The Roles of FPGA's in Reprogrammable Systems

SCOTT HAUCK, MEMBER, IEEE

Reprogrammable systems based on field programmable gate arrays are revolutionizing some forms of computation and digital logic. As a logic emulation system, they provide orders of magnitude faster computation than software simulation. As a custom-computing machine, they achieve the highest performance implementation for many types of applications. As a multimode system, they yield significant hardware savings and provide truly generic hardware.

In this paper, we discuss the promise and problems of reprogrammable systems. This includes an overview of the chip and system architectures of reprogrammable systems as well as the applications of these systems. We also discuss the challenges and opportunities of future reprogrammable systems.

***Keywords*—** *Adaptive computing, custom computing, FPGA, logic emulation, multi-FPGA systems, reconfigurable computing.*

I. Introduction

In the mid-1980's, a new technology for implementing digital logic was introduced: the field programmable gate array (FPGA). These devices could be viewed as either small, slow gate arrays (MPGA's) or large, expensive programmable logic devices (PLD's). FPGA's were capable of implementing significantly more logic than PLD's, especially because they could implement multilevel logic, while most PLD's were optimized for two-level logic. Although they did not have the capacity of MPGA's, they also did not have to be custom fabricated, greatly lowering the costs for low-volume parts and avoiding long fabrication delays. While many of the FPGA's were configured by static random access memory (SRAM) cells in the array, this was generally viewed as a liability by potential customers who worried over the chip's volatility. Antifuse-based FPGA's also were developed and for many applications were much more attractive, both because they tended to be smaller and faster due to less programming overhead and because there was no volatility to the configuration.

Manuscript received May 5, 1997; revised January 20, 1998. This work was supported in part by the Defense Advanced Research Project Agency under Contract DABT63-97-C-0035 and in part by the National Science Foundation under Grants CDA-9703228 and MIP-9616572.

The author is with the Department of Electrical and Computer Engineering, Northwestern University, Evanston, IL 60208-3118 USA.

Publisher Item Identifier S 0018-9219(98)02669-3.

In the late 1980's and early 1990's, there was a growing realization that the volatility of SRAM-based FPGA's was not a liability but was in fact the key to many new types of applications. Since the programming of such an FPGA could be changed by a completely electrical process, much as a standard processor can be configured to run many programs, SRAM-based FPGA's have become the workhorse of many new reprogrammable applications. Some uses of reprogrammability are simple extensions of the standard logic implementation tasks for which the FPGA's were originally designed. An FPGA plus several different configurations stored in read-only memory (ROM) could be used for multimode hardware, with the functions on the chip changed in reaction to the current demands. Also, boards constructed purely from FPGA's, microcontrollers, and other reprogrammable parts could be truly generic hardware, allowing a single board to be reprogrammed to serve many different applications.

Some of the most exciting new uses of FPGA's move beyond the implementation of digital logic and instead harness large numbers of FPGA's as a general-purpose computation medium. The circuit mapped onto the FPGA's need not be standard hardware equations but can even be operations from algorithms and general computations. While these FPGA-based custom-computing machines may not challenge the performance of microprocessors for all applications, for computations of the right form, an FPGA-based machine can offer extremely high performance, surpassing any other programmable solution. Although a custom hardware implementation will be able to beat the power of any generic programmable system, and thus there must always be a faster solution than a multi-FPGA system, the fact is that few applications will ever merit the expense of creating application-specific solutions. An FPGA-based computing machine, which can be reprogrammed like a standard workstation, offers the highest realizable performance for many different applications. In a sense, it is a hardware supercomputer, surpassing traditional machine architectures for certain applications. This potential has been realized by many different research machines. The Splash system [50] has provided performance on genetic string matching that is almost 200 times greater than all

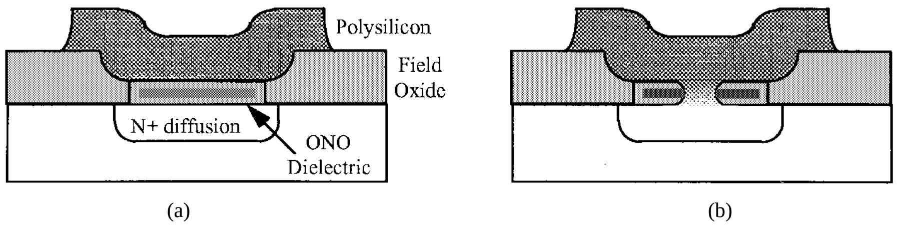


Fig. 1. Actel's programmable low-impedance circuit element (PLICE). As shown in (a), an unblown antifuse has an oxide–nitride–oxide (ONO) dielectric preventing current from flowing between diffusion and polysilicon. The antifuse can be blown by applying a 16-V pulse across the dielectric. This melts the dielectric, allowing a conducting channel to be formed (b). Current is then free to flow between the diffusion and the polysilicon [1], [54].

other supercomputer implementations. The DECPeRLe-1 system [133] has demonstrated world-record performance for many other applications, including RSA cryptography.

One of the applications of multi-FPGA systems with the greatest potential is logic emulation. The designers of a custom chip need to verify that the circuit they have designed actually behaves as desired. Software simulation and prototyping have been the traditional solution to this problem. As chip designs become more complex, however, software simulation is only able to test an ever decreasing portion of the chips' computations, and it is quite expensive in time and money to debug by repeated prototype fabrications. The solution is logic emulation, the mapping of the circuit under test onto a multi-FPGA system. Since the logic is implemented in the FPGA's in the system, the emulation can run at near real time, yielding test cycles several orders of magnitude faster than software simulation, yet with none of the time delays and inflexibility of prototype fabrications. These benefits have led many of the advanced microprocessor manufacturers to include logic emulation in their validation process.

In this paper, we discuss the different applications and types of reprogrammable systems. In Section II, we present an overview of FPGA architectures as well as field programmable interconnect components (FPIC's). Then, Section III details what kinds of opportunities these devices provide for new types of systems. We then categorize the types of reprogrammable systems in Section IV, including coprocessors and multi-FPGA systems. Section V describes in depth the different multi-FPGA systems, highlighting their important features. Last, Sections VI and VII conclude with an overview of the status of reprogrammable systems and how they are likely to evolve. Note that this paper is not meant to be a catalog of every existing reprogrammable architecture and application. We instead focus on some of the more important aspects of these systems in order to give an overview of the field.

II. FPGA Technology

One of the most common field programmable elements is PLD's. PLD's concentrate primarily on two-level, sum-of-products implementations of logic functions. They have simple routing structures with predictable delays. Since they are completely prefabricated, they are ready to use in seconds, avoiding long delays for chip fabrication. FPGA's

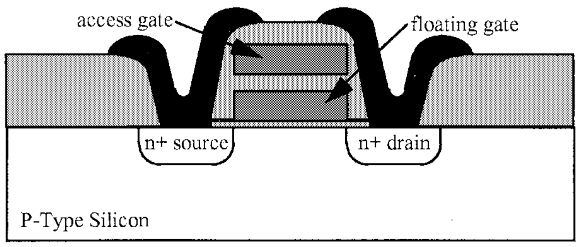


Fig. 2. Floating gate structure for EPROM/EEPROM. The floating gate is completely isolated. An unprogrammed transistor, with no charge on the floating gate, operates the same as a normal n-transistor, with the access gate as the transistor's gate. To program the transistor, a high voltage on the access gate plus a lower voltage on the drain accelerates electrons from the source fast enough to travel across the gate oxide insulator to the floating gate. This negative charge then prevents the access gate from closing the source–drain connection during normal operation. To erase, EPROM uses ultraviolet light to accelerate electrons off the floating gate, while EEPROM removes electrons by a technique similar to programming but with the opposite polarity on the access gate [134], [148].

are also completely prefabricated, but instead of two-level logic, they are optimized for multilevel circuits. This allows them to handle much more complex circuits on a single chip, but it often sacrifices the predictable delays of PLD's. Note that FPGA's are sometimes considered another form of PLD, often under the heading of complex PLD.

Just as in PLD's, FPGA's are completely prefabricated and contain special features for customization. These configuration points are normally SRAM cells, EPROM, EEPROM, or antifuses. Antifuses are one-time programmable devices (Fig. 1), which when "blown" create a connection. When they are "unblown," no current can flow between their terminals (thus, it is an "anti" fuse, since its behavior is opposite to a standard fuse). Because the configuration of an antifuse is permanent, antifuse-based FPGA's are one-time programmable, while SRAM-based FPGA's are reprogrammable, even in the target system. Since SRAM's are volatile, an SRAM-based FPGA must be reprogrammed every time the system is powered up, usually from a ROM included in the circuit to hold configuration files. Note that FPGA's often have on-chip control circuitry to load this configuration data automatically. EEPROM/EPROM (Fig. 2) are devices somewhere between SRAM and antifuse in their features. The programming of an EEPROM/EPROM is retained even when the power is turned off, avoiding the need to reprogram the chip at power-

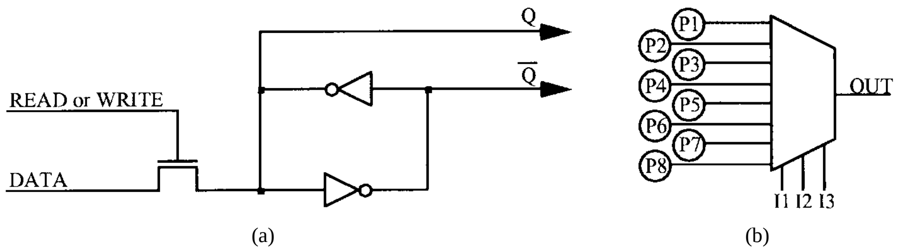


Fig. 3. Programming bit for (a) SRAM-based FPGA's [145] and (b) a three-input lookup table.

up, while their configuration can be changed electrically. However, the high voltages required to program the device often mean that they are not reprogrammed in the target system.

SRAM cells are larger than antifuses and EEPROM/EPROM, meaning that SRAM-based FPGA's will have fewer configuration points than FPGA's using other programming technologies. However, SRAM-based FPGA's have numerous advantages. Since they are easily reprogrammable, their configurations can be changed for bug fixes or upgrades. Thus, they provide an ideal prototyping medium. Also, these devices can be used in situations where they can expect to have numerous different configurations, such as multimode systems and reconfigurable computing machines. More details on such applications are included later in this paper. Because antifuse-based FPGA's are only one-time programmable, they are generally not used in reprogrammable systems. EEPROM/EPROM devices could potentially be reprogrammed in-system, although in general this feature is not widely used. Thus, this paper will concentrate solely on SRAM-based FPGA's.

There are many different types of FPGA's, with many different structures. Instead of discussing all of them here, which would be quite involved, this section will present two representative FPGA's. Details on many others can be found elsewhere [21], [26], [75], [103], [112], [127]. Note that reconfigurable systems can often employ non-FPGA reconfigurable elements; these will be described in Section V.

In SRAM-based FPGA's, memory cells are scattered throughout the FPGA. As shown in Fig. 3(a), a pair of cross-coupled inverters will sustain whatever value is programmed onto them. A single n-transistor gate is provided for either writing a value or reading a value back out. The ratio of sizes between the transistor and the upper inverter is set to allow values sent through the n-transistor to overpower the inverter. The read-back feature is used during debugging to determine the current state of the system. The actual control of the FPGA is handled by the Q and $\overline{Q}$ outputs. One simple application of an SRAM bit is to have the Q terminal connected to the gate of an n-transistor. If a "1" is assigned to the programming bit, the transistor is closed, and values can pass between the source and drain. If a "0" is assigned, the transistor is opened, and values cannot pass. Thus, this construct operates similarly to an antifuse, though it requires much

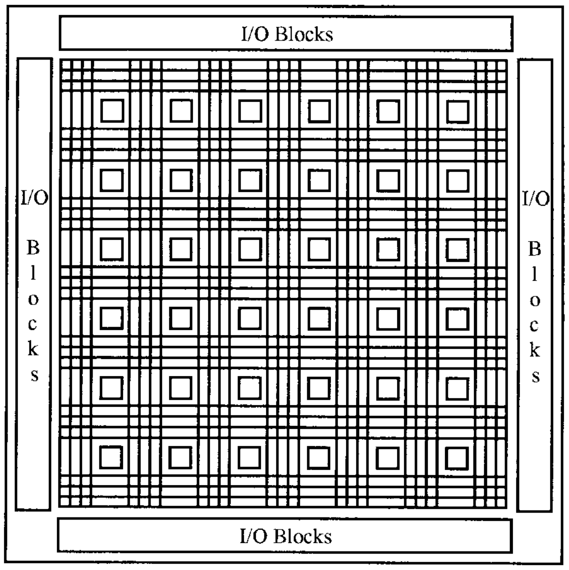


Fig. 4. The Xilinx 4000 series FPGA structure [145]. Logic blocks are surrounded by horizontal and vertical routing channels.

more area. One of the most useful SRAM-based structures is the lookup table (LUT). By connecting 2^N programming bits to a multiplexer [Fig. 3(b)], any N-input combinational Boolean function can be implemented. Although it can require a large number of programming bits for large N, LUT's of up to five inputs can provide a flexible, powerful function implementation medium.

One of the best known FPGA's is the Xilinx logic cell array [126], [145]. In this section, we will describe their third-generation FPGA, the Xilinx 4000 series. The Xilinx array is an "Island-style" FPGA [127] with logic cells embedded in a general routing structure that permits arbitrary point-to-point communication (Fig. 4). The only requirement for good routing in this structure is that the source and destinations be relatively close together. Details of the routing structure are shown in Fig. 5. Each of the inputs of the cell (F1-F4, G1-G4, C1-C4, K) comes from one of a set of tracks adjacent to that cell. The outputs are similar (X, XQ, Y, YQ), except that they have the choice of both horizontal and vertical tracks. The routing structure is made up of three lengths of lines. Single-length lines travel the height of a single cell, where they then enter a switch matrix [Fig. 6(b)]. The switch matrix allows this signal to travel out vertically and/or horizontally from the switch matrix. Thus, multiple single-length lines can be cascaded together to travel longer distances. Double-length

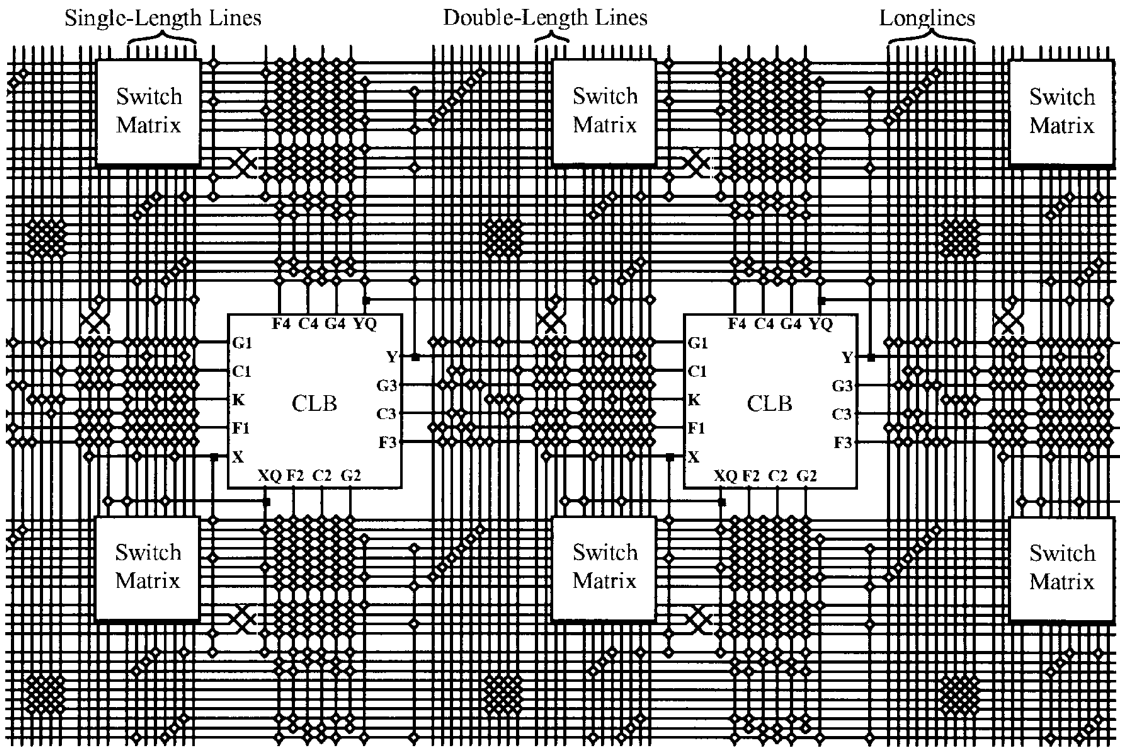


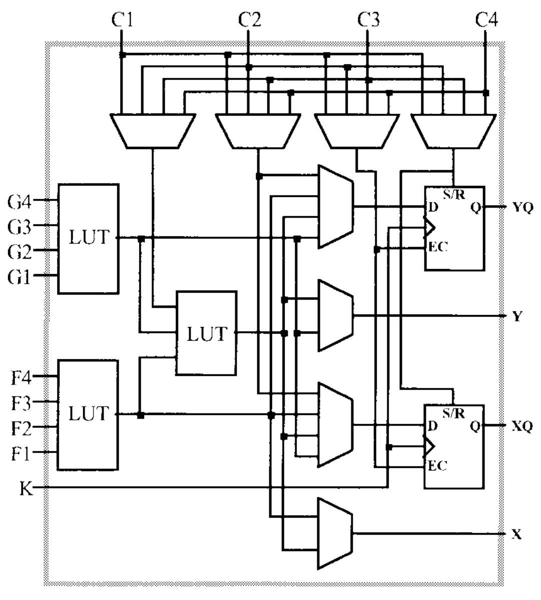
Fig. 5. Details of the Xilinx 4000 series routing structure [145]. The configurable logic blocks (CLB's) are surrounded by vertical and horizontal routing channels containing single-length lines, double-length lines, and long lines. Empty diamonds represent programmable connections between perpendicular signal lines (signal lines on opposite sides of the diamonds are always connected).

lines are similar, except that they travel the height of two cells before entering a switch matrix (notice that only half the double-length lines enter a switch matrix, and there is a twist in the middle of the line). Thus, double-length lines are useful for longer distance routing, traversing two cell heights without the extra delay and the wasted configuration sites of an intermediate switch matrix. Last, long lines are lines that go half the chip height and do not enter the switch matrix. In this way, routes of very long distance can be accommodated efficiently. With this rich sea of routing resources, the Xilinx 4000 series is able to handle fairly arbitrary routing demands, though mappings emphasizing local communication will still be handled more efficiently.

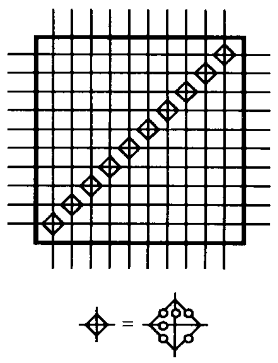
As shown in Fig. 6(a), the Xilinx 4000 series logic cell is made up of three LUT's, two programmable flip-flops, and multiple programmable multiplexers. The LUT's allow arbitrary combinational functions of its inputs to be created. Thus, the structure shown can perform any function of five inputs (using all three LUT's, with the F and G inputs identical), any two functions of four inputs (the two four-input LUT's used independently), or some functions of up to nine inputs (using all three LUT's, with the F and G inputs different). SRAM-controlled multiplexers then can route these signals out the X and Y outputs, as well as to the two flip-flops. The inputs at the top (C1-C4) provide enable and set or reset signals to the flip-flops, a direct connection to the flip-flop inputs, and the third input to the three-input LUT. This structure yields a very powerful method of implementing arbitrary, complex digital logic. Note that there are several additional features of the Xilinx FPGA not shown in these figures, including support for embedded memories and carry chains.

While many SRAM-based FPGA's are designed like the Xilinx architecture, with a routing structure optimized for arbitrary, long-distance communications, several other FPGA's concentrate instead on local communication. The "cellular"-style FPGA's [127] feature fast, local communication resources at the expense of more global, long-distance signals. As shown in Fig. 7, the CLi FPGA [75] has an array of cells, with a limited number of routing resources running horizontally and vertically between the cells. There is one local communication bus on each side of the cell. It runs the height of eight cells, at which point it enters a repeater. Express buses are similar to local buses, except that there are no connections between the express buses and the cells. The repeaters allow access to the express buses. These repeaters can be programmed to connect together any of the two local buses and two express buses connected to it. Thus, limited global communication can be accomplished on the local and express buses, with the local buses allowing shorter distance communications and connections to the cells while express buses allow longer distance connections between local buses.

While the local and global buses allow some of the flexibility of the Xilinx FPGA's arbitrary routing structure, there are significantly fewer buses in the CLi FPGA than are present in the Xilinx FPGA. The CLi FPGA instead features a large number of local communication resources.



(a)



(b)

Fig. 6. Details of the Xilinx (a) CLB and [(b), top] switchbox [145]. The multiplexers, LUT's, and latches in the CLB are configured by SRAM bits. Diamonds in the switchbox represent six individual connections [(b), bottom], allowing any permutation of connections among the four signals incident to the diamond.

As shown in Fig. 8, each cell receives two signals from each of its four neighbors. It then sends the same two outputs (A and B) to all of its neighbors. That is, the cell one to the north will send signals AN and BN, and the cell one to the south will send AS and BS, while both will receive the same signals A and B. The input signals become the inputs to the logic cell (Fig. 9).

Instead of Xilinx's LUT's, which require many programming bits per cell, the CLi logic block is much simpler. It has multiplexers controlled by SRAM bits, which select one each of the A and B outputs of the neighboring cells. These are then fed into AND and XOR gates within the cell, as well as into a flip-flop. Although the possible functions are complex, notice that there is a path leading to the B output that produces the NAND of the selected A and B inputs, sending it out the B output. This path is enabled by setting the two 2 : 1 multiplexers to their constant input and setting B's output multiplexer to the third input from the top. Thus, the cell is functionally complete. Also, with the XOR path leading to output A, the cell can efficiently implement a half-adder. The cell can perform a pure routing function by connecting one of the A inputs to the A output and one of the B inputs to the B output, or vice-versa. This routing function is created by setting the two 2 : 1 multiplexers to their constant inputs and setting A's and B's output multiplexer to either of their top two inputs. There are also provisions for bringing in or sending out a signal on one or more of the neighboring local buses (NS1, NS2, EW1, EW2). Note that since there is only a single wire connecting the bus terminals, there can only be a single signal sent to or received from the local buses. If more than one of the buses is connected to the cell, they will be coupled together. Thus, the cell can take a signal running horizontally on an EW local bus and send it vertically on an NS local bus without using up the cell's logic unit. By bringing a signal in from the local buses, however, the cell can implement two three-input functions.

The major differences between the Island-style architecture of the Xilinx 4000 series and the cellular style of the CLi FPGA is in their routing structure and cell granularity. The Xilinx 4000 series is optimized for complex, irregular random logic. It features a powerful routing structure optimized for arbitrary global routing and large logic cells capable of providing arbitrary four- and five-input functions. This provides a very flexible architecture, though one that requires many programming bits per cell (and thus cells that take up a large portion of the chip area). In contrast, the CLi architecture is optimized for highly local, pipelined circuits such as systolic arrays and bit-serial arithmetic. Thus, it emphasizes local communication at the expense of global routing and has simple cells. Because of the very simple logic cells, there will be many more CLi cells on a chip than will be found in the Xilinx FPGA, yielding a greater logic capacity for those circuits that match the FPGA's structure. Because of the restricted routing, the CLi FPGA is much harder to map to automatically than the Xilinx 4000 series, though the simplicity of the CLi architecture makes it easier for a human designer to hand-map to the CLi's structure. Thus, in general, cellular architectures tend to appeal to designers with appropriate circuit structures who are willing to spend the effort to hand-map their circuits to the FPGA, while the Xilinx 4000 series is more appropriate for handling random-logic tasks and automatically mapped circuits.

Compared with technologies such as full-custom standard cells and MPGA's, FPGA's will in general be slower and

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.