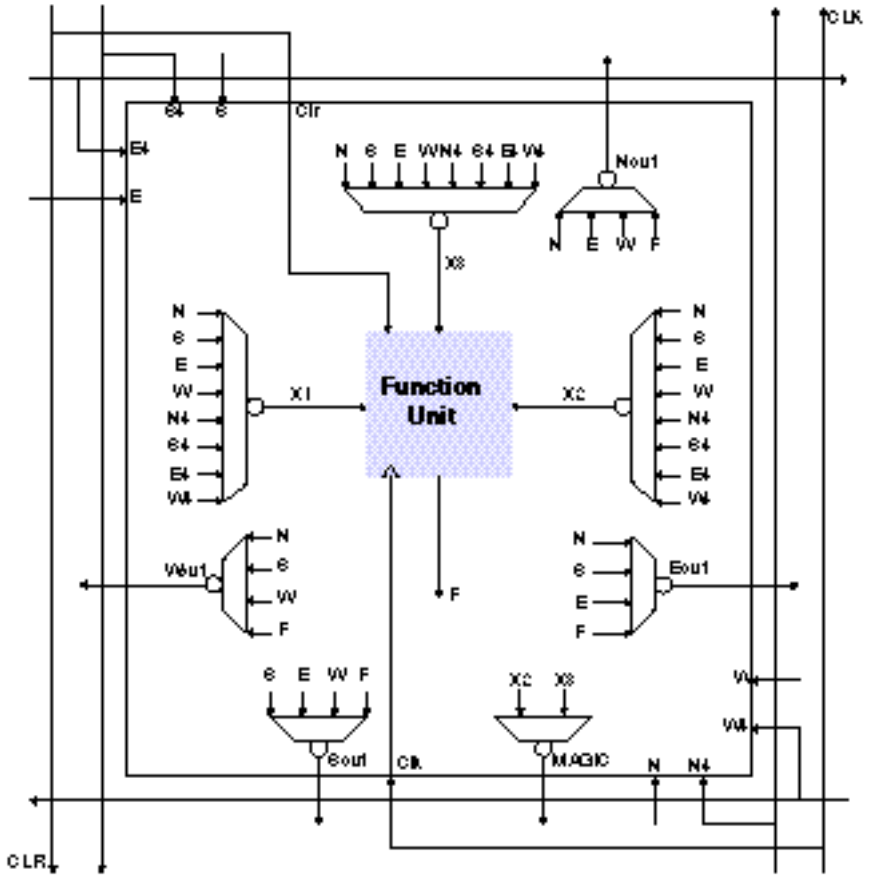
# FUNCTION UNIT FOR ONE CELL OF THE XILINX XC6216 CHIP



# EVOLVABLE HARDWARE

# RAPIDLY RECONFIGURABLE FIELD-PROGRAMMABLE GATE ARRAYS (FPGAs)

# ONE CELL OF THE XILINX XC6216 CHIP



# DESIGN PROCESS FOR FPGAs

**• First, the engineer conceives the design (which often incorporates subcircuits from a library).**

**• Second, the engineer's design of the desired circuit is *captured* by the CAD tool in the form of a schematic diagram, a set of Boolean expressions, or a general-purpose high-level description language such as VHDL (an acronym for "VHSIC Hardware Description Language" in which VHSIC is, in turn, an acronym for "Very High Speed Integrated Circuits").**

**• Third, a *technology mapping* converts the description of the circuit into logical function units of the particular type that are present on the particular FPGA chip that is to be used.**

# DESIGN PROCESS FOR FPGAs — CONTINUED

- **Fourth, a (usually) time-consuming *placement* step places the logical function units into particular locations on the FPGA.**

- **Fifth, a *routing* between the logical function units on the chip is created using the FPGA's interconnection resources. This routing process may be difficult because interconnection resources are extremely scarce for almost all commercially available FPGAs.**
  - **The placement and routing ends up determining the speed to which the chip's programmable clock is set — typically about 1/l0 of the maximum).**

# DESIGN PROCESS FOR FPGAs — CONTINUED

- **Sixth, the hundreds of thousands of *configuration bits* are created. The encoding scheme for the configuration bits of almost all commercially available FPGAs are kept confidential by the FPGA manufacturers for a variety of reasons (including deterrence of reverse engineering of the prototype products that account for much of the FPGA market).**

- **Seventh, the configuration bits are downloaded into the FPGA's memory.**

# USING FPGAs FOR GENETIC ALGORITHM WORK

**Since the fitness measurement task residing in the inner loop of a run of the genetic algorithm or genetic programming constitutes the main component of the computational burden of a run, the question arises as to whether the massive parallellism of FPGAs can be exploited to accelerate this time-consuming task.**

# PRACTICAL OBSTACLES TO USING FPGAs FOR GENETIC ALGORITHM WORK

- **First, the encoding schemes for the configuration bits of almost all commercially available FPGAs are complex and kept confidential by the FPGA manufacturers.**

# PRACTICAL OBSTACLES — CONTINUED

- **Second, the tasks of technology mapping, placement, routing, and creation of the configuration bits consume so much time as to preclude practical use of a FPGA in the inner loop of the genetic algorithm or genetic programming.**
- **Even if these four tasks could be reduced from the usual hours/minutes to as little as 10 seconds for each individual in the population, these four tasks would consume $10^6$ seconds (278 hours) in a run involving a population as minuscule as 1,000 for as few as 100 generations.**
- **A run involving a population of 1,000,000 individuals would multiply the above unacceptably long time (278 hours) by 1,000 (to 278,000 hours).**

# PRACTICAL OBSTACLES — CONTINUED

- **Third, the 500 milliseconds typically required for the task of downloading the configuration bits to a FPGA is insignificantly small for an engineer who has spent hours, days, or months on a single prototype design.**
  - **However, this downloading time would consume 14 hours for even a minuscule population of 1,000 that was run for as few as 100 generations.**
  - **A run involving a population of 1,000,000 individuals would multiply this already unacceptably long time by 1,000.**
  - **What's worse – both of these unacceptably long times (278 hours and 14 hours) are merely *preliminary* to the time required by the FPGA for the actual problem-specific fitness measurement.**

# PRACTICAL OBSTACLES — CONTINUED

- **Thus, there is a discrepancy of *numerous orders of magnitude* between the time required for the technology mapping, placement, routing, bit creation, and downloading tasks and the time available for these preliminaries in the inner loop of a practical run of the genetic algorithm or genetic programming.**
- **Thus, reconfigurability is not enough for practical work on FPGAs with genetic algorithms and genetic programming. *Rapid reconfigurability* is what is needed – where "rapid" means times ranging between microseconds to milliseconds for *all five* preliminary tasks (technology mapping, placement, routing, bit creation, and downloading).**

# PRACTICAL OBSTACLES — CONTINUED

- **Fourth, the genetic algorithm starts with an initial population of randomly created individuals and uses probabilistic operations to breed new candidate individuals. These randomly created individuals do not conform to the design principles that are employed by humans.**
  - **Most commercially available FPGAs are vulnerable to damage caused by combinations of configuration bits that connect contending digital signals to the same line. The process of verifying the acceptability of genetically created combinations of configuration bits is complex and would be prohibitively slow in the inner loop of GA/GP. Invulnerability to damage is needed in order to make FPGAs practical for the inner loop of the genetic algorithm or genetic programming.**

# XILINX XC6216 FEATURES THAT ADDRESS THESE PROBLEMS

- **Invulnerability to damage because of multiplexer (MUX) architecture**
- **The encoding scheme for the configuration bits is public. This openness enables us to write software for configuring the FPGA.**
- **The encoding scheme for the configuration bits is simple in comparison to most other FPGAs — thereby potentially significantly accelerating the technology mapping, placement, routing, and bit creation tasks. This simplicity is critical because these tasks are so time-consuming as to preclude practical use of conventional FPGAs in the inner loop of a genetic algorithm.**

# XILINX XC6216 FEATURES THAT ADDRESS THESE PROBLEMS — CONTINUED

- **The XC6216 streamlines the task of downloading the configuration bits.**
  - **The fact that the configuration bits reside in the address space of the host processor accelerates the downloading.**
  - **This fact also makes it possible to change single configuration bits without downloading any others.**
  - **Moreover, the downloading of the configuration bits is faster for the Xilinx XC6000 series than almost all other FPGAs because the downloading is done via a memory bus that handles 32 bits simultaneously and that operates at speeds far in excess of those of current serial interfaces.**
  - **The Xilinx XC6000 series has a wild card feature that facilitates downloading of regular bit patterns.**

# NEGATIVE FEATURES OF FPGAs

- **First, the clock rate (established by a programmable oscillator) at which a FPGA actually operates is often much slower (typically around ten-fold) than that of contemporary microprocessor chips.**

- **Second, the operations that can be performed by the logical function units of a FPGA are extremely primitive in comparison to the 32-bit arithmetic operations that can be performed by contemporary microprocessor chips.**

# TYPES OF PROBLEMS WHERE FPGAs MAY BE USEFUL FOR GA/GP WORK

- **One indicator of the possible suitability of a problem for FPGAs is the prominence of bit-level operations (or operations that can be conveniently and efficiently recast as bit-level operations).**
  - **For example, for problems of image processing, pattern recognition, and manipulation of sequential data, a single multiplexer or flip-flop can often perform the same computation that a conventional microprocessor will perform with a 32-bit operation.**

# TYPES OF PROBLEMS WHERE FPGAs MAY BE USEFUL FOR GA/GP WORK — CONTINUED

- **A second indicator is the prominence of parallelizable computations. Problems containing a large number of identical parallelizable computations (e.g., cellular automata problems) are especially suitable for FPGAs because they can potentially maximize the benefits of the device's fine-grained parallelism.**
  - **Problems containing numerous disparate parallelizable computations that must be performed serially in a conventional microprocessor are also suitable for FPGAs. For these problems, the disparate parallelizable computations are housed in different areas of the FPGA.**

# TYPES OF PROBLEMS WHERE FPGAs MAY BE USEFUL FOR GA/GP WORK — CONTINUED

- **A third indicator of the possible suitability of a problem for FPGAs is the prominence of computations that can be pipelined.**
  - **The stages of the pipeline many correspond to different fitness cases or different steps of a simulation or computation.**
  - **The benefit of pipelining rises proportionately with the number of stages in the pipeline that the FPGA can process simultaneously.**

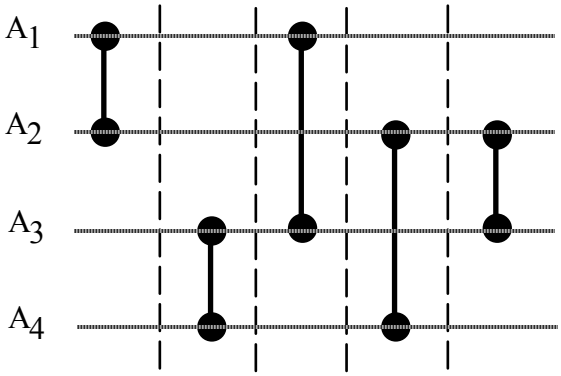
# EXAMPLE PROBLEM

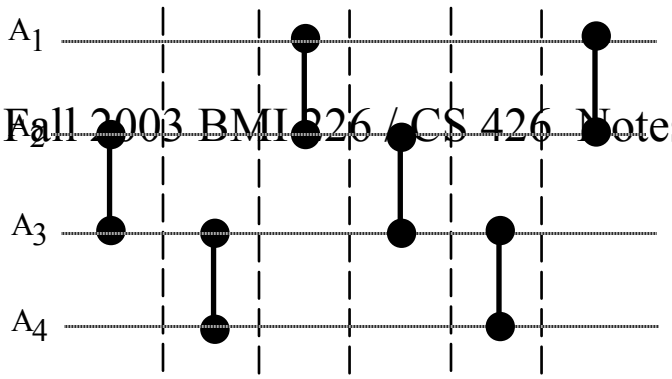
# SORTING NETWORKS

- **They sort *N* items**
- **They are composed of COMPARE-SWAP operations**
- **They consume fixed-time**
- **They operate in a way that is *oblivious* to the *N* items**

# SORTING NETWORK – 4 ITEMS

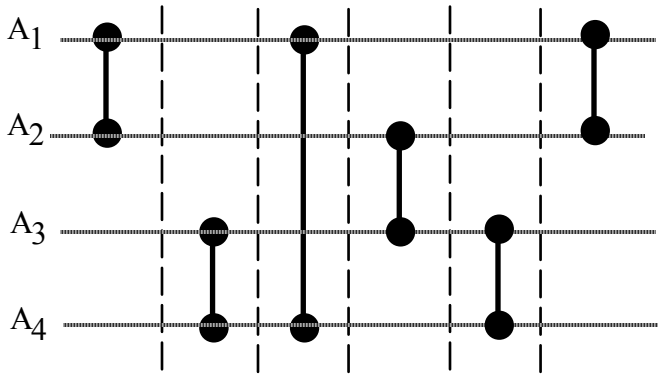
# EXAMPLE OF OPTIMAL NETWORK



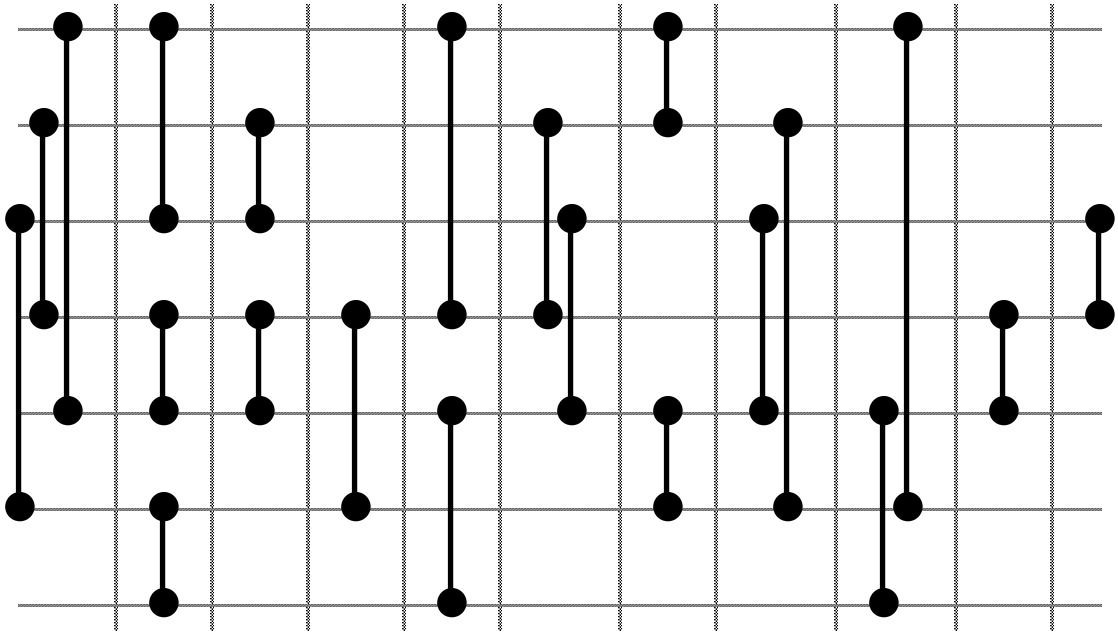




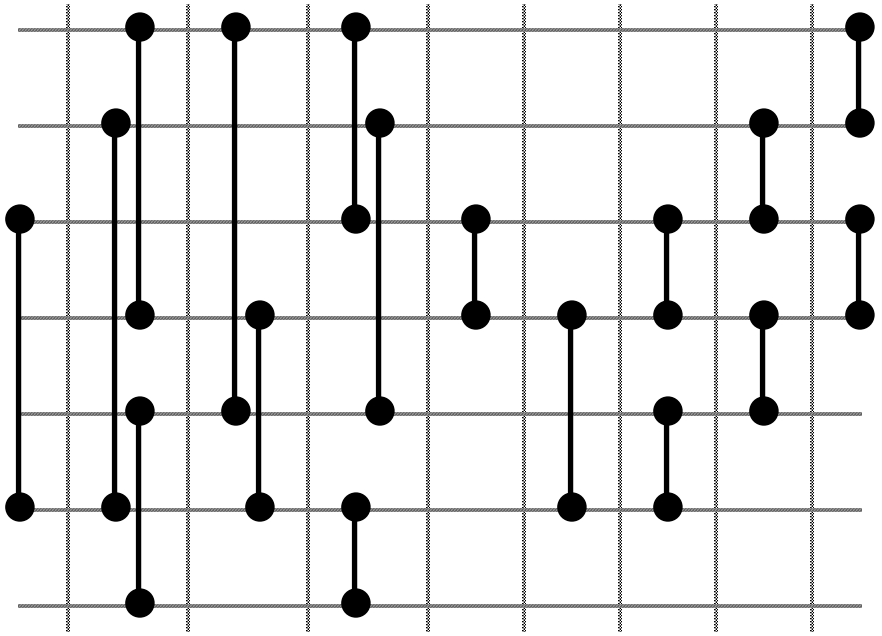

# 2 EXAMPLES OF NON–OPTIMAL NETWORKS



# A NON-OPTIMAL SORTING NETWORK FOR 7 ITEMS



# SORTING NETWORK – 7 ITEMS



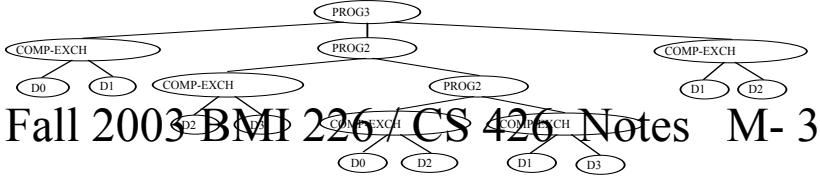
# CO-EVOLUTION OF FITNESS CASES (HILLIS)

# NUMBER OF COMPARISON-SWAPS IN VARIOUS SORTING NETWORKS FOR 16 ITEMS

| | |
|---|---|
| **Bose and Nelson (1962)** | **65** |
| **Hillis Simulated Evolution - WITHOUT parsites (1990, 1992)** | **65** |
| **Batcher and Knuth (1964)** | **63** |
| **Shapiro (1969)** | **62** |
| **Hillis Simulated Evolution - WITH parsites (1990, 1992)** | **61** |
| **Green** | **60** |

# THE COMPARE-EXCHANGE FUNCTION FOR SORTING NETWORK PROBLEM

| **Two Arguments** | | **Two Results** | |
|---|---|---|---|
| $A_i$ | $A_j$ | $R_i$ | $R_j$ |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 |




# PROGRAM TREE FOR MINIMAL 4-SORTER

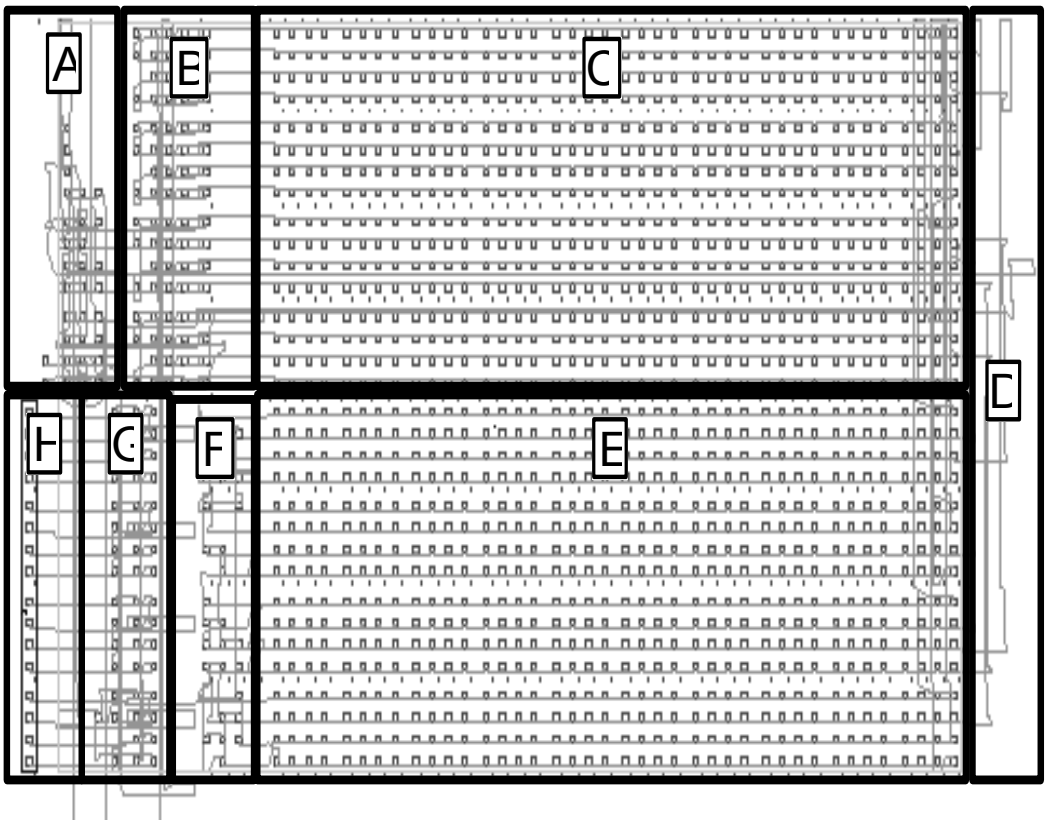
# TABLEAU FOR MINIMAL SORTING NETWORK PROBLEM FOR 7 ITEMS USING FPGA

| | |
|---|---|
| **Objective:** | **Synthesize the design of a minimal sorting network for sorting seven items using a field-programmable gate array for the fitness evaluation task.** |
| **Program architecture:** | **One result-producing branch.** |
| **Initial function set for the result-producing branches:** | **$F_{rpb\text{-}initial}$ = {`COMPARE-EXCHANGE`, `NOP`, `PROG2`, `PROG3`, `PROG4`}.** |
| **Initial terminal set for the result-producing branches:** | **$T_{rpb\text{-}initial}$ = {`D1`, ..., `D7`} for sorting seven items.** |
| **Fitness cases:** | **The 7 ∞ 128 = 896 bits appearing in all possible vectors of seven bits.** |
| **Raw fitness:** | **Raw fitness is 1 plus the number of bits (0 to $7 \times 2^7 = 128$) for which the sorted bits are incorrectly plus 0.01 times the number of `COMPARE-EXCHANGE` functions that are actually executed. that are actually executed.** |
| **Standardized fitness:** | **Same as raw fitness.** |

| | |
|---|---|
| **Hits:** | **The number of `COMPARE-EXCHANGE` functions that are actually executed.** |
| **Wrapper:** | **Exchanges where $i = j$ and exchanges that are identical to the previous exchange are ignored. Only the first $Cmax$ `COMPARE-EXCHANGE` functions in a program are executed.** |
| **Parameters:** | **$M$ = 1,000 (for sorting seven items). $G$ = 501. $S_{\mathrm{rpb}}$ = 300. $N_{\mathrm{rpb}}$ = 1.** |
| **Result designation:** | **Best-so-far pace-setting individual.** |
| **Success predicate:** | **Fitness appears to have reaches 1.16.** |

# MINIMAL SORTING NETWORK PROBLEM USING FPGA

## 32×64 PORTION OF THE XILINIX XC6216 CHIP FOR SORTING NETWORKS




- **There are 2 such portions in the full 64×64**
- **This figure does not show the ring of input-output blocks on the periphery of the chip that surround the 64×64 area of cells**

# MINIMAL SORTING NETWORK PROBLEM — CONTINUED

**•The problem of synthesizing the design of sorting networks was run on a host PC Pentium type computer with a Virtual Computer Corporation "HOT Works" PCI board containing a Xilinx XC6216 field-programmable gate array.**

- **This combination permits the field-programmable gate array to be advantageously used for the computationally burdensome fitness evaluation task while permitting the general-purpose host computer to perform all the other tasks.**

# MINIMAL SORTING NETWORK PROBLEM — CONTINUED

- **The host PC begins the run by creating the initial random population (with the XC6216 waiting).**
- **Then, for generation 0 (and each succeeding generation), the PC creates the necessary configuration bits to enable the XC6216 to measure the fitness of the first individual program in the population (with the XC6216 waiting).**
- **Thereafter, the XC6216 measures the fitness of one individual.**
  - **Note that the PC can simultaneously prepare the configuration bits for the next individual in the population and poll to see if the XC6216 is finished.**

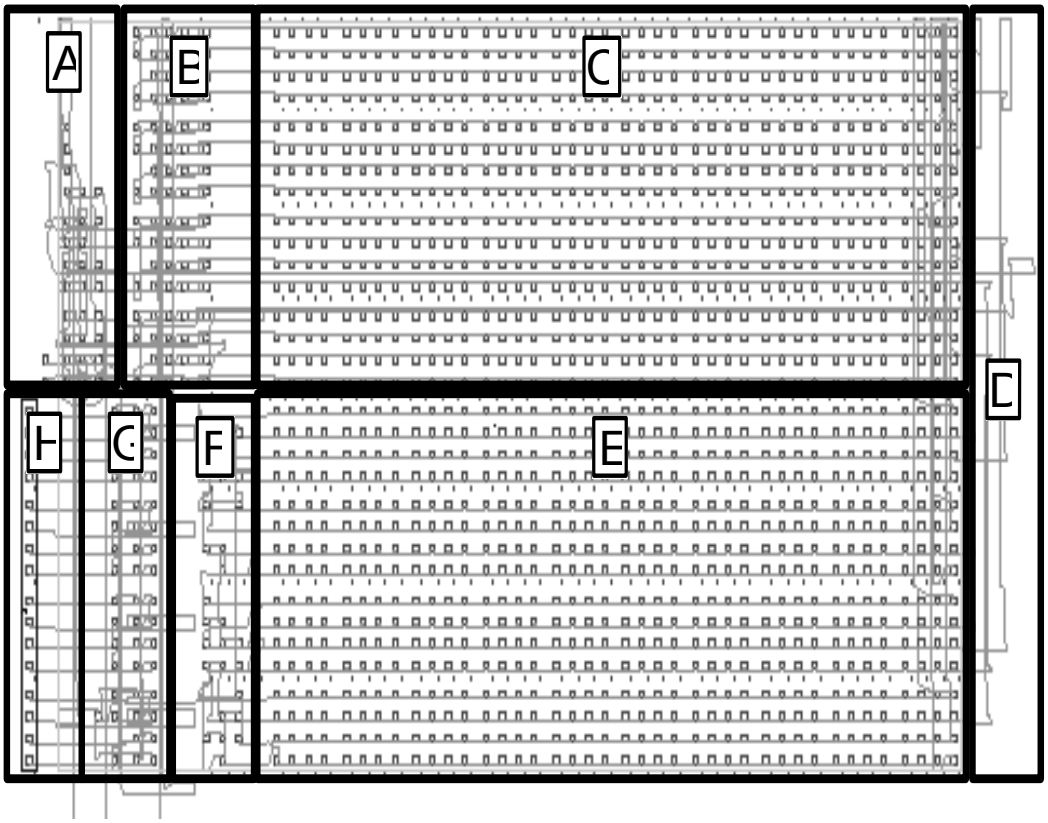
# MINIMAL SORTING NETWORK PROBLEM — CONTINUED

- **After the fitness of each individual in the current generation of the population is measured, the genetic operations (reproduction, crossover, and mutation) are performed (with the XC6216 waiting).**
  - **Since the tasks of creating the initial random population and of executing the genetic operations can be performed very rapidly in comparison with the fitness evaluation task, it is acceptable to leave the XC6216 waiting while these tasks are being performed.**

# MINIMAL SORTING NETWORK PROBLEM — CONTINUED

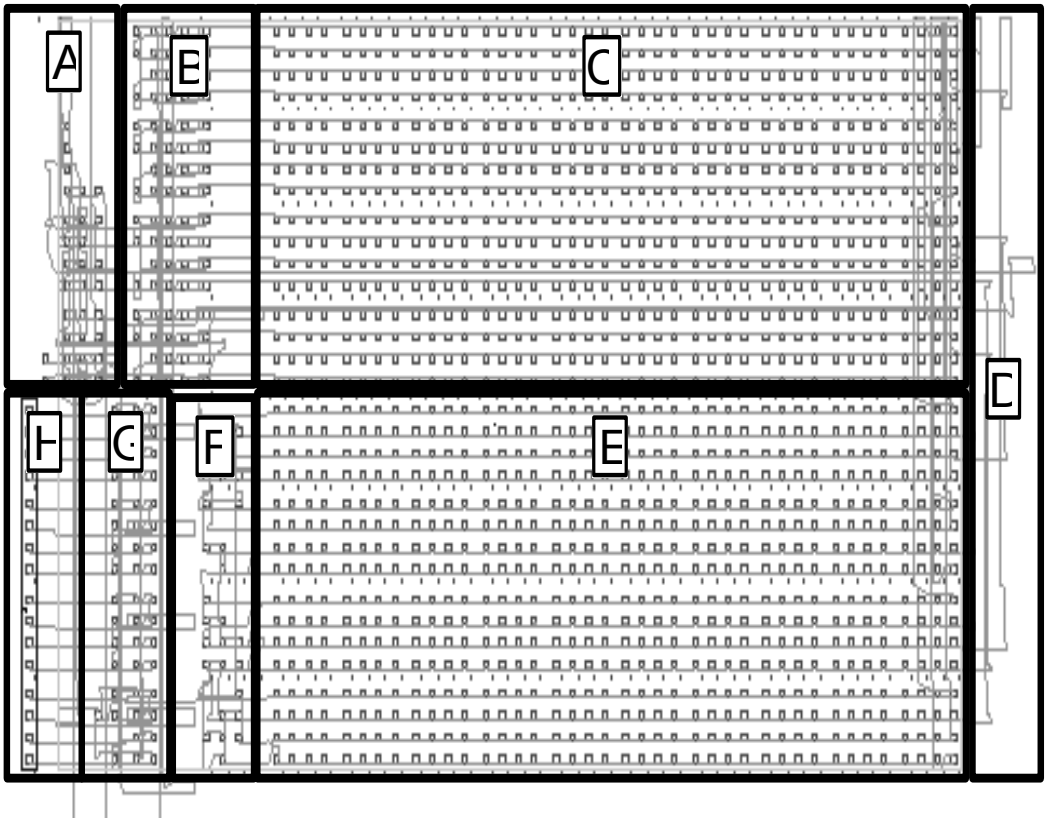
- **The clock rate at which a field-programmable gate array can be run on a problem is considerably slower than that of a contemporary serial microprocessor that might run a software version of the same problem.**
- **Thus, in order to advantageously use the Xilinx XC6216 field-programmable gate array, it is necessary to find a mapping of the fitness evaluation task onto the XC6216 that exploits at least some of the massive parallelism of the 4,096 cells of the XC6216.**

# MAPPING THE MINIMAL SORTING NETWORK PROBLEM TO AN FPGA



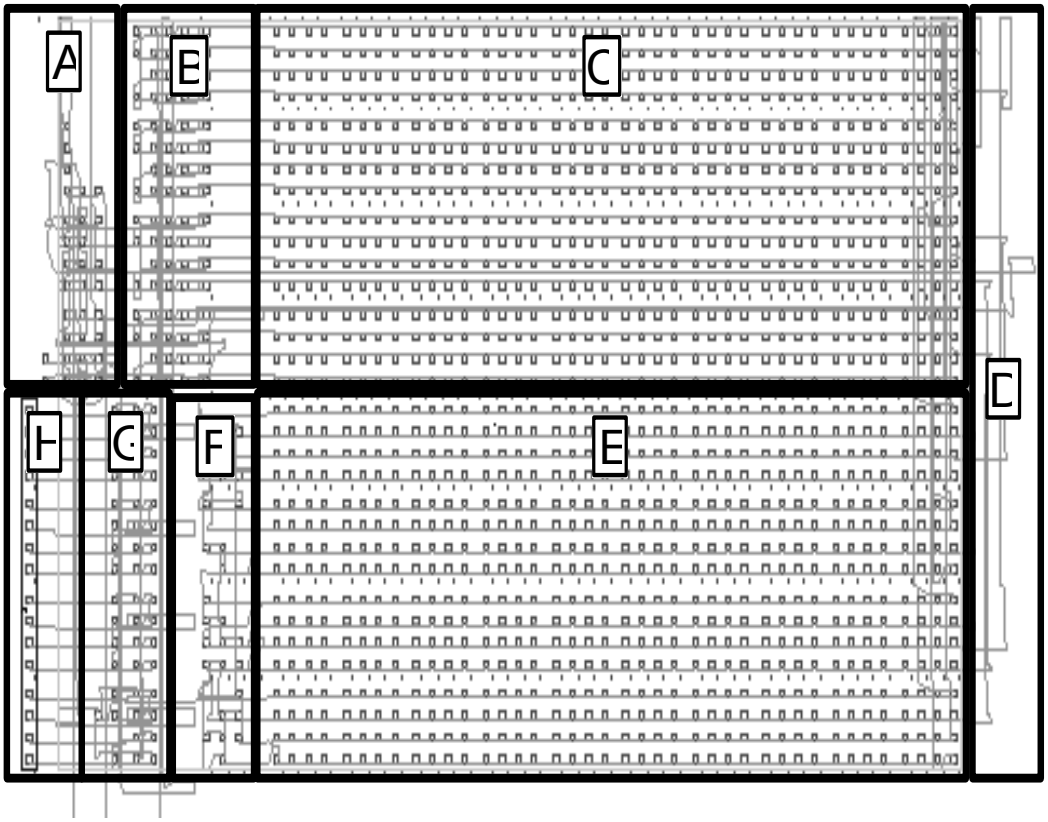

- **Broadly, fitness cases are created in area B, sorted in areas C, D, and E, and evaluated in F and G.**

# MAPPING — CONTINUED



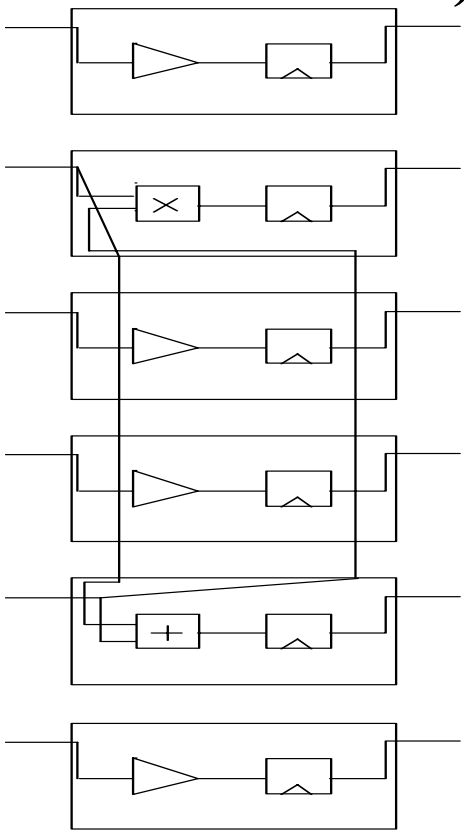

- **For a *k*-sorter ($k \leq 16$), a 16-bit counter B counts down from $2^k$-2 to 0 under control of control logic A.**
- **The vector of *k* bits resident in counter B on a given time step represents one fitness case of the sorting network problem.**
- **The vector of bits from counter B is fed into the first (leftmost) 16×1 vertical column of cells of the large 16×40 area C.**

# MAPPING — CONTINUED




- **Each 16×1 vertical column of cells in C (and each cell in similar area E) corresponds to one `COMPARE-EXCHANGE` operation of an individual candidate sorting network.**
- **The two large areas, C and E, together represent the individual candidate sorting network.**
- **The vector of 16 bits produced by the 40th (rightmost) sorting step of area C then proceeds to area D.**

# IMPLEMENTATION OF (COMPARE-EXCHANGE 2 5)



| Two Arguments | | Two Results | |
|---|---|---|---|
| $A_i$ | $A_j$ | $Ri$ | $R_j$ |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 |

# MAPPING — CONTINUED

- **For area C, each cell in a 16×1 vertical column is configured in one of three main ways.**
- **First, the logical function unit of exactly one of the 16 cells is configured as a two-argument Boolean AND function (corresponding to result $R_i$).**
- **Second, the logical function unit of exactly one other cell is configured as a two-argument Boolean OR function (corresponding to result $R_j$).**
  - **Bits i and j become sorted into the correct order by virtue of the fact that the single AND cell in each 16×1 vertical column always appears above the single OR cell.**
- **Third, the logical function units of 14 of the 16 cells are configured as "pass through" cells that horizontally pass their input from one vertical column to the next.**

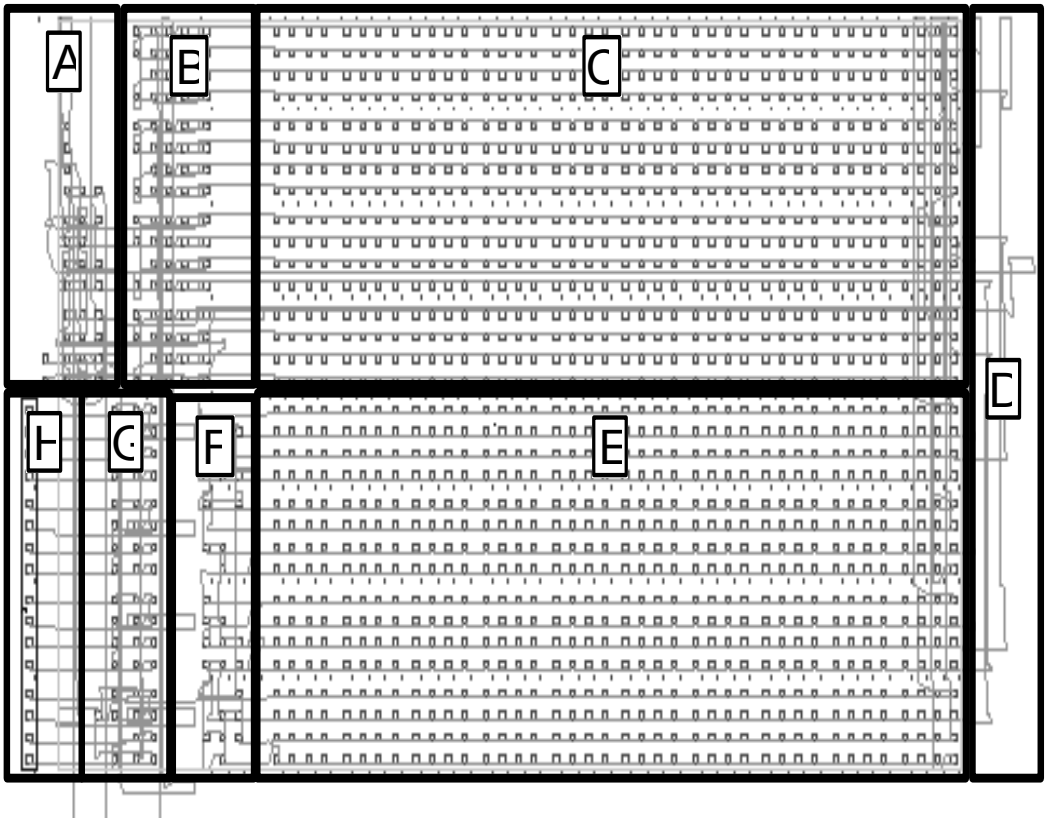
# MAPPING — CONTINUED

- **Every cell in the Xilinix XC6216 has the additional capacity of being able to convey one signal in each direction as a "fly over" signal that plays no role in the cell's own computation.**
- **Thus, the two "intervening" "pass through" cells (3 and 4) that lie between the AND and OR cells (1 and 5) is configured so that it conveys one signal vertically upwards and one signal vertically downwards as "fly over" signals.**
- **These "fly overs" of the two intervening cells (3 and 4) enable cell 2's input to be shared with cell 5 and cell 5's input to be shared with cell 2. Specifically, the input coming into cell 2 horizontally from the previous vertical column (i.e., from the left in the figure) is bifurcated so that it feeds both the two-argument AND in cell 2 and the two-argument OR in cell 5 (and similarly for the input coming into cell 5).**

# MAPPING — CONTINUED

- **There are additional subtypes each of AND and OR cells and "pass through" cells because all the cells in area E differ in chirality (handedness) from those in area C in that they receive their input from their right and deliver output to their left.**
- **Within each cell of areas C and E, the one-bit output of the cell's logical function unit is stored in a flip-flop.**
- **The contents of the 16 flip-flops in one vertical column become the inputs to the next vertical column on the next time step.**
- **The overall arrangement operates as an 87-stage pipeline (the 80 stages of areas C and E, the three stages of answer logic F, and four stages of padding at both ends of C and E).**

# MAPPING — CONTINUED




- **Area D is a U-turn area that channels the vector of 16 bits from the rightmost column of area C into the first (rightmost) column of the large 16×40 area E.**

# MAPPING — CONTINUED

- **The final output from area E is checked by answer logic F for whether the individual candidate sorting network has correctly rearranged the original incoming vector of bits so that all the 0's are above all the 1's.**
- **Answer logic F determines whether the 16 bits coming from the 80th column of the pipeline are properly sorted – that is, the bits are of the form $0^j1^{16-j}$.**
- **The 16-bit accumulator G is incremented by one if the bits are correctly sorted.**
  - **Note that the 16 bits of accumulator G are sufficient for tallying the number of correctly sorted fitness cases because the host computer starts counter B at $2^k$-2, thereby skipping the uninteresting fitness case consisting of all 1's (which cannot be incorrectly sorted by any network).**
- **The final value of raw fitness is reported in 16-bit register H after all the $2^k$ - 2 fitness cases have been processed.**

# MAPPING — CONTINUED

- **A "past zero" flip-flop is set when counter B counts down to 0.**
- **As B continues counting, it rolls over to $2^{16} - 1 = 65{,}535$ (for a 16-sorter) and continues counting down. When counter B reaches $2^{16} - 87$ (with the "past zero" flip-flop being set), control logic A stops further incrementation of accumulator G.**
- **The raw fitness from G appears in reporting register H and the "done bit" flip-flop is set to 1.**
- **The host computer polls this "done bit" to determine that the XC6216 has completed its fitness evaluation task for the current individual.**

# MAPPING — CONTINUED

- **The flip-flop toggle rate of the XC6216 chip is 220 MHz (i.e., about the same as a contemporary Pentium or PowerPC microprocessor device at the time of this work).**
- **The flip-flop toggle rate of the chip is an upper bound on the speed at which a field-programmable gate array can be run.**
- **In practice, the speed at which a FPGA is actually run is governed by the longest routing delay.**
- **It is common to see a 10-to-1 reduction in clock rate in FPGAs because of routing delays. Indeed, the current (unoptimized) version of the FPGA design for the sorting network problem is run at 20 MHz.**

# 6 WAYS THAT THIS DESIGN EXPLOITS PARALLELISM

- **First, the tasks performed by areas A, B, C, D, E, F, G, and H are examples of performing disparate tasks in parallel in physically different areas of the FPGA.**

- **Second, the two separate 32×64 areas operating in parallel on the chip are an example (at a higher level) of performing identical tasks in parallel in physically different areas of the FPGA.**
- **Third, the XC6216 evaluates the $2^k$ fitness cases independently of the activity of the host PC Pentium type computer (which simultaneously can prepare the next individual(s) for the XC6216). This is an example (at the highest level) of performing disparate tasks in parallel.**

# 6 WAYS THAT THIS DESIGN EXPLOITS PARALLELISM — CONTINUED

- **Fourth, the Boolean AND functions and OR functions of each COMPARE-EXCHANGE operation are performed in parallel (in each of the vertical columns of areas C and E).**
  - **This is an example of recasting a key operation (i.e., the COMPARE-EXCHANGE operation) as a bit-level operation so that the FPGA can be advantageously used.**
  - **It is also an example of performing two disparate operations (AND and OR) in parallel in physically different areas of the FPGA (i.e., different locations in the vertical columns of areas C and E).**

# 6 WAYS THAT THIS DESIGN EXPLOITS PARALLELISM — CONTINUED

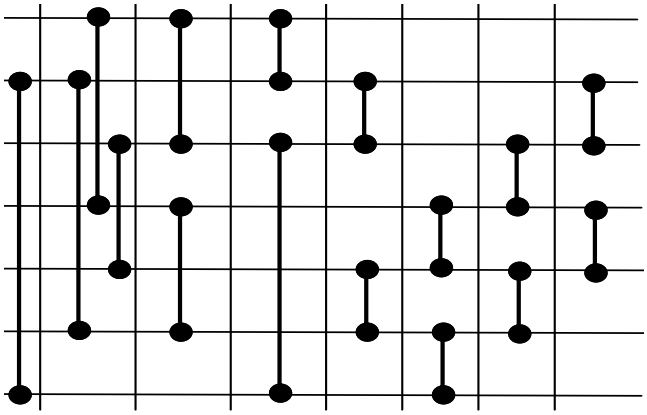
- **Fifth, numerous operations are performed in parallel inside control logic A, counter B, answer logic F, accumulator G, and reporting register H.**
  - **The parallelization inside answer logic F is especially advantageous because numerous sequential steps are required on a conventional serial microprocessor to determine whether *k* bits are properly sorted.**
  - **Answer logic F is an example of a multi-step task that is both successfully parallelized and pipelined on the FPGA.**

- **Sixth, most importantly, the 87-step pipeline (80 steps for areas C and E and 7 steps for answer logic F and accumulator G) enables 87 fitness cases to be processed in parallel in the pipeline.**

# RESULTS — MINIMAL SORTING NETWORK PROBLEM

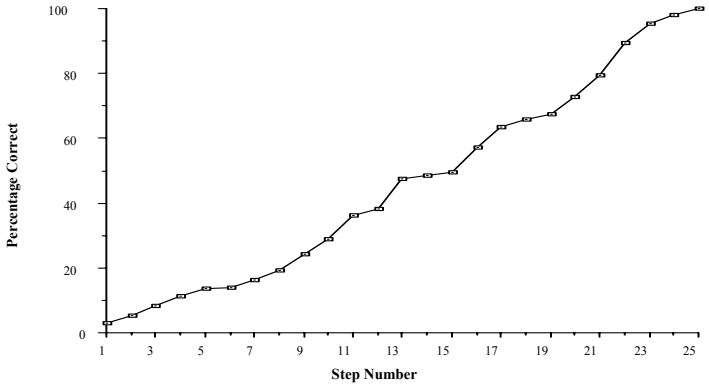
- **A 16-step 7-sorter was evolved that has two fewer steps than the sorting network described in O'Connor and Nelsons' patent (1962) and that has the same number of steps as the 7-sorter that was devised by Floyd and Knuth subsequent to the patent and described in Knuth 1973.**

## GENETICALLY EVOLVED 7-SORTER



# DEFAULT HIERARCHIES VERSUS EVOLUTIONARY INCREMENTALISM

# PERCENTAGE OF CORRECTLY SORTED FITNESS CASES AFTER EACH STEP FOR EVOLVED MINIMAL 9-SORTER




# HUMAN-DESIGNED 9-SORTER