I Overview

Silicon Objects software development tools enable engineers to design, verify, program and debug their algorithms and protocols on FPOA devices. The design flow uses industry standard EDA tools and complements them with MathStar provided tools that are specific to the FPOA.

MathStar Silicon Object development flows are a revolutionary departure from conventional ASIC or FPGA development flows. Traditional design flows typically include any number of iterations on system models, RTL models, gate models, library maintenance tasks, synthesis runs, floor planning exercises, block connection, timing closure iterations, inter-block routes, global timing closure, clock tree design, and finally DRC/LVS rule checking. While FPGA designs do not have some of the complexities inherent in ASIC design, in many instances, reaching timing closure for ASICS and high-end FPGA devices can both run into many weeks of effort as performance and design density is pushed.

In addition to the time required to reach chip tape-out for .13 micron based ASIC design, the tools cost can be several multiples of the chip fabrication costs.

Any design engineer familiar with contemporary chip design methods will readily understand the FPOA tool flow. Designs are entered and behaviorally simulated using an IDE, such as Summit Visual Elite. Then they are compiled into an intermediate Assembler representation. This representation is then mapped into the hardware resources of the Silicon Objects device. The object code is loaded onto the array from a PROM or through the JTAG interface very similar to the methods used by FPGAs today. However, unlike both ASIC and FPGA designs, Silicon Object designs have a completely deterministic timing structure. As such, they are timed only on cycle boundaries of the internal clock with no regard for gate-level timing. This greatly simplifies the process, improving productivity and development predictability.

The Field Programmable Object Array (FPOA): Hardware Resources Backgrounder

The FPOA is a heterogeneous medium-grained array composed of hundreds of individual processing elements. Each element is called a Silicon Object.

Within the array the data path and control path are loosely coupled, yet independently configured. The data path is 16 bits wide while the control path is bit-wise granular.

Each Silicon Object has its own program and data memories. Further, each object operates on its own, without the aid of global control. The FPOA could be characterized as a Multiple Instruction Multiple Data (MIMD) machine.

Communication between Silicon Objects is through either the eight nearest neighbor connections or through the long reach connections (Party Lines). Objects are allowed to change communication patterns on a per-clock basis.

Each Silicon Object has a loadable configuration map that contains both operation and communication directions. The device can be reconfigured “in system” by loading a new configuration map created by the MathStar tools.

The control path guides program execution while data is moved and operated upon via the 16 bit data path. From this view, instructions are the mechanisms that tie the independent control and data paths together within the array.

There are presently six data path Silicon Object types used to create FPOAs: the Arithmetic Logic Unit (ALU) Silicon Object, the Content Addressable Memory (CAM) Silicon Object, the Cyclic Redundancy Check (CRC) Silicon Object, the Integer/Real Multiplier Accumulate (MAC) Silicon Object, the Register File (RF) Silicon Object, and the Truth

Function (TF) Silicon Object. There are several more planned for later versions of the product.

In addition, RAM memory resources and configurable I/O wrap the Silicon Object core to complete a specific device definition. The type and ratio of the Silicon Objects, memory resources and I/O were chosen based on detailed study of the attributes of communication processing algorithms and are defined to target specific applications spaces.

I/O Subsystems have been tailored to the different applications spaces as well.

II Design Flow

There are three major layers to the Silicon Object Design Flow. It is possible to enter the tool flow through a third party IDE (such as Summit VE) or through MathStar's assembly level programming language known as Object HDL (OHDL). These two flows overlap, in that the IDE based design flow is a superset of the Assembly code based design flow. There are unique benefits to both of these design flows and they can be used interchangeably as required by the designer. As an analogy, there are often situations in conventional processor code development where assembly level libraries or code sequences are used to meet very tight timing, code size, or performance requirements within the context of higher level language code development. MathStar software provides the designer with the ability to work both at the abstract level, and dive down to the hardware details in assembly level as required in an integrated development environment. See Figure 1.

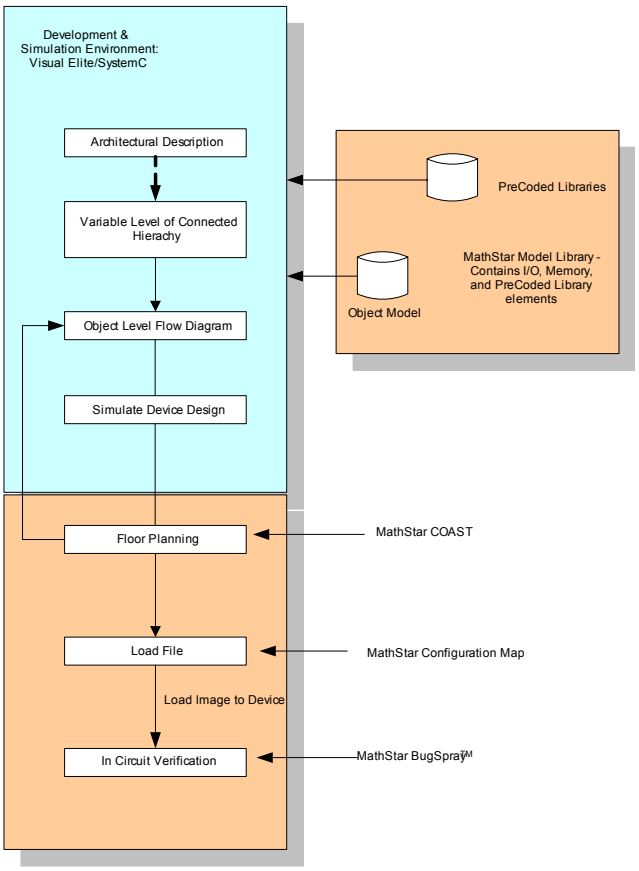

Figure 1.MathStar Design Flow

- High-level Description Language. This language could be SystemC, Streams-C, or Handel-C as long as it follows such language syntax constructs. These languages have knowledge of timing and provide a complete design and simulation capability inside of an IDE (Visual Elite from Summit). Due to the granularity of Silicon Objects, a higher level of abstraction is allowed than is usual within a traditional synthesizable HDL. The SystemC is parsed and converted to MathStar OHDL for physical placement onto the array.
- Silicon Object Assembly Language. (OHDL). This is MathStar's programming language. It is the representation that MathStar translates other input data structures into. It is also a stand alone development language that can be used to create a design. If performance, timing, or other

requirements dictate, the designer can chose to implement behavior directly in OHDL. Using this level of abstraction permits the designer complete control over resource allocations and object utilization.

- COAST (**CO**nnection & **AS**signment **T**ool) provides a means for the Silicon Object developer to visualize and manipulate Silicon Object grid location assignment and Silicon Object interconnections. COAST assists the user in the process of converting algorithms expressed in OHDL code into a grid of programmed and configured Silicon Objects, complete with all location and interconnection information. Once valid assignments and interconnections have been realized for all Silicon Objects, COAST will allow the developer to generate a binary load image file.

This file (the configuration map) can then be loaded into the Silicon Object device.

III Designing

Programming and timing

A key notion in programming a Silicon Object array is the relationship of timing and signals. In a conventional FPGA or ASIC, the maximum operating frequency of the device is determined by the longest or “critical” signal propagation path. Typically, significant effort and resources are expended in reaching “timing closure” against an operating frequency specification. It is arguable that for devices using .13 micron geometries and below, accommodating timing closure issues accounts for a very substantial portion of the overall design effort, often measured in months. MathStar's Silicon Object devices have completely deterministic performance, based on the clock cycle. In the FPOA the timing of signals is quantized and resolved into system clock periods. Programmers need only to confirm that relative clock period timing of signals is properly accounted for and scheduled appropriately, regardless of the path the signal takes from one object to another.

The MathStar assembler and COAST editor enforce timing requirements through the use of timing properties attached to signals declared by the user. Signal timing is primarily determined by object assignment, and to a lesser extent the Manhattan distances taken by party line connection. The eight nearest neighbors to any object are guaranteed to have zero clocks of latency. Party line route lengths can take from one or more clock cycles, depending on object distance requirements.

A key feature of the objects is the ability to select communications paths dynamically, depending on any of several state values in any given clock cycle. In this way, communication paths are only reserved temporally in the cycle in which they carry valid information. They are then free in the next cycle to carry information completely unrelated to the current cycle.

Process

Programming the device involves four key tasks to successfully translate an abstract behavioral description into an executable load image for a FPOA; instruction coding, simulation, signal connection, configuration map generation.

Instruction coding

Instruction coding is accomplished either in the SystemC environment (Summit VE for example) or the OHDL environment.

For each environment, MathStar provides a cycle accurate simulation model. These models fully reflect the deterministic behavior of each object and are used to implement and simulate the design.

Within the VE environment, for example, an object array is created that reflects the actual FPOA that is being used to implement the design. (Figure 2).

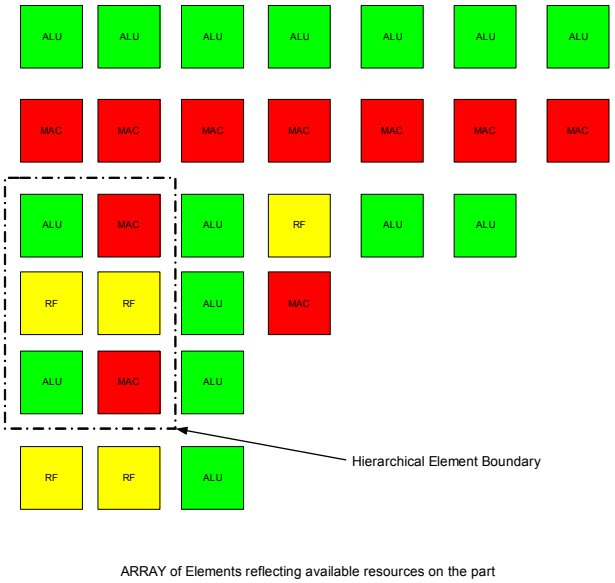

Figure 2. MathStar FPOA Template in SystemC

Within VE, the user then opens the Silicon Object model and provides the coding required by that object for the design. The coding is in SystemC, but is constrained to reflect the hardware limits of the actual objects. Each object will have it's own set of code that can compile into the simulator.

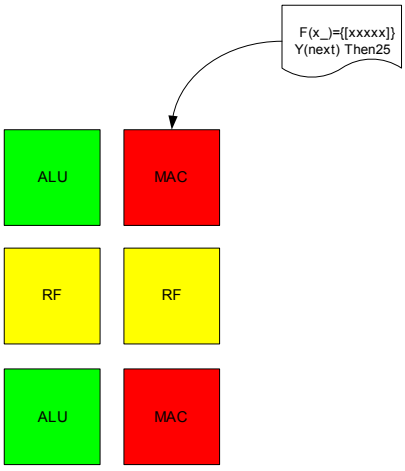

Figure 3.Coding for the Object

Designs are checked by the OHDL generator, to prevent the user from exceeding the resource limits of a given object.

Control flow synchronization is accommodated through any of several methods including out-of-band data "valid" bits, mailbox handshaking through control signals or registers, and explicit cycle timing synchronization. Any or all of these may be implemented in the behavioral code.

After coding is complete, the device connections are made. In VE, the connections are made using the MathStar Channels. In OHDL, the connections are made using named ports.

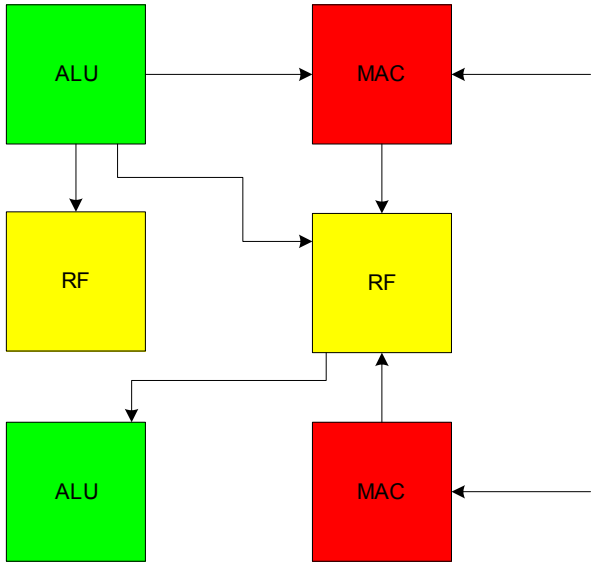

Figure 4.Connecting the Objects using Channels

After instructions and connections are mapped into the objects comprising a "logical" array, the device behavior is simulated using the SystemC simulation environment. The design can be iterated on quickly in this environment and the user quickly determines whether the behavior is correctly described, for both functional correctness and timing or performance.

Object Assignment

After a design is successfully coded and simulated, it must be assigned to a physical object array. Conceptually, assignment involves allocating a set of coordinates to each object so that the configuration map can be created and a load file generated. In practice, assignment is the process of allocating a segment of code to an object and then adjusting the interfaces between the objects to meet timing and connectivity requirements with available interconnect resources.

It is important to note that a silicon object array is a fixed array of resources in which the conventional ASIC/FPGA notions of "placement" and "timing" as a function of gate location, gate delay and wire delay do not apply. The communications paths in a FPOA are predetermined and quantized by the internal clock period of the device. Thus, communications from one object to another take place over paths that are timed by clock periods (i.e. "hops"). Users "time" a behavior based on the latency and throughput requirements of the algorithm as expressed in clock period delays being mapped.

Assignment typically is initiated around the physical locations of infrequent fixed function objects in the array, such as CAM objects. ALU objects are then placed adjacent to these objects under the influence of the connectivity and timing relationships established for the signals entering and exiting the objects.

The COAST toolset provided by MathStar is used to perform the assignment and signal connection tasks to convert floating object designs into fixed loadable images. In Figure 5, a simple three-object design has been placed and the nearest neighbor paths have been highlighted for the objects that communicate. The designer can pick up an object and move it to another more desirable location and the COAST tool will automatically elect the appropriate communications paths and maintain signal connectivity.

As described earlier, object assignment is the most significant factor in realizing the timing relationship between signals. If objects are placed adjacent, then nearest neighbor interconnect can be utilized guaranteeing single-cycle communications between two objects. In this way, the relationship between instruction mapping and object assignment completely defines the overall chip-level signal timing. Careful attention to mapping and assignment will yield an extremely high-performance, functionally dense, design using a minimum of object resources.

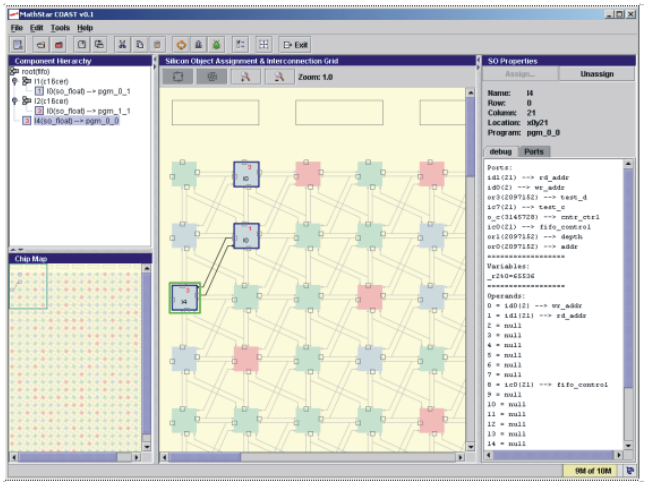

Figure 5.COAST Connections

Signal Connection

A signal is a path from one object to another, along a fixed grid of predetermined potential paths. The actual path chosen then determines the number of clock periods required to propagate the signal from the source object type to the destination object.

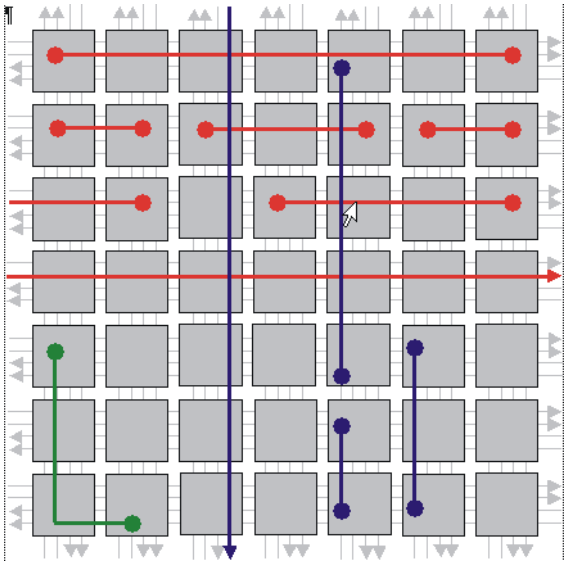

Figure 6. Nearest Neighbor and Party Lines

In the context of programming a Silicon Object, signal connection occurs over two potential resources: nearest neighbor interconnect and party lines. In practice, connection of nearest neighbors is a trivial exercise since assignment determines adjacency and all adjacent objects have direct communications. In situations where it is not physically possible to adjacently locate all objects that communicate with each other, party line connection resources can be utilized to provide the necessary connectivity. As shown in Figure 6, party lines connection are very flexible in that any object in the array can reach any other resource in the array. Party lines can be segmented at configuration to accommodate several objects along the line. For the programmer, the party lines and the nearest neighbor interconnects are almost interchangeable. The tools determine the communication channel depending on the timing constraints attached to a particular signal, or the type of objects to which the signal is attached.

Loading the Array

Subsequent to completion of mapping, assignment, and connection, the compiler or assembler generates an encrypted load image file to be burned into the program PROM or loaded directly into the device via the JTAG interface. At reset, the FPOA loads the program file from the PROM, which configures the array to behave in the manner expressed in the program.

IV Conclusion

The tools, programming model, and programming process for a MathStar FPOA are both revolutionary and easy to comprehend. SystemC or OHDL programs are created as instruction sequences that are mapped around the array. Communication paths are chosen based on system clock cycle timing requirements, not on gate level propagation. In this way, significant time and cost savings over conventional ASIC and FPGA development processes can be realized. There are no synthesizers, timing closure issues, or extensive million-gate route bottlenecks to hinder the fielding of extremely high-performance, functionally dense designs.