5. ReAl Systems on Silicon

Programmable and application-specific integrated circuits

This chapter illustrates how resource arrays can be incorporated into integrated circuits, preferably into programmable ones. Conventional large programmable integrated circuits (field programmable gate arrays; FPGAs) comprise programmable cells (logic cells) that are connected by signal paths and switching matrices that are also programmable (fig. 5.1). The individual logic cells are of a comparatively basic structure. With a single cell, for example, a combinational operation with four to eight inputs and one or two flip-flops can be implemented. The amount of hardware needed for programming the logic cells and the interconnecting networks is significant. In order to implement certain functional requirements it is often required to have more than ten times the number of transistors in comparison to true application-specific circuits (on which the circuitry is optimized down to the transistor).

Therefore, especially complex and frequently used functions are implemented on some FPGAs the hard (non-progammable) way (this concerns interface controllers and so on as well as complete processors). Such circuits are however comparatively expensive and less universal. For example, it is possible that the embedded hard processor is too large for the particular application (circuit is too expensive) or that it does not perform well enough (complex additional hardware is necessary).

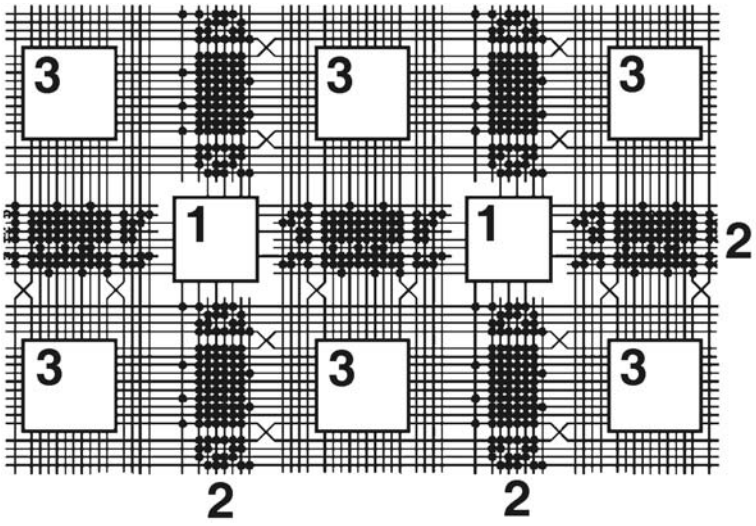


Fig. 5.1 Structure of an conventional FPGA (source: Xilinx). 1 - switching matrices; 2 - programmable signal paths; 3 - programmable logic cells.

ReAl FPGAs

In order to avoid these disadvantages, resource cells can be provided that comprise processing circuitry and a certain amount of memory. The resource cells are not comprising programmable logic cells but transistors or gates hardwired together. Only the mode of operation and the processing width can be set up under program control[1)].

1): Here, the term "programing" means running an appropriate program, not the conventional programming of an FPGA.

A typical resource cell:

- Function: a universal arithmetic logic unit (ALU).
- Processing width: can be set up under program control (for example, up to 32 bits).
- Concatenation: will be supported for all parameters.
- Memory: for example, 1k words of 32 bits.

The signal paths are optimized according to typical ways the resources are operated. They can be designed relatively simply because it is not necessary to support arbitrary connections among all cells on the integrated circuit. Figs. 5.2 to 5.8 show different arrangements of resource cells. Such arrangements can be provided on programmable as well as on application-specific integrated circuits.

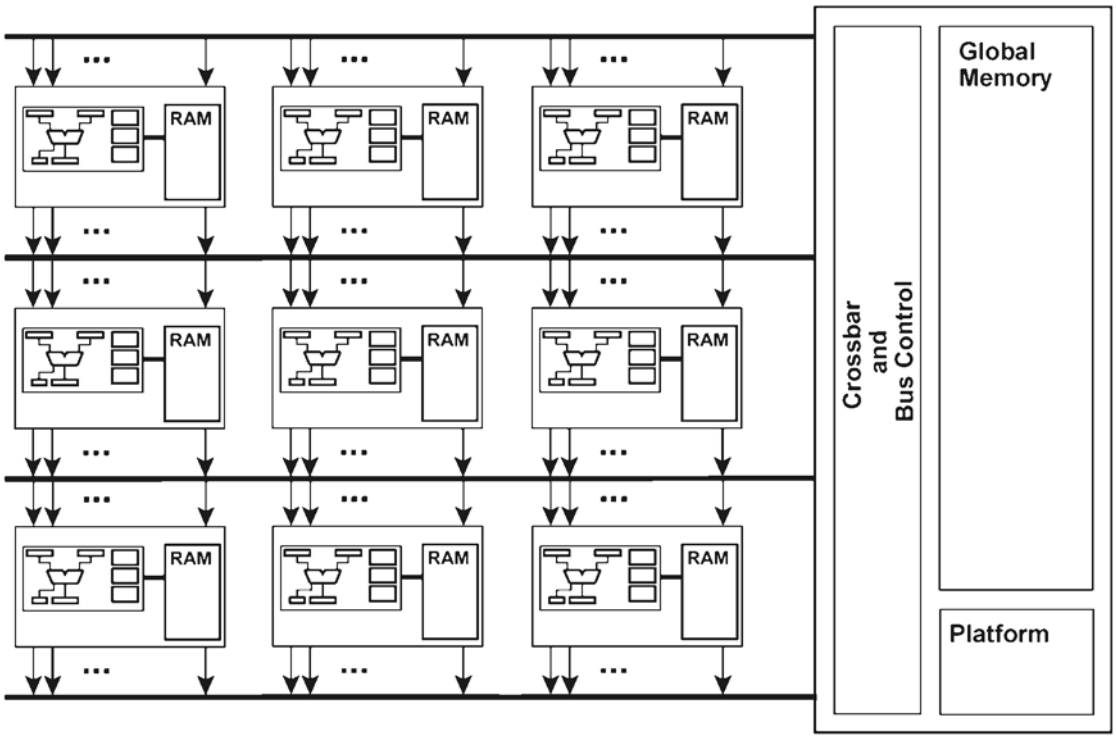


Fig. 5.2 A arrangement of resource cells on a integrated circuit.

The resource cells are connected together via bus systems. Each bus system is capable of supplying a row of resource cells with parameters and to pass on the results of the resource cell row arranged above. The bus control provides the connection between the individual bus systems. On the integrated circuit, a common (global) memory subsystem as well as the platform circuitry is arranged.

Fig. 5.3 shows how such an arrangement can be expanded by external memories. Often, very large memory capacities (megabytes to gigabytes) are required. It is then advantageous to utilize the high volume, inexpensive memory circuits (for example, DDR-DRAMs) or memory modules.

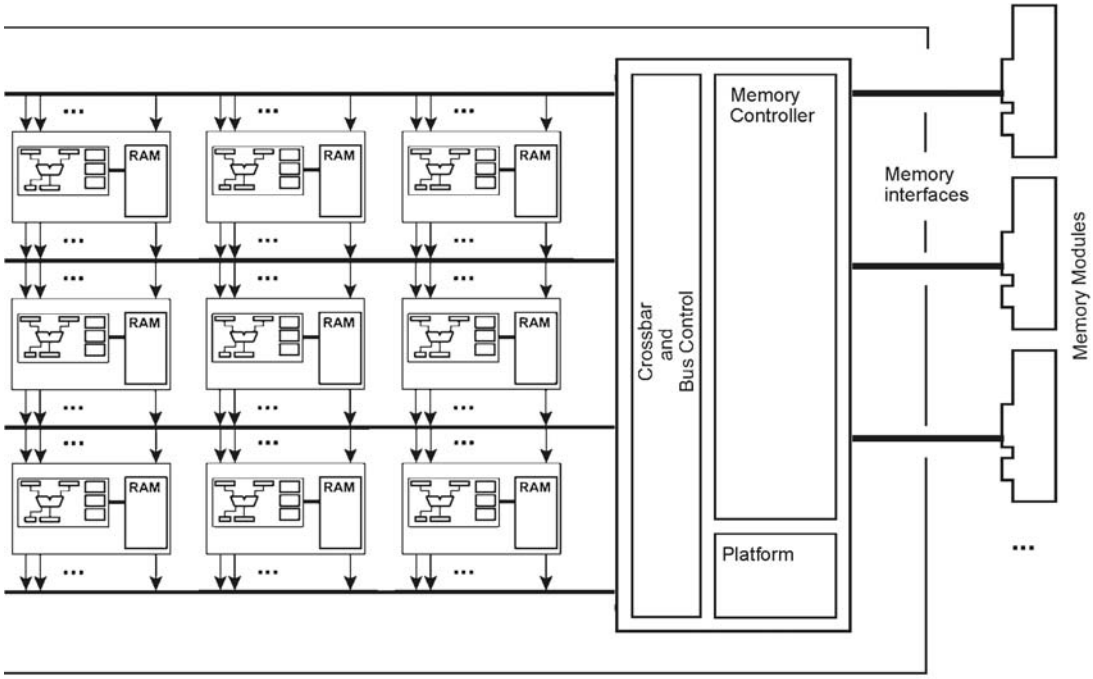


Fig. 5.3 An integrated circuit with resource cells and external memory.

Fig. 5.4 illustrates the connection of resource cells as inverted tree structures (in the example: as binary trees). The evaluation of nested expressions can be easily mapped onto inverted tree structures. The advantage: the connections between the cells of the tree structure are simple point-to-point connections. They are short and require no programming provisions on the integrated circuit. In order to utilize the silicon real estate well, according to fig. 5.5 two tree structures are arranged in opposite directions. For loading the operands and for retrieving the results, two bus systems are provided. The inputs and outputs of the consecutively arranged inverted tree structures are connected in a reshuffled arrangement to the bus systems (the first bus system is connected to the operand inputs of the first tree and the result outputs of the second tree; the second bus system is connected to the operand inputs of the second tree and the result outputs of the first tree and so on). As a result of the reshuffled and opposite arrangement, each of the bus systems is utilized in the same way for write and read access (uniform workload distribution). The two bus systems are also uniformly loaded with the same number of receivers and drivers.

A deep tree structure cannot always be utilized appropriately. As the nesting depth in application programs is not that large[1)], a combination between a binary tree and a stack is advantageous. The last operation unit of the tree receives one of its operands from the stack and pushes its results onto the stack. In order to accelerate the stack access, a stack cache is provided for each tree. The stack cache can be a conventional cache memory array that is addressed by a stack pointer (fig. 5.6). Fig. 5.7 shows that by opposite arrangement of such tree structures (according to fig. 5.5), the silicon real estate can be utilized very well[2)].

---

1): It is usually not more than 32. Well-proven computer architectures provide for nesting levels between 8 and 32.

2): The free areas in fig. 5.7 result only because the illustration is not to scale. In practice, they can be utilized, for example, for larger stack cache memories.

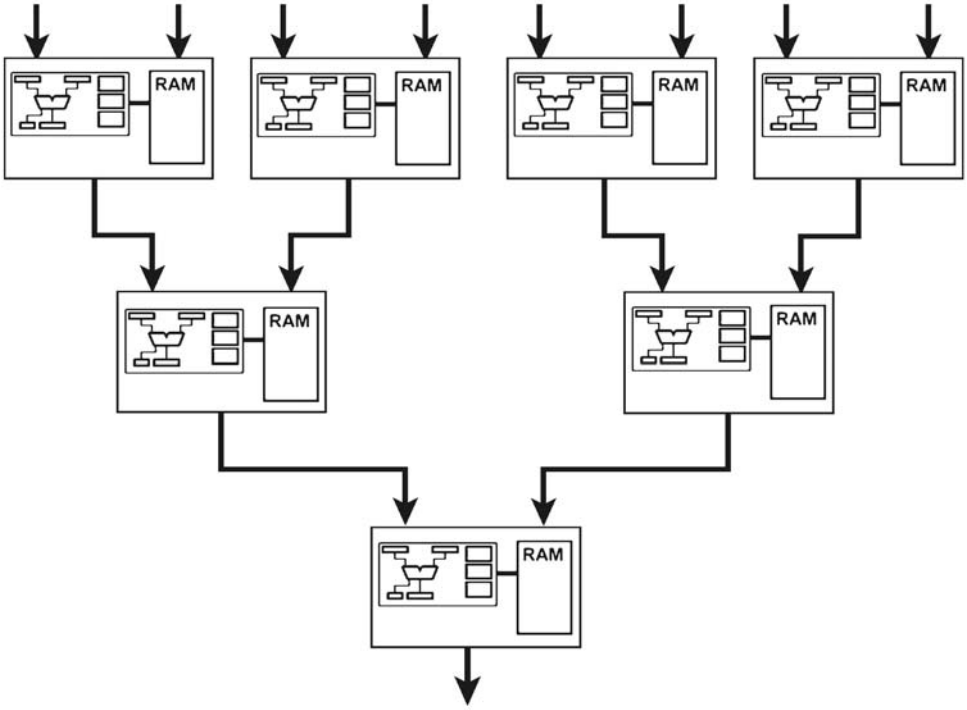


Fig. 5.4 Resource cells connected to an inverted binary tree.

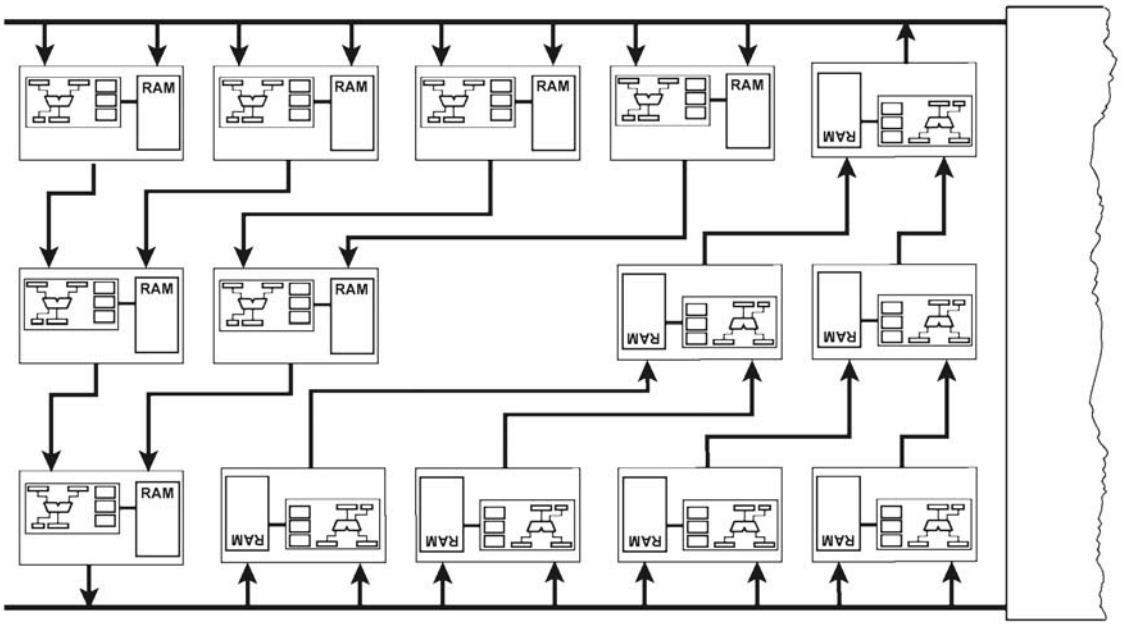


Fig. 5.5 An integrated circuit containing resource cells connected to inverted binary trees.

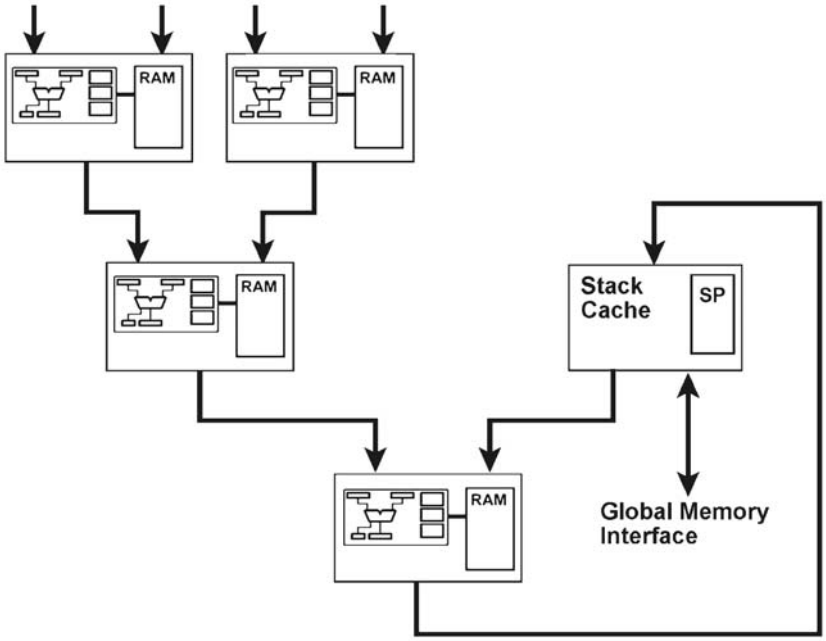


Fig. 5.6 An inverted binary tree configuration supplemented by a stack cache.

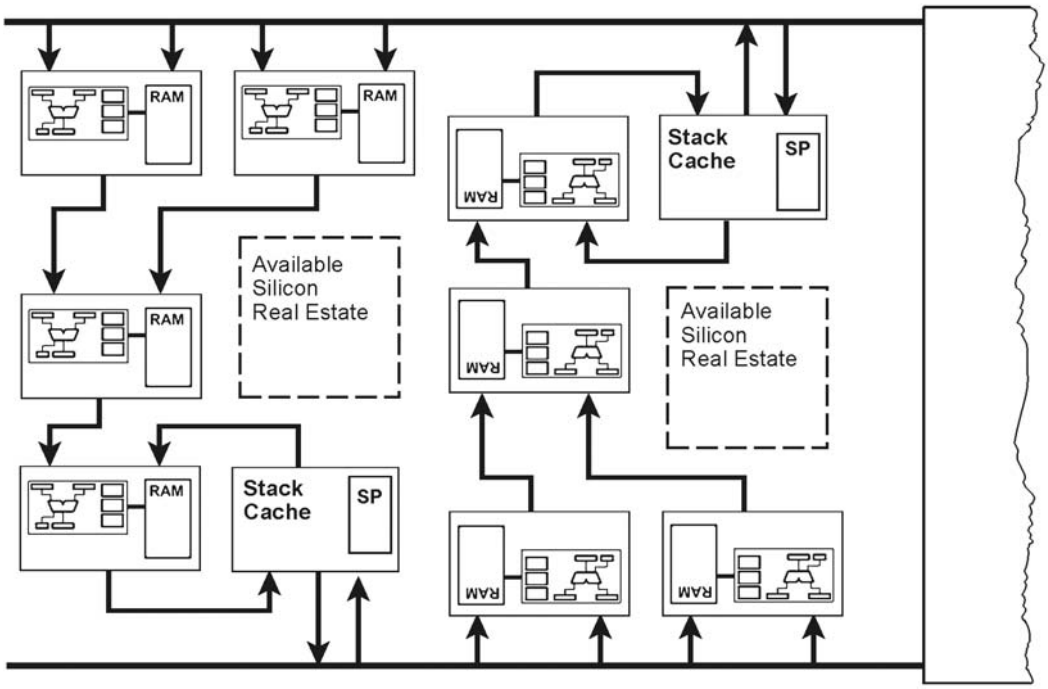


Fig. 5.7 An integrated circuit containing resource cells connected to inverted binary trees and supplemented by stack caches.

General-purpose high-speed interfaces connected via switching hubs are an alternative to bus systems. They are supported in some FPGA families. Fig. 5.8 illustrates an integrated circuit whose resource cells are connected by point-to-point interfaces to hubs that, in turn, are connected together by programmable (global) signal paths.

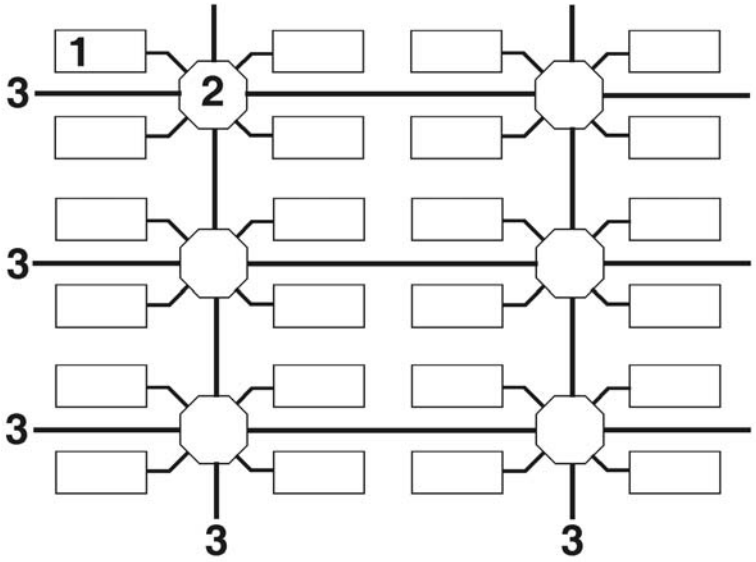


Fig. 5.8 Resource cells connected via point-to-point interfaces and switching hubs. 1 - resource cell; 2 - switching hub; 3 - point-to-point-interface.

ReAl application development

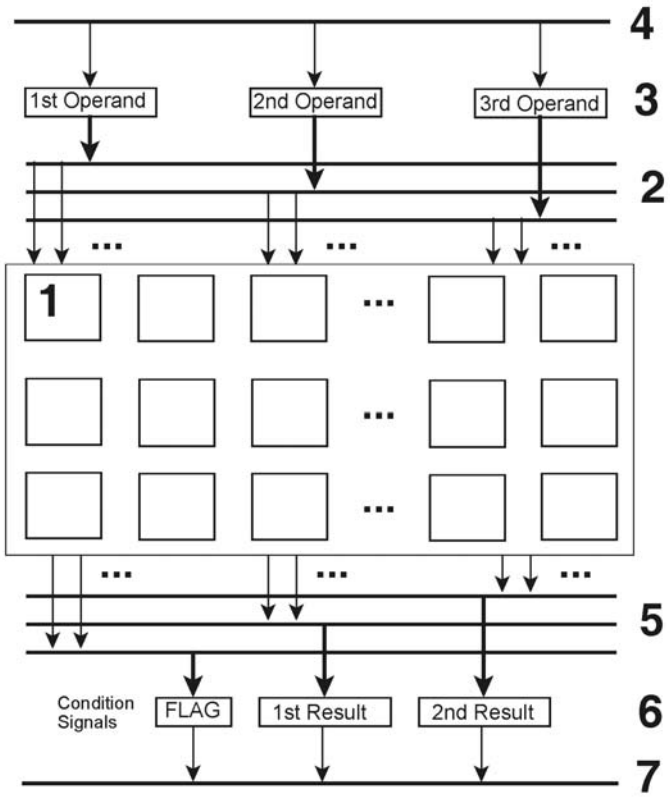
Conventionally, an applications developer must decide which functions are to be implemented by software (on an embedded processor) and which by hardware (by means of programmable logic). In ReAl systems there is no such separation. Instead, by the development system and – at run time – the built-in platform a dedicated resource configuration is generated and modified as needed dynamically (processors are generated as needed (on the fly) and are optionally dismantled again).

A typical development flow:

1. The application problem is described with suitable means (flowcharts, state diagrams, programming languages and the like).
2. The development system generates appropriate ReAl code (that requests resources, supplies them with parameters and so on).
3. Accordingly, a circuit with a suitable resource configuration is selected and the ReAl code is modified, if necessary.

Programmable (soft) resources

General-purpose hard resources (for example, ALU structures) are not suitable for certain applications. Therefore, it is often advantageous to provide areas with conventional programmable logic cells in order to be able to build as needed arbitrary application-specific circuitry. Such arrangements however cannot cooperate easily with other (hard) ReAl resources. Therefore, these cell areas are surrounded by hardwired interfaces that correspond to those of the hard resource cells, like parameter registers, concatenation hardware, bus interfaces and so on (fig. 5.9).



1 - programmable logic cell; 2 - internal operand paths (programmable); 3 - operand registers; 4 - global operand interface; 5 - internal result paths (programmable); 6 - result registers; 7 - global result interface.

Fig. 5.9 A programmable (soft) resource cell.

Such soft resource cells are connected like the hard resource cells to the global signal paths of the integrated circuit, for example, by means of the bus systems illustrated in fig. 5.2[1)]. The registers of the global interfaces are accompanied by the necessary control provision and, if required, by additional circuitry for supporting concatenation (like pointer and state registers). Since this circuitry is implemented the hard way, it requires only comparatively little silicon area.

There are FPGA circuits in which the programming data of the logic cells and of the connecting signal paths are held in RAM structures. Conventionally, they are programmed anew after each power-on (by loading the RAM content) before actual operation begins. During operation (at run time) reprogramming is not possible. In contrast to this, ReAl integrated circuits based on hard resource cells can be designed such that they can be reprogrammed even at run time (because the arrangement of hard cells, for example, according to fig. 5.2, remains unchanged while the soft cells are reprogrammed). Reprogramming can be initiated by s-operators and u-operators (resource administration) as well as c-operators and d-operators (concatenation). For this purpose, the programming signal paths of the soft cells in question are to be connected to corresponding hard structures, for example, to appropriate platform circuitry.

---

1): According to fig. 5.2 the inputs of the operand registers are connected to the upper bus system, the outputs of the result registers are connected to the lower one, respectively.