A Configurable System-on-Chip Device Facilitates Customization and Reuse

Danesh Tavana
Steve Knapp

Triscend Corporation

2000 System-on-Chip Design

ABSTRACT

Time to market pressures, increasing system complexity, and smaller process geometries, are creating a market vacuum that will be increasingly addressed by an important emerging category of devices: the Configurable System-on-Chip (CsoC). These application specific programmable devices (ASPP) are single chip combinations of microprocessors, memory, dedicated peripheral functions, and embedded programmable logic. They provide unprecedented time-to-market benefits and field customization for the electronic systems of this upcoming decade.

Integration of microprocessors, memory, peripherals, and programmable logic is made possible with a new bus architecture called the Configurable System Interconnect Bus (CSI) developed at Triscend Corporation. The Configurable System Interconnect Bus was specifically designed to facilitate re-use, guarantee timing, increase system throughput, and reduce system debug time in applications that require intense time-to-market and field upgrade.

AUTHORS/SPEAKERS

Danesh Tavana
VP Engineering, Triscend Corporation

Danesh Tavana has 19 years of experience in system-level and semiconductor design. Prior to co-founding Triscend on May 1997, he spent 5 years at Xilinx on various FPGA design and soft-core development projects. Before Xilinx he worked for NeXT computer, Inc. and Adaptec on various CMOS ASIC projects. Danesh began his career at Monolithic Memories in 1981 as a Bipolar IC Design Engineer. He holds eight US Patents and received his BSEE from U.C. Berkeley, and MSEE from Santa Clara University.

Steven Knapp
Vice President of Applications, Triscend Corporation

Steve has been actively involved in programmable logic applications for more than 14 years. Prior to joining Triscend in 1997, Steven had founded OptiMagic, a company specializing in developing software and intellectual property for programmable logic. Before that, he spent 11 years at Xilinx, where he held various management and sales positions. Steven holds several patents in parameterized logic design entry and has a BSc in materials science from MIT.

Introduction

Six competing requirements challenge the embedded systems designer: time-to-market, performance, cost, physical size, power consumption, and product features (Figure 1). The designer's task is to find the best possible compromise between these requirements in order to deliver the most effective product to the customer.

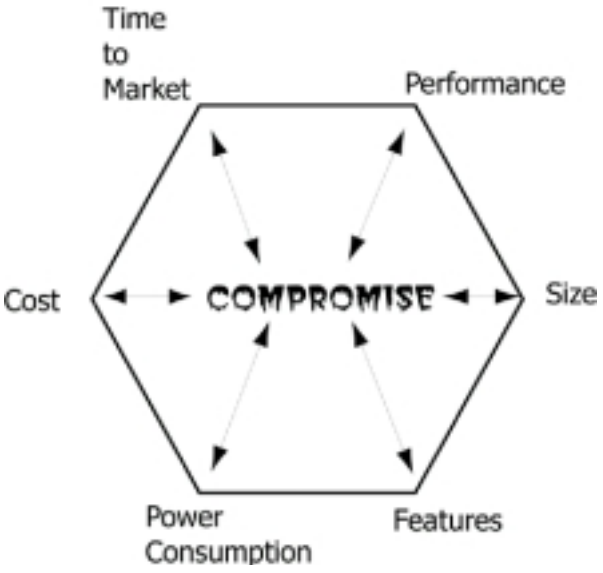


Figure 1. Embedded Systems Challenge

For many applications, Triscend's Configurable System on Chip (CSoC) products offer designers a more attractive balance between these factors than alternative solutions. Design re-use, and field upgrade through remote software download, when combined with a CSoC device, fundamentally change the rules of embedded system design by breaking the link between time to market and product features.

CSoC Introduction

Embedded system designers develop products with a specific function or application in mind. In almost every application there is a central processing unit that is responsible for the overall system's supervision, handling of complex state machines, and number crunching algorithms. The central processor also manages the system's memory and I/O, which are the peripherals that interface to various entry or display devices. In contrast to personal computers, the embedded system applications are usually differentiated in both hardware and software. Unlike the PC chip sets and prevailing standards around it, the embedded system world is plagued with innumerous design choices and lack of standards. Triscend's CSoC device is an ideal single chip device for the embedded system designer who wants both hardware and software flexibility on a single extensible platform.

Imagine if as a system designer you could begin your design with a popular industry-standard processor that is supported by leading 3rd party compiler, assembler, and debugger tool vendors. The processor is a familiar, proven architecture with a large availability of freeware applications. The processor's performance is boosted through pipelining in a "Turbo" mode operation, and by employing advanced CMOS processing technologies.

What if you also had an integrated DMA controller that offloads the processor from large data transfers, freeing it for more important tasks? A glue-less interface to external byte-wide Flash, EPROM, or SRAM improves performance and eliminates external latches that are typically required in time-multiplexed address/data bus interfaces. In applications requiring even higher performance with lower power dissipation, an on-chip SRAM ranging in density from 8K bytes to 64K bytes provides data or code storage for the processor, and data buffer space for the DMA controller.

What if the system came with an integrated on-chip bus specially designed to insure single cycle data transfers and programmable address decoding for the peripherals embedded in the chip? This bus is specially designed to facilitate "soft module" re-use by abstracting the processor specific signaling requirements from the peripherals, and by supporting a friendly drag and drop software tool for developing micro-controller derivatives. Programmable I/O (PIO) pins operate independently from the bus and are tightly coupled to an on-chip embedded programmable logic core. Flexible pin assignment for optimum PCB layout and a host of field programmable options like output drive strength, slew rate control, pull up, pull down, registered I/O, or low power operation are some of the user selectable features for each package pin.

Finally, what if all this was offered with a large amount of on-chip programmable logic with sophisticated built-in system debugging hardware that includes a JTAG interface and a breakpoint unit? And what if the device came with a rich debugging environment where the device can be altered in the field infinitely and debugged using standard logic debugging or processor ICE debugging techniques? Now you can have a device that does exactly that. Triscend Corporation's E5 device (Figure 2) is a single chip CSoC that is the ideal embedded system platform for creating flexible, fast time-to-market applications.

Implementing the E5 as a dedicated logic chip with only a small area of re-configurability provides one to two orders of magnitude more efficiency in silicon area than a similar implementation in a pure programmable logic device. Other obvious advantages offered by the integration include increased performance and lower power. An important architectural feature of the E5 CSoC

is the Configurable System Interconnect (CSI) bus that allows each resource on the bus to communicate to other resources, thus leveraging and building upon the existing resources.

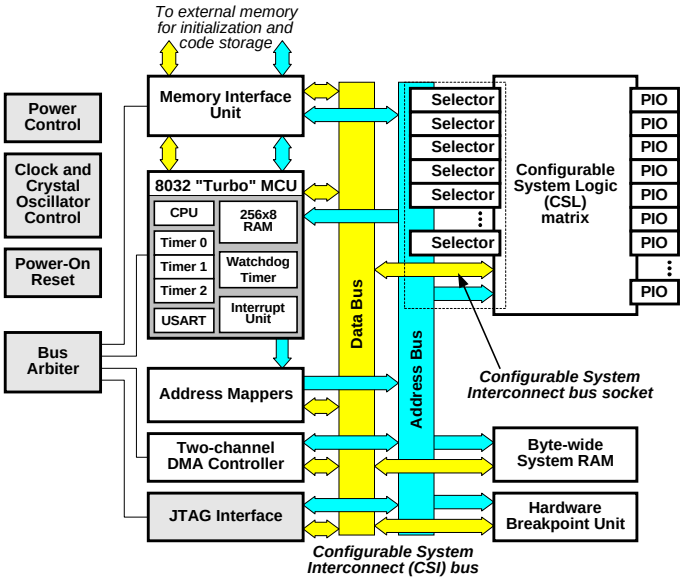


Figure 2. Triscend's E5 Configurable System on Chip Block Diagram

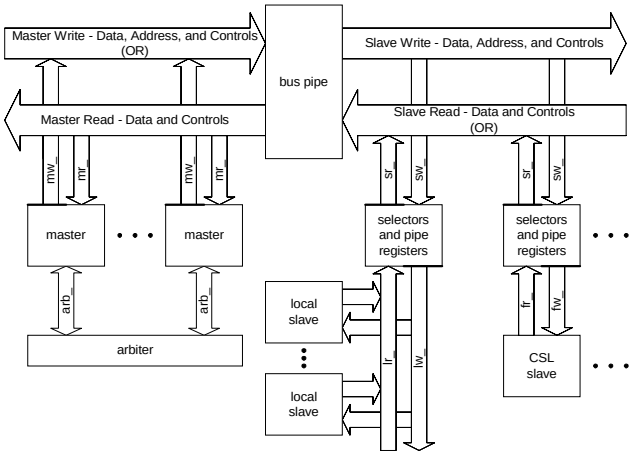
CSI Bus Introduction

The configurable system interconnect (CSI) bus is designed to facilitate design re-use within the configurable system logic (CSL). The bus is distributed throughout the CSL in fixed logic. It is connected and operates with all system masters including the processor, DMA controller, JTAG interface, and the external Memory Interface Unit (MIU). Memory mapped and DMA slaves may be implemented to handle data transactions. User logic may obtain the services of the processor and DMA controller as proxy masters through an interrupt or DMA request respectively. The user is assured that the bus will reach any logic implemented within the CSL. The maximum bus performance specification can be met regardless of the placement algorithm's quality of results.

The primary objective in the design of this bus was to make it easy to use. The bus operates synchronously. The default transaction duration is one clock cycle. Wait states may be added when necessary. The appropriate bus signals may be configured to connect to the CSL logic as required for the user's design. Additionally, the synchronous decode of addresses and commands is handled for the user by selectors. There are many selectors distributed throughout the CSL. Each selector provides simple read and write signals to be routed to the user logic. The addresses and commands that a selector responds to may be individually configured.

CSI Bus Architecture

The bus operates synchronously and supports multiple masters and slaves, DMA, and wait states. The bus is pipelined, has separate write and read data paths, uses multiplexed or logical OR networks to combine signal sources, and supports a fast default cycle with optional wait states when necessary. A round-robin arbitration scheme is implemented for masters. The slave side includes multiple decoded and qualified read and write enable signals generated by selectors. A logical bus architectural diagram and signal flow is shown in Figure 3. There are four primary bus segments: master read/write, and slave read/write related to the distribution and collection of the bus signals prior to the pipeline registers. The multi-source instances of all signals in each collection segment are combined via logical OR gates or multiplexed networks into a consolidated bus. This prevents power consumption concerns in the event of any contention that is typical of tri-state bus structures.



arb_	Arbitration request and early response
mw_	Master write collection segment
mr_	Master read distribution segment
sw_	Slave write distribution segment
sr_	Slave read collection segment
lw_	Non-CSL slave interface distribution
lr_	Non-CSL slave interface collection
fw_	CSL interface distribution segment
fr_	CSL interface collection segment

Figure 3. CSI Bus architecture and Bus signal description

The data bus signals within the CSL travel in one of two directions, either from the system masters to the

slave logic on a write operation (labeled as fw_ in Figure 3), or from CSL slave logic to system masters on a read operation (labeled as fr_ in that same figure). All bus signals flowing to the user logic are referred to as "write" signals and all bus signals from the user logic as "read" signals.

Bus Distribution in CSL

CSI Bus signals do not need to travel very far through the routable signals of the CSL. All signals are synchronized throughout the CSL just before entering the user networks. A complete set of the shared or common physical bus signals is available in each CSL bank. A bank includes 128 logic cells and 8 selectors. All of the synchronized data, address and control signals are logically equivalent and may be treated as aliases. The CSL configuration software can usually optimize the performance and ability to place and route by selecting the closest available alias.

The logic block architecture and programmable routing scheme for CSL was determined prior to the addition of the bus. The bus signals enter general user logic within the CSL logic through existing interconnect resources. The logic block of each configurable logic cell includes a four input LUT and a register bit. Each pair of these is grouped with an associated programmable switching matrix and routing channels and then used as a fundamental building block or logic tile.

Logic tiles are grouped into 8x8 banks that are arrayed two dimensionally to provide the required amount of Configurable System Logic embedded within a CSoC. Figure 4 shows the floor plan of a Triscend E520 device containing approximately 30,000 gates of programmable logic gates.

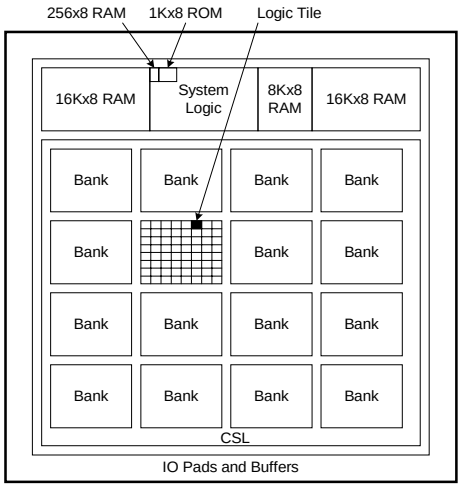


Figure 4. Physical Floor-plan of a Triscend E520

Every bank boundary has a complete set of bus pipeline registers. CSI bus signal to and from the banks are pipelined allowing the number of banks to be scaled while maintaining constant bus interface timing characteristics at the user logic.

There is one selector per column of cells in the boundary above each bank. Each selector delivers a decoded and piped read and write signal. It is fully programmable as to which address bits are included in its decode and the decode value. Bus signals are distributed to the banks by a cascade of buffers that is repeated once per bank. They are collected from the banks by a similar cascade of OR gates. A bus signal crosses the chip only once in the horizontal and vertical dimensions.

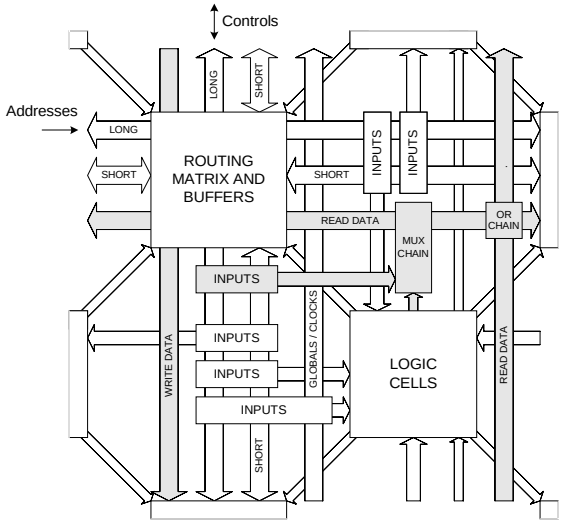


Figure 5. Signal flow in a Logic Tile

Four vertical 32-bit data busses span each bank, two write and two read, four bits per tile per bus, least significant nibble to the right. This data bus is also used to access the CSL configuration memory. The data bus signals are buffered and piped at bank boundaries. The read data path is implemented as a chain of OR gates within the bank, with one OR gate per tile. Since an 8032 microprocessor has an 8-bit bus, all data bytes are combined into one byte at the top edge of the CSL. The write data bits are driven to the existing routing resources within the tile. A horizontal 32-bit read data bus also spans each bank with four bits per tile. The horizontal data path is implemented as a bi-directional multiplexed chain with the select control derived from the programmable routing channels and the select data from the logic tile's LUT, registers, or carry chain. Within each tile, the horizontal read data may be configured to connect to the corresponding vertical read data OR chain.

The signal flow associated with each logic tile is illustrated in Figure 5. Aside from the data busses, the general interconnect includes 48 nets within each tile.

There are four sets of eight "short" nets, each set extending to one of the four adjacent tiles. Each set of eight "long" nets spans the bank, one vertically and one horizontally. At bank boundaries, vertical long nets may be used as bus control signal inputs or outputs, while horizontal long nets may be used as bus address lines.

CSI Socket Signals

The CSI bus has separate address, command protocol, write data, and read data signals. Figure 6 shows the CSI Socket bus diagram. The bus signals within the CSL are grouped functionally and described in the following sections. Within the CSL there are many copies, or entry and exit points for the synchronized bus signals, all of which are logically equivalent or aliases. The signals marked "Physical" are available to the user. The signals in the sections marked "Virtual" are not directly available to the user, only via the operation of a selector.

Common Bus Signals (Physical)

These are the shared or common CSI Bus signals that are available at many alias points throughout the CSL.

Name	Signal Description
bclk	Bus clock
dws[7:0]	Slave write data bus, into slaves
adr[31:0]	Address bus
waited	Indicates previous cycle was a wait state
waitnext	Assert to cause next cycle to be a wait state
drs[7:0]	Slave read data bus, out of slaves

Selector Signals (Physical)

Each of "n" selectors on the bus drives a pair of the following enable control signals. The index identifies the signals of one of the selectors in the CSL. The first two signals are generated when configured for address and command decodes. Alternatively, a select rather than a write select may be generated in the pair. When the selector is used for DMA, the "reqsel" and "acksel" signals are available instead.

Name	Signal Description
wrsel[n-1:0]	Decoded write enable
rdsel[n-1:0]	Decoded read enable
sel[n-1:0]	Decoded select, on read or write enable
reqsel[n-1:0]	Channel selected DMA request
acksel[n-1:0]	Channel selected DMA acknowledge

Bus Protocol Signals (Virtual)

The following signals are not directly available to the user inside the CSL though they are bussed throughout the CSL. These are handled within the dedicated decoders and synchronizing logic at the bank boundaries. Signals Wren and Rden are not directly available to the user logic, only via selectors.

Name	Signal Description
brst_	Bus reset (not directly available to user logic, only via initial values)
Wren	Write enable (not directly available to the user logic, only via selectors)
Rden	Read enable (not directly available to the user logic, only via selectors)
waitnow	Assertion causes the current cycle to be a wait state (not available to user logic)

Bus DMA Signals (Virtual)

There are two direct memory access (DMA) channels. The index identifies which channel the signals are associated with. Inside the CSL, these DMA signals are connected to the selectors.

Name	Signal Description
dmareq[1:0]	DMA requests, channel 0 and 1
dmaack[1:0]	DMA acknowledges, channel 0 and 1

External Memory Coordination Signals (Virtual)

Additional devices may be connected to the external memory interface bus. These signals allow decoding of external devices to be implemented with selectors in the CSL. The complexities of the protocol involved here are handled by the selector logic.

Name	Signal Description
xmreq	External memory cycle request out from selectors in CSL
xmact	External cycle active timing input to selectors in CSL

Breakpoint Signals (Physical)

The breakpoint generator can receive a break signal from the CSL or have the final break delivered to the CSL. They are grouped with the bus signals because they are distributed and synchronous with the bus.

Name	Signal Description
bpreq	Breakpoint request from CSL to breakpoint generator
bpevt	Breakpoint event from breakpoint generator to CSL

CSI Socket Selectors

Selectors are used to decode the bus address and command protocol. The decode is done ahead of the registers delivering the signals into each bank. This significantly improves the bus performance that may be achieved within the user programmable logic. For the ultimate bus performance, no CSL user logic resources

need to be involved in interfacing registers to the bus, only routing resources.

The selector also provides additional services in handling complexities of the bus interface protocol. The basic read and write protocol is not very complex. However, the selector is also used to add wait states and DMA transactions, and to generate a chip select and coordinate the operation of the external memory bus normally dedicated to the external configuration memory. The selector will also be used to handle word and sub-word specific decodes with wider data busses; however, presently the entire system only deals with bytes.

A selector may be configured to deliver a synchronized version of the some of the virtual bus signals. For example, to get "Wren", set all addresses in the decode to don't care. Each selector is roughly comparable to having a relatively simple but fast and wide programmable logic device (PLD) such as a 16V8 in a corresponding system board implementation.

Selector Parameters

There are a total of 70 bits for configuration and control available for each selector. They are described in the following sections. The configuration bits are set to a static value prior to overall use of the component. The DMA bits may be initialized and used in a static manner but with only two channels, they will typically be used in a dynamic manner by the application program running on the processor.

Address Decode

All 32 address bits are involved in the decode. Each bit may be included in the decode and required to be either a one or zero. Also, any address bits may be excluded, or don't care, as far as the decode is concerned. If both "s1" and "s0" are ones for an address bit, then that address bit is don't care. If both are zero for any bit then the selector is disabled. Configuration software may convert other address range definition forms, such as base and block size, to the actual physical form for the designer.

Register Name	Description
s0[31:0]	Enable decode when corresponding address is 0
s1[31:0]	Enable decode when corresponding address is 1

Wait States

When wait states may need to be generated, the first wait state must be generated by the selector. The first wait state cannot be generated by user logic because it must be generated synchronously on the same clock that the decode is first sent to the user logic.

Register Name	Description
Wse	Wait state enable (causes generation of first wait state)

Chip Select, Read or Write Enable

There are physically only two signals for the user from each selector. When the selector is used to decode addresses, rather than generating a decoded read and a decoded write, it may generate a decoded read and a decoded operation (covering both read and write).

Register Name	Description
riw	Merge read into write enable ("wrsel" becomes "sel")

DMA Enable

The selector may be used to deliver one of the DMA acknowledge signals to a DMA device and also to deliver a local DMA request signal to one of the DMA channels. The address decode is then used internal to the selector to reference the DMA channeling control bits.

Register Name	Description
dsa	DMA selector mode (1=channel DMA, 0=decoder)

DMA Channeling

These control bits are the least significant two bits of either a high or low nibble at the address decoded by this selector. They are in only one nibble as determined by the selector placement, since every other selector is dedicated to an alternate nibble. The more significant bit selects which channel it is connected or channeled through, while the least significant bit enables the connection. For these bits to be accessible, the selector must be enabled for DMAb by setting the DMA enable bit "dsa".

Register Name	Description
dmc	DMA channel (1 or 0)
dme	DMA enable (1=enable, 0=disable)

External Memory

A transaction decoded by the selector may use the external memory bus normally dedicated to the external configuration memory. The data, address, and general read and write signals may be reused, conserving general configurable pins for other uses. In the minimum case, only a chip select would be provided directly by the selector via a pin. This configuration bit enables the selector to coordinate operation with that of the memory interface controller.

Register Name	Description
xme	External memory enable

Non-CSI Socket Signals

There are other signals between the CSL and the dedicated system logic that are not part of the bus. The non-bus signal groups interfacing to the CSL are listed in the next few sections but not otherwise covered here.

Sideband Signals

Several signals to the system are neither bussed nor distributed throughout the CSL. These sideband signals interface asynchronously into the CSL only at the top edge. The interrupt requests are related to bus operations in that they may request that the processor perform some. The signals shown here are specific to the 8032 implementation.

Name	Signal Description
irq[1:0]	Interrupt requests to 8032 (INT1_ , INT0_)
hpintreq	Interrupt request at fixed highest priority
timer[3:0]	Input to 8032 timer channels (T2EX, T2, T1, T0)
rxd	Input to 8032 serial interface (RXD input)
rxdo	Output from 8032 serial interface (RXD output)
txd	Output from 8032 serial interface (TXD)
rstc	Output of processor reset control from CSL
xclk	External clock input for 32KHz oscillator
cpurst	CPU was reset input

Pad Buffer Signals

Internal logic connects to the "n" external input/output pads (PIOs) via buffers. These are connected to an extra layer of interface logic and routing tiles around the outer edge of the CSL.

Name	Signal Description
Ipio[n-1:0]	Input from input buffer
Opio[n-1:0]	Output to output buffer
Epio[n-1:0]	Enable to output buffer

Clock and Global Signals

There are six general purpose clock or global signals. These are available to the registers within the CSL. These are not part of the bus, but are bussed in the sense that they are delivered throughout the CSL with relatively low skew. Each may be driven either from an associated PIO or from within the CSL via that associated PIO.

Name	Signal Description
clk[6:1]	User available clock domains

CSoC Design Methodology

FastChip™ is a powerful development tool for Triscend Corporation's Configurable System-on-Chip (CSoC) devices. It supports design flows that match embedded system product needs. Two methods of hardware design entry are supported within FastChip. The "Derivative on Demand" design flow facilitates design re-use and speeds time to market in applications requiring standard peripherals with little or no custom design. A second more traditional design flow maximizes product differentiation and customization by supporting standard interfaces to popular 3rd party EDA design entry, synthesis, and simulation tools.

FastChip interacts with 3rd party MCU development tools that include compilers, assemblers, linkers, instruction set simulators, and debuggers. It interfaces to these tools via ASCII header files and executable hex files. A simple five step design methodology is all it takes to complete a design from concept to working silicon.

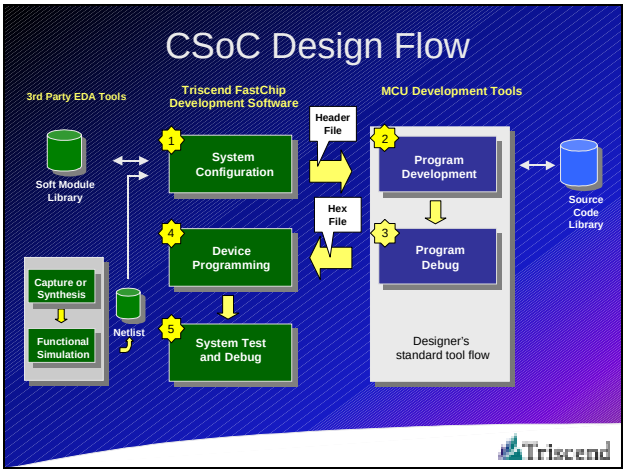


Figure 6. CSoC Design Flow

Step 1: System Configuration

The CSoC hardware is customized at this step by choosing functions from either 3rd party or Triscend's soft module library. A simple drag and drop graphical user environment with parameterized core generators facilitates design re-use. If a custom logic function is required, then 3rd party design entry tools such as Orcad and View Logic Schematic capture tool, or HDL synthesis tools such as FPGA express can be used to generate EDIF files for import.

Step 2: Generating Header File and Writing Code

A software utility within FastChip eases the task of "C" or assembly micro-controller code development. This software automatically produces a header file that assigns logical addresses and symbolic names for all user registers in the CSoC. By subsequently including the

header file in the main code, the user can easily refer to on-chip registers by their symbolic names. Header files also encapsulate code segments that initialize the device in the specified mode of operation. For more complex peripherals, whole device driver code segments are included in the header file to reduce application development cycles and improve time to market. The software application is compiled, assembled, and linked using off-the-shelf MCU development tools from Keil and Tasking.

Step 3: Debugging code and Generating Hex file
After writing the software application, 3rd party instruction set simulators allow quick code debug before generating the final Hex file that is download into the CSoC.

Steps 4: Downloading the Program
When satisfied with the proper functionality of the application software, the final CSoC configuration bit-stream is created using the Bind and Download software utility. Bind maps the logic defined for each peripheral in the Configurable System Logic (CSL) and interconnects the cells with one another, with the CSI Bus Socket, and with the PIO pins. Bind is also referred to as Map, Place, and Route in typical PLD software tools. The result of the Bind process and the object code generated by the compiler in the previous step is then merged to create a single data file called the initialization program. The initialization program configures the processor at power-up, similar to a processor's bootstrap program. The next step is to "Download" the initialization program into the CSoC.

Three methods exist to download this program. Typically the program is written into external, non-volatile memory such as FLASH memory. The FLASH device can be programmed in a device programmer or programmed in-system through the JTAG port. Likewise, the initialization program can be downloaded directly into the chip's internal SRAM memory through the JTAG port, bypassing external memory altogether. Alternatively, the initialization program can be written into an external non-volatile serial PROM device.

Steps 5: In-System debugging
The stage is now set for exhaustive in-system verification. At this point, the CSoC device is in its target application running at speed and its JTAG port is connected to a PC for debug monitoring and runtime control. The FastChip software acts as an intermediary to your standard CPU debugging tools. The debugging tool's commands are translated by FastChip software to equivalent JTAG commands that control the breakpoint unit. This allows compete control and monitoring of the hardware to manage breakpoint events and examine user register contents using 3rd party debugger interfaces.

Conclusion

A CSoC device with a powerful on-chip bus was described in this paper. This bus provides a standard framework for delivering quick time-to-market custom micro-controller solutions. A software development environment was also described that provides a graphical, user friendly development environment that interfaces to standard 3rd party EDA and MCU development tools.

References

"Triscend E5 Configurable Processor Family," Triscend Corporation Press Release, November 1998.

"A Bus Centric Configurable Processor System," Press Article.

"Triscend E5 Configurable Processor Family", Product Description.
"Configurable Processors: An Emerging Solution for Embedded Systems Design." White Paper, Triscend Corporation.
"Triscend Configurable System-on-Chip Learning Center," Application available at www.triscend.com

"Configurable System Interconnect Bus User's Guide" Triscend Corporation