Software-Defined Hardware for Software-Defined Radios

Using programmable logic in Amateur Radio applications.

By John B. Stephensen, KD6OZH

Recently, I decided to upgrade my homebrew HF transceiver.[1] The goals of the new design were to replace most of the analog filtering and analog-control circuitry with software using digital signal processing (DSP) and to provide separate transmit and receive signal processing for full-duplex operation with amateur satellites. I also suspected that DSP could be used to improve AGC action and noise blanking.

My first impulse was to use a DSP chip—either on an evaluation board or by making a board with high-speed ADCs, DACs, digital up-converters and

[1]Notes appear on page 50.

153 S Gretna Green Way
Los Angeles, CA 90049-4015
kd6ozh@arrl.net

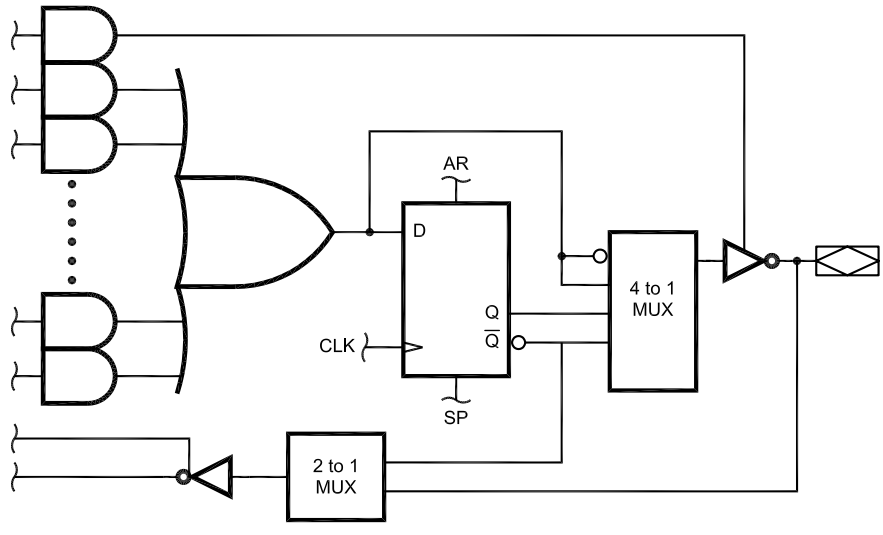


Fig 1—A SPLD macrocell (source: Lattice Semiconductor Corporation).

down-converters. The only inexpensive board uses the Analog Devices ADSP-2181 and that is going out of production. I looked at newer DSP chips, but evaluation board prices are in the $300 to $1000 range. The chips themselves are inexpensive in quantity but most are packaged only in ball-grid arrays. I also evaluated ASICs (application-specific integrated circuits) designed for the wireless infrastructure market such as digital up- and down-converters. Some are available in QFP packages with 0.65 or 0.8-mm-pitch leads, but they are expensive, their dynamic range is limited, and they are not optimal for the narrow-band modes amateurs tend to use.

After doing several paper designs and estimating costs, I took a different approach by using programmable logic devices (PLDs). PLDs are better suited to Amateur Radio applications than ASICs because they are more customizable. The facilities that are needed for narrow-band modes like AM, SSB, MFSK16, PSK31 and CW can be programmed into these parts. In addition, implementing only the functions that are needed for amateur applications minimizes the cost of a software-defined radio.

Four types of PLDs are available: simple programmable logic devices (SPLDs), complex programmable logic devices (CPLDs), field-programmable gate arrays (FPGAs) and combinations of MCUs and FPGAs called a system on a chip (SoC). Since many outside of the computer industry are unfamiliar with these parts I'll describe them here.

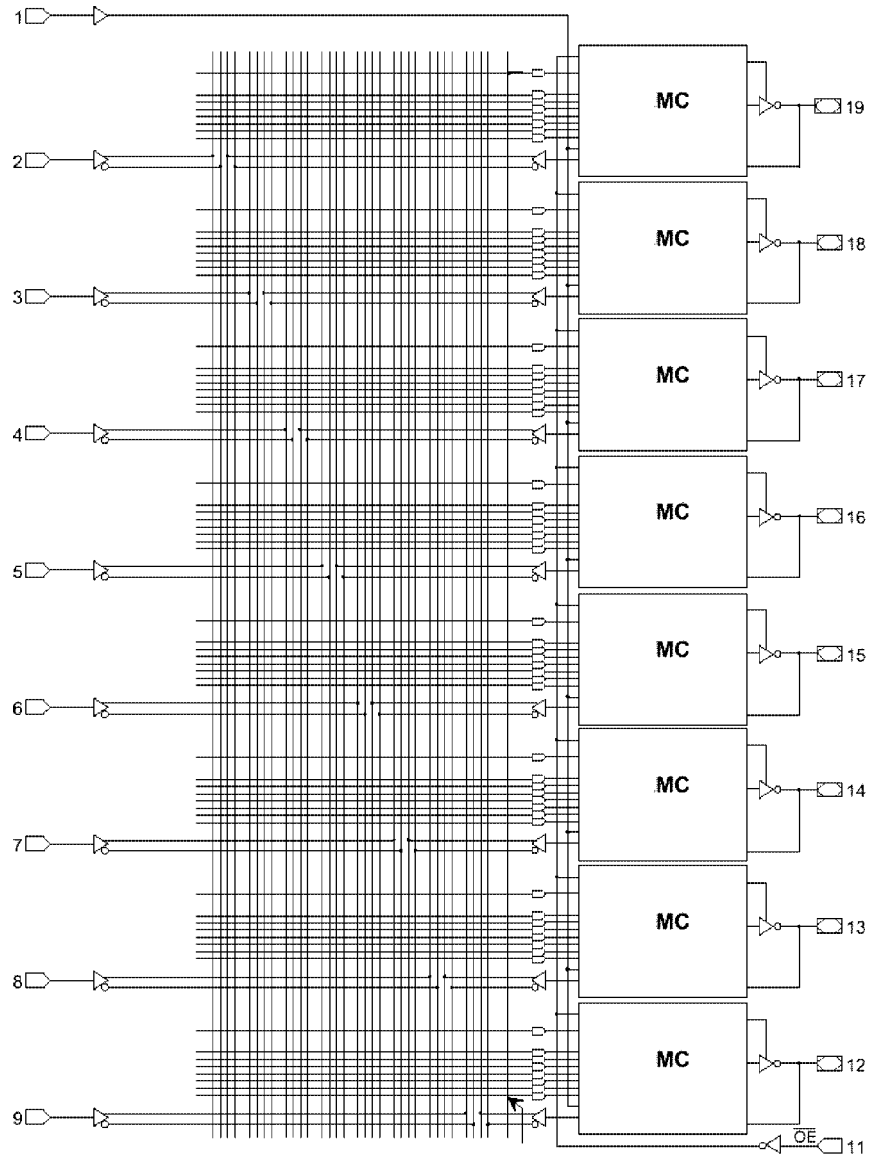


Fig 2—A 16V8 SPLD programmable AND-array (source: Lattice Semiconductor Corporation).

SPLDs

SPLDs have been available for 15 years. They consist of 8-10 D-type registers with programmable combinatorial logic ahead of the registers as shown in Fig 1. The registers may be bypassed to create pure combinatorial logic. The logic for each register consists of a number of **AND** gates that are connected to an **OR** gate that drives the register input. The multiplexers are configuration devices and are controlled by nonvolatile memory. The **AND**-gate inputs are also programmable and may be connected to any input pin, any register output bit or left unused (see Fig 2).

The original SPLD products used fusible links for programming, and they could be configured only once. Present devices contain EEPROM memories to hold the configuration data; they can be reconfigured 100-1000 times. Typical parts include the 16V8 (which has 8 registers and 16 inputs) and the 22V10 (which has 22 inputs and 10 registers).

SPLDs are mature devices and are being replaced by CPLDs in new designs. Many CPLDs now cost less than SPLDs and offer more functionality.

CPLDs

CPLDs build on the SPLD by putting arrays of SPLDs on a single chip. They provide more programmability in the form of programmable-interconnection arrays and specialized I/O control blocks. Fig 3 shows the general architecture of a CPLD. Notice that there are several clock, enable and clear pins, which have programmable polarity and may be routed to any *macrocell* via the programmable interconnect array (PIA). The PIA also accepts inputs from the macrocells. CPLDs may contain 32 to 512 macrocells in 2 to 32 logic-array blocks (LABs).

The macrocells in the logic-array blocks are the equivalent of SPLDs plus additional logic as shown in Fig 4. SPLDs typically have one dedicated clock input, while CPLDs provide multiple clock inputs and allow clocks to be derived from the programmable **AND**-array. Many manufacturers also provide programmable expansion of the **AND**-array where unused logic in one macrocell may be rerouted to another macrocell.

The I/O facilities contained in CPLDs are more flexible that SPLDs (see Fig 5). In a SPLD, each macrocell had a dedicated output pin. CPLDs have dedicated drivers next to the I/O

pins that may be connected to macrocells via a programmable switch matrix. Driver slew rates and logic levels are also programmable in many devices. Input logic levels also may be programmable and optionally registered.

CPLD Programming

CPLD manufacturers provide software for programming CPLDs. This consists of a compiler that takes a description of the desired logic and creates the configuration data and a downloader that loads the configuration into the CPLD (see Fig 6). The logic description may be entered by drawing schematic diagrams or by entering Boolean equations in a language such as *ABEL*. The download software usually works with a cable that attaches a few pins on the CPLD to the PC parallel port. Most manufacturers provide a free version of the compiler and download software that is suitable for amateur purposes.

Schematic design entry is easy to use and works well for small designs. A typical application for a CPLD is a phase-locked loop (PLL). For example, the phase-locked crystal oscillator design that I published in *QEX*[2] can be simplified by using a CPLD to replace both the microcontroller (MCU) and PLL chips. The reference and VCO counter modulus can be programmed directly into the CPLD. Fig 7 shows the design of the phase detector. Fig 8 shows the design of the VCO and reference frequency dividers. The design software includes TTL MSI equivalents to minimize the design effort. Finally, Fig 9 shows the top-level schematic with the preset counter modulus. The modulus can be set to any value and programmed into the CPLD EEPROM configuration memory.

The PLL design fits into a 32-macrocell CPLD that costs $1 in small quantities from suppliers such as Lattice Semiconductor (Mach 4 series) or Altera (MAX 3032A series). The original PLL plus MCU cost was almost $10.

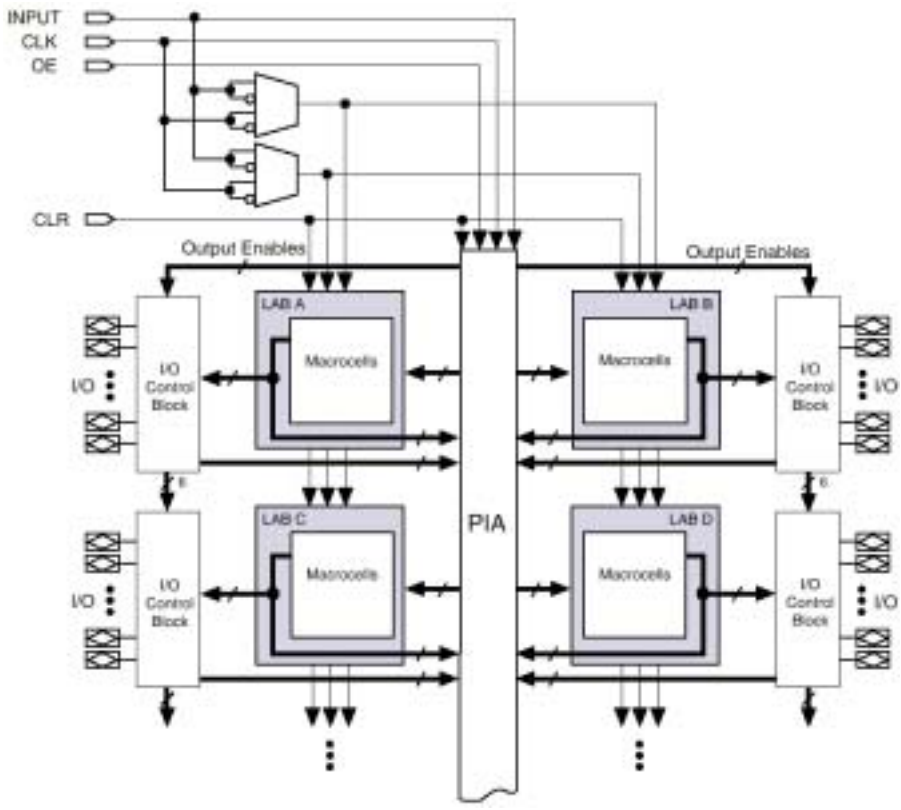


Fig 3—A typical CPLD block diagram (source: Altera Corporation). See Note 4.

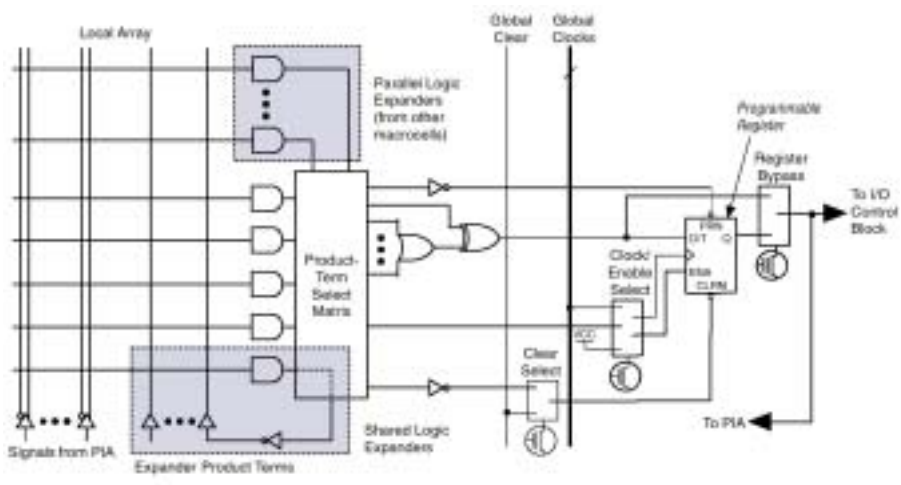


Fig 4—A typical CPLD macrocell logic block (source: Altera Corporation). See Note 4.

FPGAs

The field-programmable gate array (FPGA) provides even greater logic densities. The original products had a "fine-grained" architecture. They consisted of a sea of gates that could be interconnected via programmable switches and busses. Anything could be constructed from the gates, but gate utilization of 50% or less was common because of routing limitations.

Recently, FPGAs have begun to look more like huge CPLDs as the basic cells have become more complex. The cells of modern FPGAs typically contain one or more look-up tables (LUTs) to define arbitrary logic functions and one, two or four output registers. These can be interconnected to form adders, multipliers or other functions that are commonly used in digital signal processing (DSP). The Atmel AT40K series is a good example as it is particularly suitable for amateur use.

The FPGA consists of a square array of 256 to 2304 core logic cells arranged as groups of 16 cells as shown in Fig 10. Between the groups of core cells are RAM cells that may be interconnected with the logic cells. I/O cells are located near the bonding pads around the periphery of the die. They may be connected to logic and RAM cells via direct connection or busses that are dispersed throughout the FPGA.

The core cell consists of two three-input LUTs and a D register as shown in Fig 11. There are four inputs to the cell: W, X, Y and Z. These inputs may

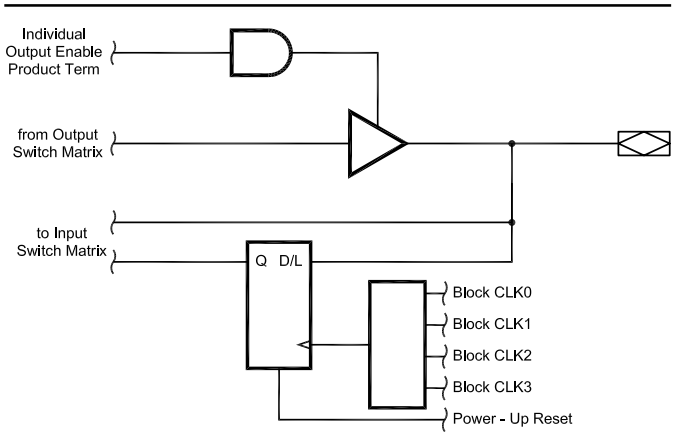


Fig 5—A typical CPLD I/O control block (source: Lattice Semiconductor Corporation).

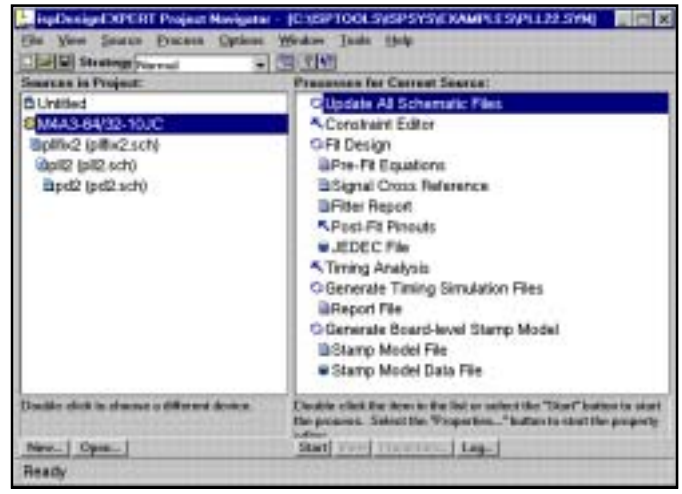


Fig 6—CPLD design software for a PC. See Note 4.

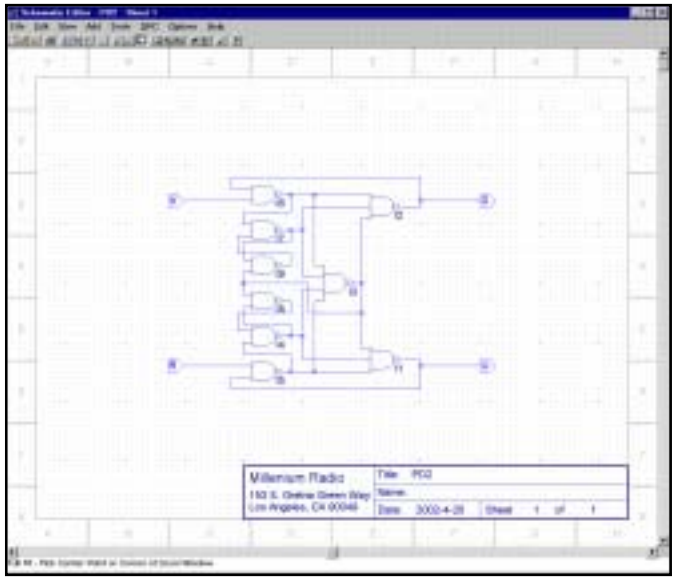


Fig 7—Phase-detector schematic input. See Note 4.

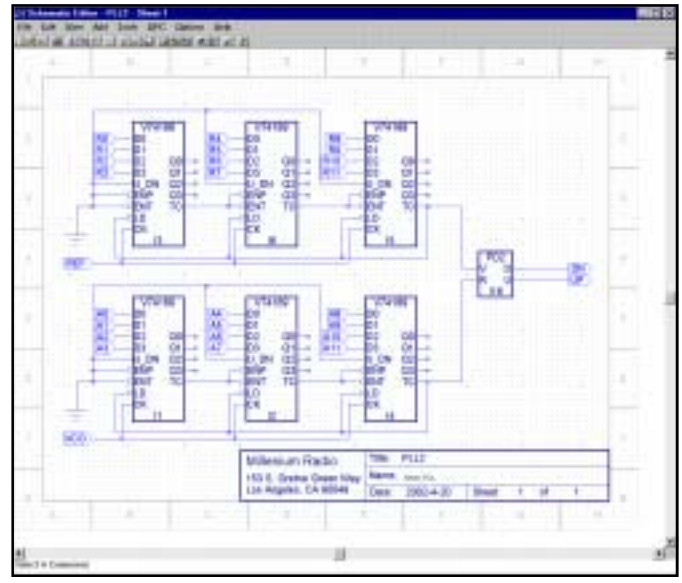


Fig 8—Reference and VCO divider schematic input. See Note 4.

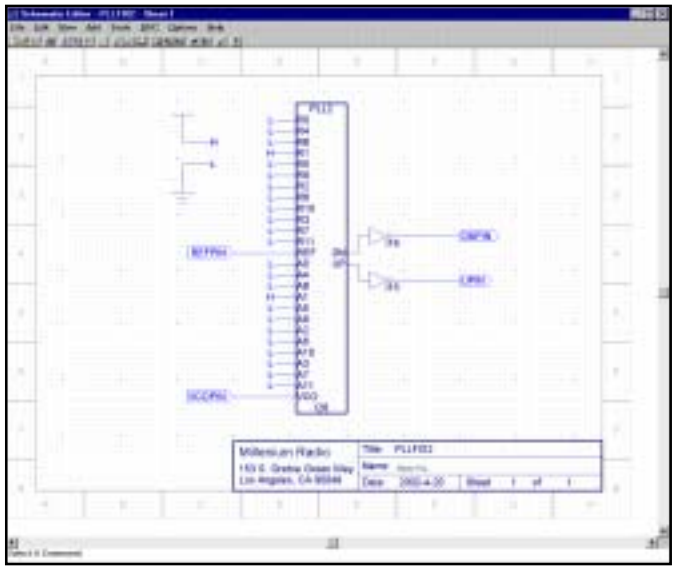


Fig 9—A top-level schematic for a custom PLL. See Note 4.

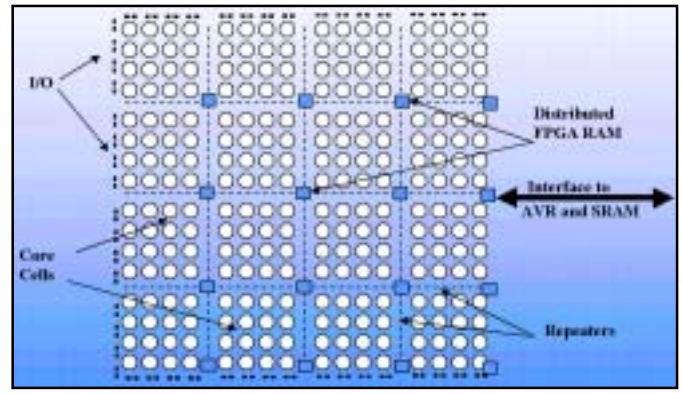


Fig 10—AT40K FPGA architecture (source: Atmel Corporation). See Note 4.

come from direct connections to adjacent cells as shown in Fig 12 or from busses that run throughout the FPGA as shown in Fig 13. The busses have programmable lengths. Bus segments may be isolated to a group of 16 core cells or interconnected via programmable repeaters to run through the entire array.

The core cell has three outputs that may come from the LUTs, the register or the tristate bus driver. The X and Y outputs may use the orthogonal or diagonal direct connections to adjacent cells shown in Fig 12. This is useful for fast-carry propagation and the construction of efficient parallel adders and multipliers. The tristate L output connects to one of ten bus lines adjacent to each cell as shown in Fig 13. The busses are useful for multiplexing data from multiple cells and connecting to input ports on multiple cells.

The core cells may be configured for various applications. A full adder is shown in Fig 14A. The adder may be combined with the **AND** gate in the cell to create parallel multipliers as shown in Fig 14B. The counter cell in Fig 14C shows how internal feedback may be used without requiring any external busses or connections. The cell may be used as plain combinatorial with three inputs and two outputs or four inputs and one output as shown in Fig 14D.

The FPGA also contains 2,048 to 18,432 bits of distributed RAM in 16 to 144 RAM cells as shown in Fig 15. Each RAM cell contains 32 4-bit-wide entries, and it may be configured as a single or dual-port RAM with synchronous or asynchronous I/O. In any configuration, the RAM has a 12-ns cycle time. Address and data ports are available via the busses shown in Fig 16.

RAM is desirable for many purposes including storage of data constants or microcode for FPGA processing elements. It can also be used to replace registers where access to individual bits is not required. Both applications allow the placement of more logic into each FPGA.

The I/O connection on the periphery of the FPGA die may be reached directly from adjacent cells or via the bus network as shown in Fig 17. Input and output pins may have dedicated registers or tie directly to cells or busses. Inputs and outputs can be programmed to operate at TTL or CMOS levels. The inputs may optionally have Schmidtt-triggers for noise-rejection and the outputs may optionally be tri-state to drive external busses. The drive current is programmable so slew rates may be limited in order to reduce EMI.

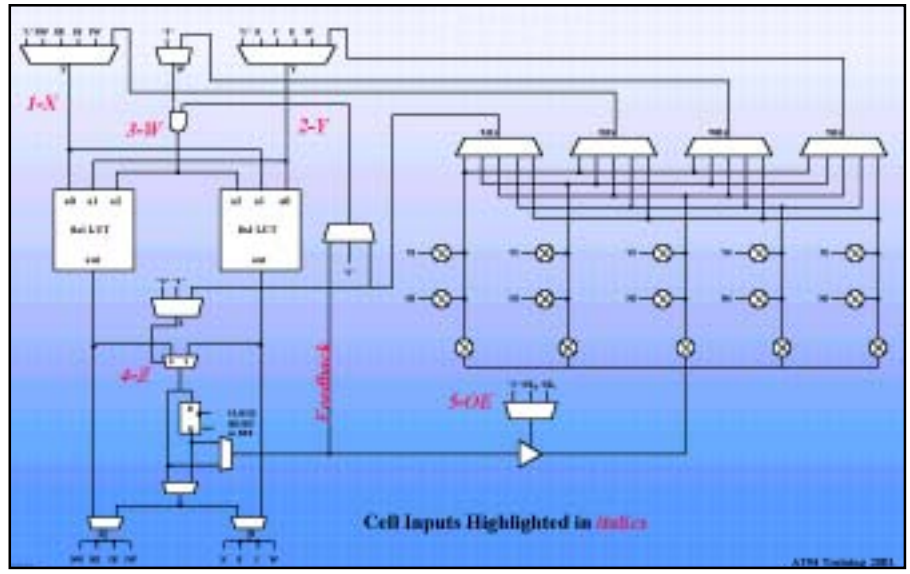


Fig 11—AT40K core cell (source: Atmel Corporation). (Beware! Antel uses a circle with cross to indicate *switches.* They are *not* mixers.) See Note 4.

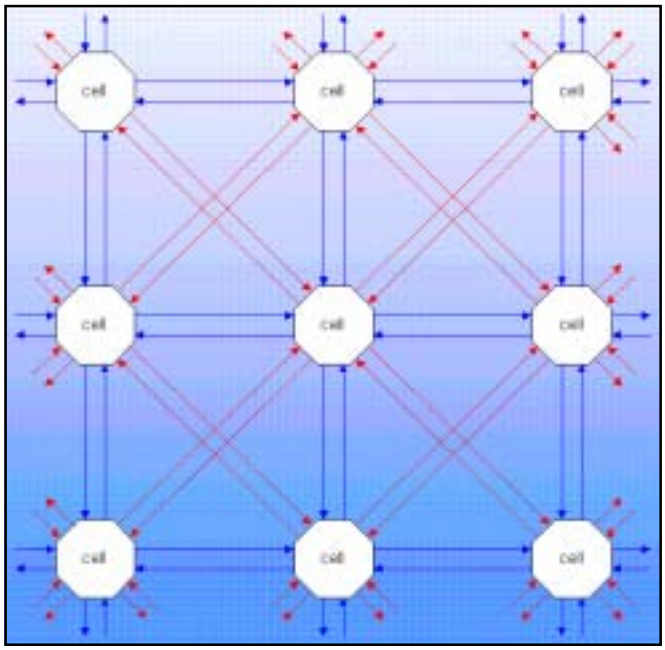


Fig 12—Direct cell-to-cell connections (source: Atmel Corporation). See Note 4.

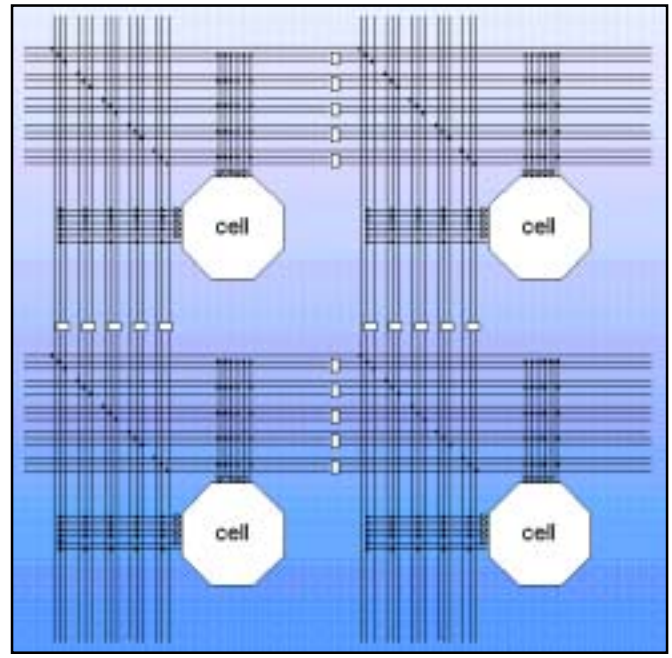


Fig 13—AT40K busses (source: Atmel Corporation). See Note 4.

FPGA Programming

The FPGA manufacturer provides software packages for entering design information and compiling it into configuration bits for downloading to the FPGA. Design entry can be done via schematics or *ABEL* as with CPLDs or by using higher-level languages such as *Verilog* or *VHDL*.[3] These languages provide a way to describe the behavior of the logic and to create "test benches" to simulate the input to the logic and verify the correct output. A simple *Verilog* language description of combinatorial logic is shown in Fig 18.

The design software synthesizes a gate-level design and then places and routes the design for the FPGA being used. The result is a configuration file that can be loaded into the FPGA via a PC parallel port. One major difference between FPGAs and CPLDs is that the configuration is loaded into RAM on the FPGA. This has the advantage that the configuration may be changed as many times as desired. Configurations can even be changed in real time so only the logic needed for the current operating mode need be resident in the FPGA.

Atmel also provides a design language called *Macro Generation Language* (*MGL*) that allows the specification of logic designs with very tight control over the placement and routing in the FPGA. These macros can be optimized for minimal cell usage and/or maximum speed. *MGL* allows the creation of fast reusable modules that can be referenced from *Verilog* or *VHDL* files.

MGL is similar to many structured programming languages. A macro definition consists of an interface block and

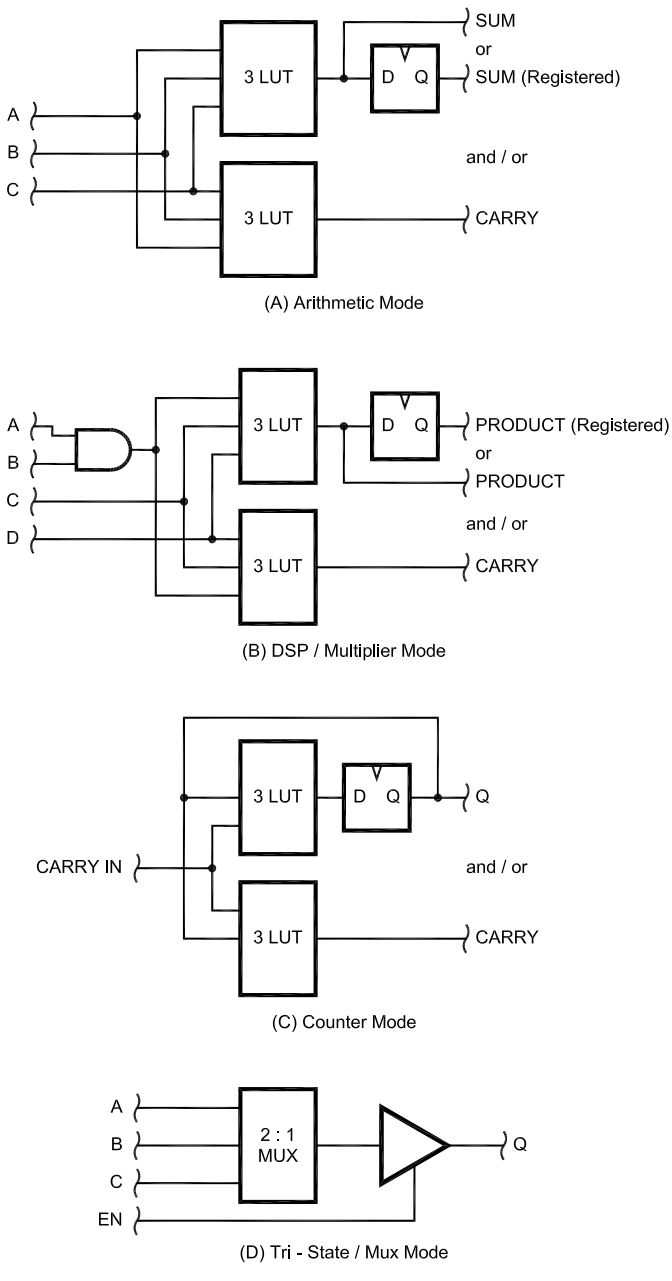


Fig 14—Core-cell configurations (source: Atmel Corporation).

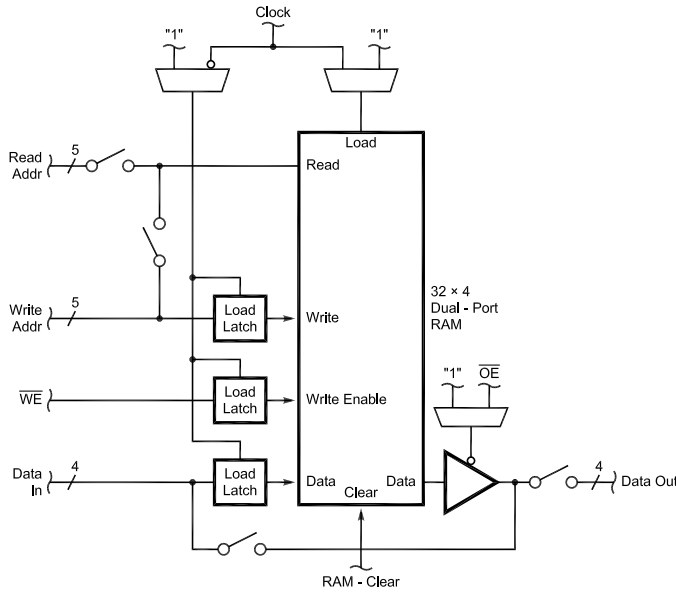


Fig 15—AT40K RAM cell (source: Atmel Corporation).

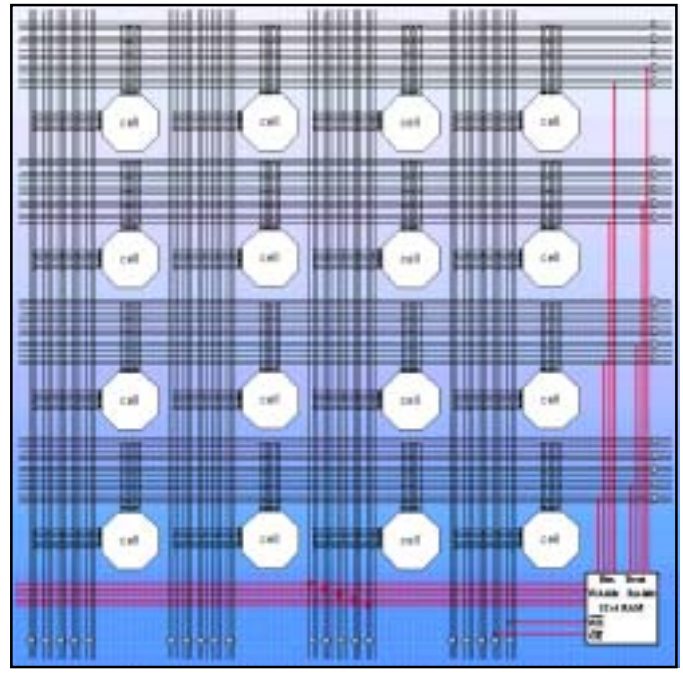


Fig 16—AT40K RAM cell bus usage (source: Atmel Corporation). See Note 4.

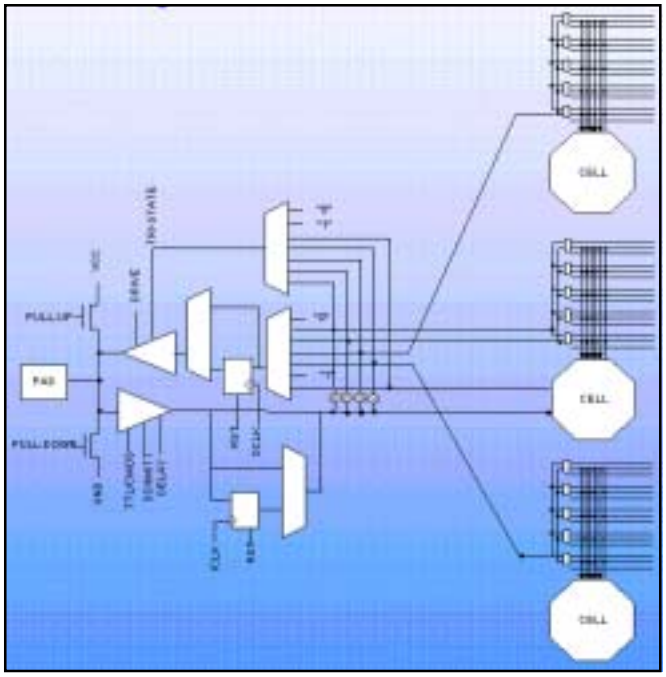
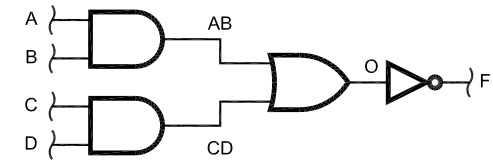


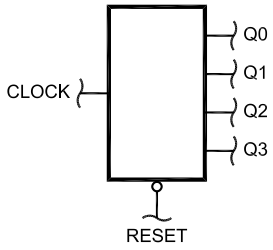
Fig 17—AT40K FPGA I/O (source: Atmel Corporation). See Note 4.




```
module AOI (A, B, C, D, F);
   input A, B, C, D;
   output F;
   wire F;
   wire AB, CD, O,

   assign AB = A & B;
   assign CD = C & D;
   assign O = AB | CD
   assign F = ~O;
endmodule
```

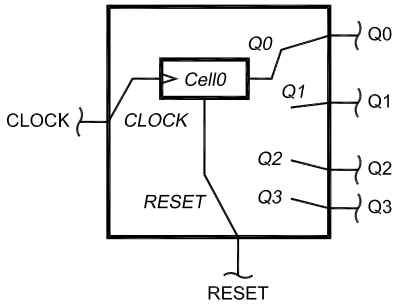
Fig 18—Schematic diagram and equivalent *Verilog* description (source: Doulos CBT).




```
counter : macro;
...

interface "Count" {width} of counter is
   inputports ( "CLOCK", "RESET" );
   for i in 0 to (width - 1) loop
      outputports ( "Q" {i} );
   end loop;
end interface;
```

Fig 19—Interface description in Atmel *MGL* (source: Atmel Corporation).

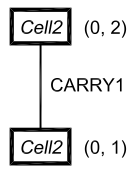



```
// Get macro from vendor library
aMacro   : macro := getmacro ( "FGEN1RF" );


instance "Cell0" of aMacro is
   location ( 0, 0 );
   functiong ( "!FB" );
   connections ( "CLK - > "CLOCK",
                 "RS" - > "RESET",
                 "G" - > "Q0" );
end instance;
```

Fig 20—Component "instantiation" in Atmel *MGL* (source: Atmel Corporation).

a contents block. The interface block defines the input and output ports for the macro. These are the signals that will be connected to external logic when the macro is used. Fig 19 shows one example, the interface to a four-bit counter. The snippet of *MGL* code defines **RESET** and **CLOCK** as the two inputs to the counter and Q0 through Q3




```
route of "CARRY1" is
   nodes ( (0, 1, "yOut" )
           (0, 2, yIn) );
end route;
```

Fig 21—Routing in Atmel *MGL* (source: Atmel Corporation).

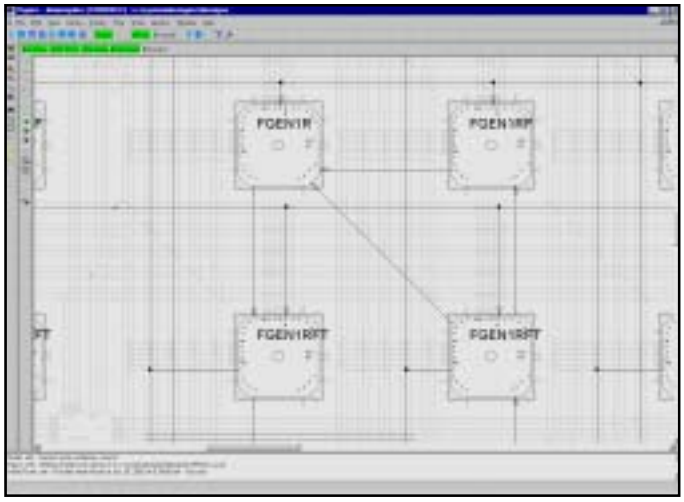


Fig 22—FPGA design software showing four cells and interconnection. See Note 4.

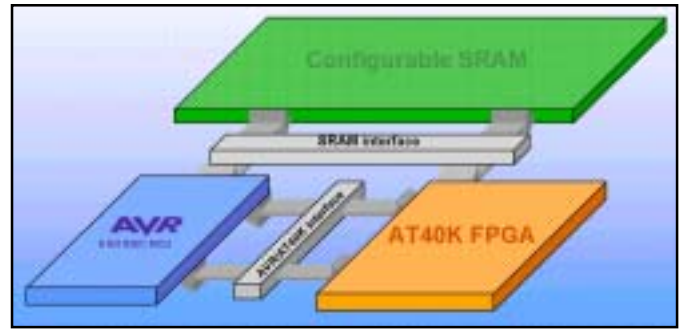


Fig 23—An Atmel FPSLIC (source: Atmel Corporation). See Note 4.

as the outputs from the counter.

The *MGL* contents block describes the underlying implementation of the macro. It *instantiates* components, connects the components together via nets and specifies the physical routing of these nets. Fig 20 shows the instantiation of a flip-flop and its connection to nets. First, the variable *aMacro* is assigned a value of FGEN1RF, which is part of the Atmel vendor library of dynamic macros. It defines a configuration of the AT40K core cell that has a LUT producing one output, which is stored in a register, and the stored value is fed back to the LUT.

The *location* statement creates an instance of the core cell and places it at the bottom left corner of the macro. This position is relative to the eventual placement of the macro. The *functiong* statement defines the contents of the LUT such that the output is the complement of the value fed back from the register. The *connections* statement then connects the ports on Cell0 to the ports of the macro.

Fig 21 shows how a direct connection between core cells is specified in *MGL*. The *route* statement contains a list of *nodes* that are to be interconnected. In this case, the Y output of Cell1 is connected to the Y input of Cell2. A more complex route, using a bus, would have a longer list of nodes and a specification of the type and location of the bus to use. The route can be specified to any degree of completeness as the routing can be completed using automated tools.

After the macro has been defined, debugged and executed, the generated macro can be imported into *Figaro*—the Atmel-provided tool for placement and routing on the FPGA. The process is similar to routing traces on PC boards. If the macro has been defined correctly, the cell placement will already be optimal. The automatic-routing software can be run to route any connections not fully defined in *MGL*. Fig 22 shows a close-up of four core cells after routing.

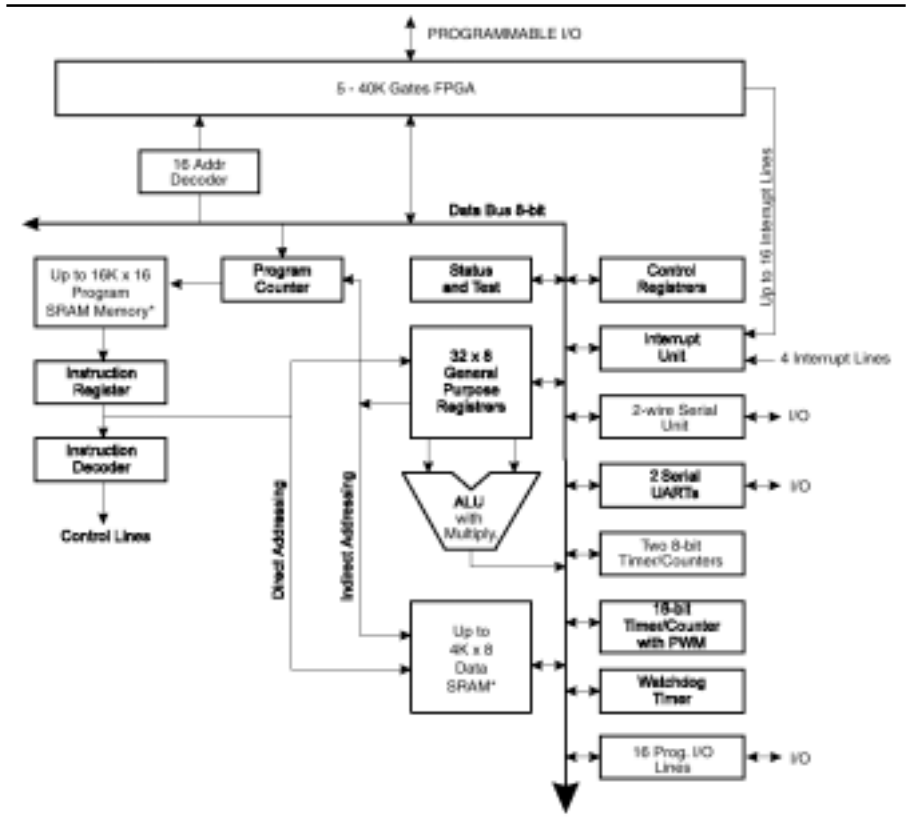


Fig 24—The AVR RISC MCU and peripherals (source: Atmel Corporation). See Note 4.

SoC—System on a Chip

A recent trend in the semiconductor industry has been the introduction of various "systems on a chip." These products provide a CPU, memory and programmable logic on one die to minimize the size of portable programmable devices. The CPU may be either "soft" (a gate-level design that can be downloaded onto the gate array) or "hard" (a custom-designed CPU sharing the die with an FPGA). The custom CPU uses less die area, but there are few companies with both gate array and microprocessor products.

A software-defined radio requires several processing functions that have traditionally resided in multiple chips. A microcontroller provided general housekeeping functions. A specialized

DSP chip provided filtering, modulation and demodulation. Multiple PLL and DDS chips provided frequency control. Multiple chips required long interconnections and tended to increase the level of spurious emissions.

A SoC with a CPU and FPGA can provide all major housekeeping, signal-processing and frequency-control functions. This simplifies the design and reduces cost without sacrificing any performance. The SoC that I have selected is the Atmel AT94K10AL. Atmel calls this a field-programmable system-level integrated circuit or "FPSLIC." It contains a 20- or 32-MIPS 8-bit RISC CPU, two serial ports, counter-timers, 36 kB of fast dual-port memory and a 576-cell FPGA that can be programmed from the MCU. Fig 23 shows the major components in the FPSLIC and Fig 24 is the MCU block diagram.

IC Packaging

One initial concern when selecting the SoC was the package size. The desire to produce small portable devices has driven package sizes down towards the size of the die. Technology has progressed from DIPs in the 1970s, to plastic J-leaded chip carriers(PLCC) in the 1980s, small outline packages (SOP) and quad flat packs (QFP) in the 1990s and now the ball-grid array (BGA) packages. A BGA package has all connections on the bottom of the package using a rectangular grid of solder bumps spaced as close as 0.8 mm. BGAs are mounted on a PC board that has solder paste silk-screened onto the mounting pads. The assembly is then exposed to a hot inert gas that melts the solder bumps and the solder paste to attach the component to the board. BGA packages are not suitable for home projects.

Luckily, manufacturers of components for industrial control and professional video/audio equipment do not have to produce tiny components for tiny products and continue to use SOP, QFP and LCC packages. Small CPLDs with 32-64 macrocells are available in a PLCC-44 package. Several FPGAs in sizes up to the 1200-cell range and the Atmel FPSLIC are available in PLCC-84 packages. The PLCC was originally designed to ease transition from through-hole to surface-mount PC-board technology. There are contacts on all four sides of the package spaced 50 mils (0.05 inches) apart. This package can either be soldered directly to the surface of a PC board or plugged into a socket. The sockets have leads on a 100-mil grid for compatibility with through-hole designs. This is ideal for construction of a prototype (or a one-of-a-kind unit) with point-to-point wiring as the sockets fit in pre-punched copper-clad boards.

Design Software

This type of design moves much of the traditional hardware prototyping work onto the PC with software-based simulation. I use the Atmel-provided FPSLIC-design software that integrates the FPGA and RISC CPU programming and verification tools (Fig 25) into one package.

The MCU programming is done in assembly language. The AVR CPU is

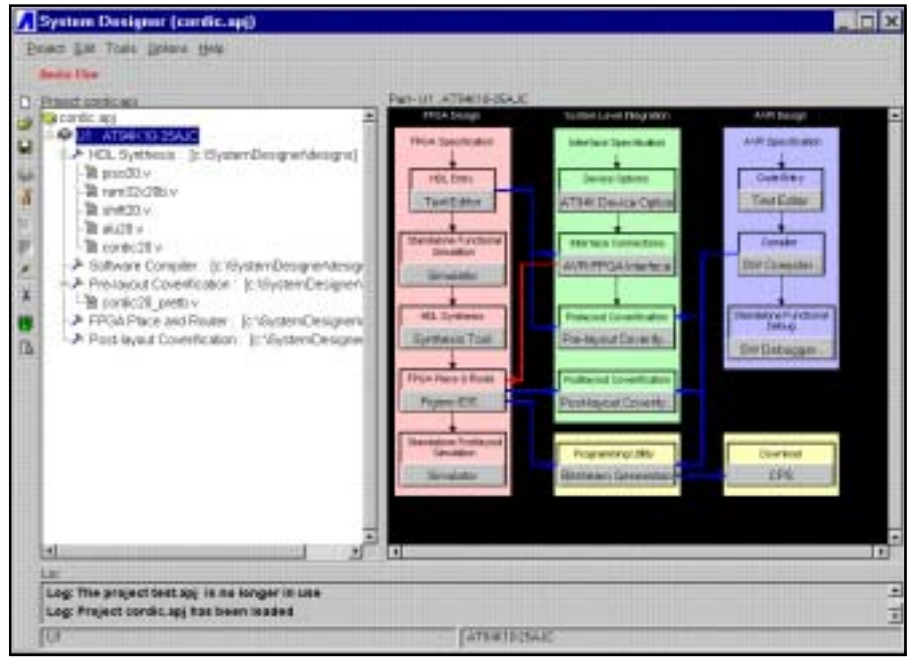


Fig 25—FPSLIC design software main screen. See Note 4.

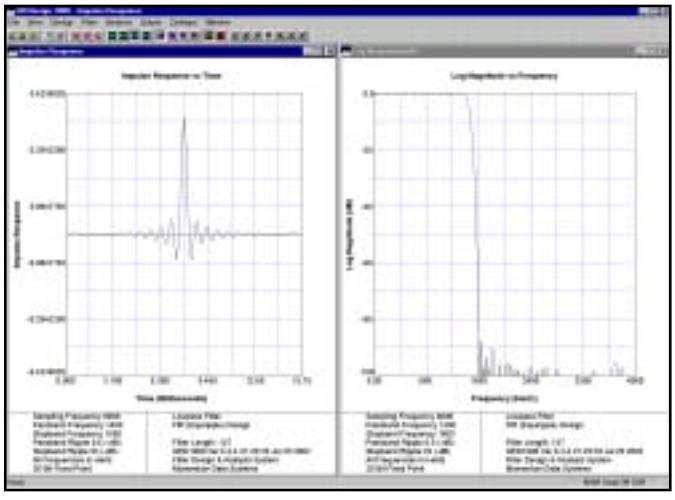


Fig 26—DSP filter design software. See Note 4.

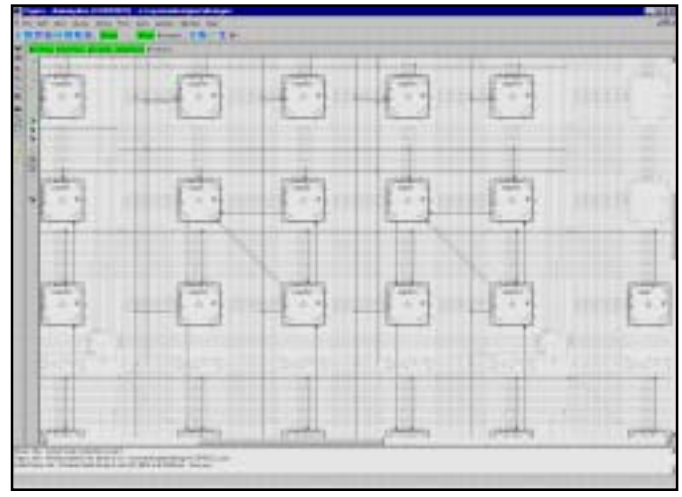


Fig 27—The upper right corner of CORDIC quad serial arithmetic unit. See Note 4.

a two-address general register machine that makes assembly-language programming easy compared to the old single-accumulator, single-address architecture used in 8051 MCUs.

The DSP filter-design software is from Momentum Data Systems (Fig 26). This software generates coefficients for either FIR or IIR filter designs optimized for a minimal number of taps for a given frequency response. There are several public-domain filter-design packages available on the Web that could be used in its place, and filter design could also be done with products like *MathCAD*.

FPGA Performance Results

The DSP version of my transceiver has the last IF at 13-19 kHz. This frequency is low enough to allow use of low-cost 24-bit audio converters with high dynamic ranges, and it is high enough to allow use of low-cost monolithic crystal filters as roofing filters. The frequency-conversion scheme is similar to that used in the Drake R8 but with ferrite filters replaced by DSP. The FPSLIC generates and processes digitized baseband in-phase and quadrature signals at a combined 16 ksps rate.

The approach used has been to design macros that implement high-speed DSP functions efficiently in the gate array hardware and do the rest of the processing in software. The following functions have been implemented as macros for the gate array:

- Dual 40-bit DDS phase accumulator
- Dual 19-bit CORDIC (coordinate rotation incremental computer) phase-angle-to-amplitude conversion unit
- 20-bit MAC filter coprocessor unit
- 24-bit serial ADC and DAC interface

These functions are tied together and interfaced to the MCU in *Verilog*. This allows hand-optimization where needed for speed and quicker programming for control circuitry that has less stringent requirements. Low-speed functions, such as AGC, are implemented in AVR assembly language. The two-cycle multiply instruction in the AVR CPU is ideal for implementing DSP functions.

Hardware implemented in a gate array has a different set of constraints than hardware in ASICs or TTL logic ICs on a PC board. The designer must always be aware of routing delays. In the AT40/94K series FPGAs, a direct orthogonal or diagonal connection from cell to cell has a delay of only 0.1 ns. A connection via a bus can incur delays of up to 11 ns depending on bus length. Very often, serial implementations of arithmetic functions will outperform parallel implementations. The MAC and CORDIC macros were the most difficult to implement, and they use a combination of serial and parallel logic to minimize size while retaining maximum speed.

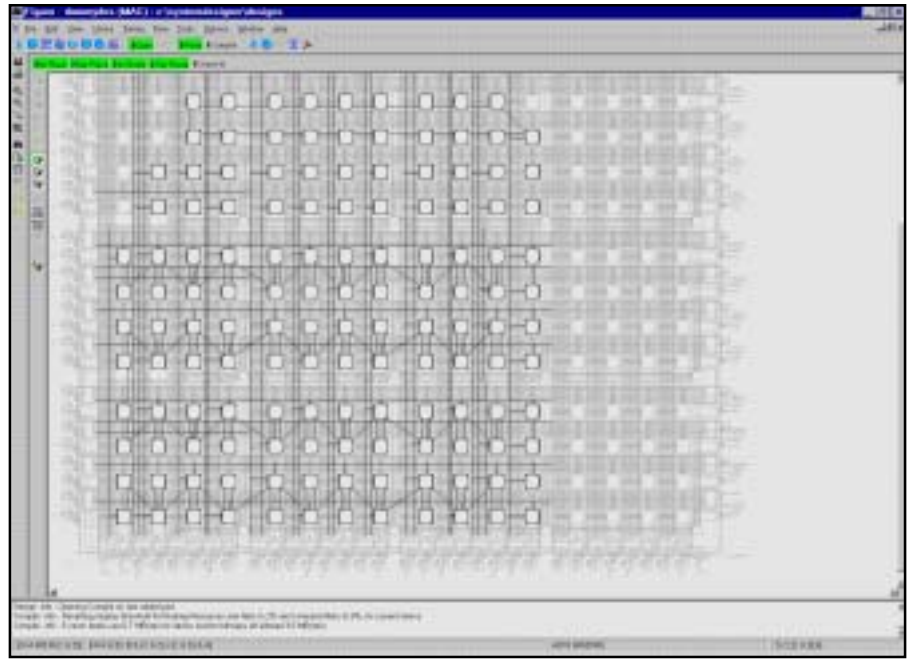


Fig 28—A serial-parallel multiplier-accumulator unit. See Note 4.

The CORDIC serial arithmetic unit requires only 98 FPGA core cells and is capable of 800,000 sine and cosine calculations per second. CORDIC is an algorithm that calculates sines and cosines using only shift and add operations. It is used to generate the frequency-reference signals for the transceiver and has spur levels below any current DDS ASIC.

Fig 27 shows the upper end of two serial arithmetic units that implement simultaneous bit-serial dual cross-connected shift and add operations to implement the core of the algorithm. The propagation delay in the most critical path has been reduced to 9.54 ns.

The MAC arithmetic unit requires only 145 FPGA core cells and is capable of 4.8 million 20-by-20-bit multiply-accumulate operations per second. Addition is done with 20 bits in parallel and multiplication is done by serial add and shift operations. The accumulator is 44 bits wide to accommodate all 40 bits of the product and prevent rounding errors. Four additional bits are provided to the left of the decimal point to prevent overflow when the transient value of the sum of products exceeds ±1.

Fig 28 shows the entire serial-parallel multiplier-accumulator unit. Orange cells (light squares) are used in the macro and gray cells are unused. The high packing density is achieved with serpentine routing that minimizes the length of several vertical and horizontal delay paths simultaneously. The maximum delay in any bit-serial data path is 9.17 ns. The 10.16-ns delay shown in the figure is for one multiplicand data bit, which changes state only once every 20 clock cycles.

Conclusions

The FPSLIC has proven viable for use in SDRs, and it provides a better solution for narrow-bandwidth modes than do ASICs designed for wireless applications. A follow-on article will describe the hardware that surrounds the FPSLIC to convert between the digital and analog domains and translate signals to and from the 16-kHz IF.

Notes

[1]J. Stephensen, KD6OZH, "The ATR-2000: A Homemade, High-Performance HF Transceiver," *QEX*, Pt 1, Mar 2000, pp 3-15; Pt 2, May 2000, pp 39-51; Pt 3, Mar 2001 pp 3-8; Letters to the Editor, May 2001, p 62.

[2]J. Stephensen, KD6OZH, "A Stable, Low-Noise Crystal Oscillator for Microwave and Millimeter-Wave Transverters," *QEX*, Nov 1999, pp 11-17.

[3]J. Wiseman, KE3QG, "Modern Digital Design for the Radio Amateur," *QEX*, Dec 1997, pp 3-12.

[4]Several of the figures in this article are captured from complex computer-screen images that do not reproduce well in print or in black and white. Interested readers can view these images full size and in color on their computers by downloading a package from the ARRL Web **www.arrl.org/qexfiles/**. Look for 9X02STEP.ZIP. □□