Whitepaper

Advanced SoC Debug with Multi-FPGA Prototyping

Author:
Steve Walters

Release date:
April 2022

Contents

As SoC designs advance in complexity and performance, and software becomes more sophisticated and SoC-dependent, SoC designers face a relentless push to “shift left” the co-development of the SoC silicon and software to improve time-to-market. Consequently, SoC verification has evolved to include multi-FPGA prototyping, and higher prototype performance, to support longer runs of the SoC design prototype, running more of its software, prior to silicon – in an effort to avoid the skyrocketing costs associated with silicon respins. While FPGA prototyping for SoC design verification by its nature remains a “blunt instrument”, FPGA prototyping is still the only available pre-silicon verification option, beyond hardware emulation, for achieving longer periods of SoC design operation capable of running software, and, in some cases, “plugging” the SoC design prototype directly into real target-system hardware. Not surprisingly, commercial FPGA prototype suppliers are using the latest FPGA technology to implement FPGA prototyping, offering multi-FPGA prototyping platforms, and advancing FPGA prototyping debug tool capabilities, to meet customer demands for more effective SoC verification.

Ideally, SoC design debug tools for FPGA prototyping would enable software simulation-like verification and debug at silicon speeds – providing visibility of all internal SoC design nodes, not impede prototype performance, provide unlimited debug trace-data storage, and be quickly reconfigurable for revisions to the SoC design and/or the debug setup. In reality, today’s SoC design debug tools for FPGA prototyping falls short of the ideal, and multi-FPGA prototyping adds to the challenge of achieving ideal SoC design debug tool capabilities. As a result, today’s FPGA prototyping for SoC design debug offers tradeoffs among the ideal debug tool capabilities, and it is left to the SoC design verification team to configure an “optimal” verification strategy for each SoC design project – with consideration for future scaling-up and improved verification capabilities.

This white paper reviews some of the multi-FPGA prototyping challenges for SoC design verification and debug, and, reviews one example of a commercially available multi-FPGA prototyping debug capability offered by S2C Inc., a leading supplier of FPGA prototyping solutions for SoC design verification and debug (s2ceda.com).

Figure 1. S2C Multi-FPGA Prototyping Platforms and MDM Pro Debug Hardware

FPGA Prototype Essential Debug Tool Characteristics

The *expected value* of an FPGA prototype for verification/debug is derived from enabling SoC design operation in a modeled end-product context (or an actual end-product) for long periods of time, while executing the end-product's software – with the objective of verifying "intended SoC design operation" prior to committing to silicon. The *expected value* of an FPGA prototype is achieved when an SoC designer can identify all causes of an SoC design's failure to operate as intended prior to silicon – hence, advanced FPGA prototyping debug tool capabilities are critical to achieving the expected value from the FPGA prototype.

When considering an FPGA prototype debug solution, the *essential debug tool characteristics* include;

✓ A plethora of debug probes

✓ "Deep" debug trace-data storage

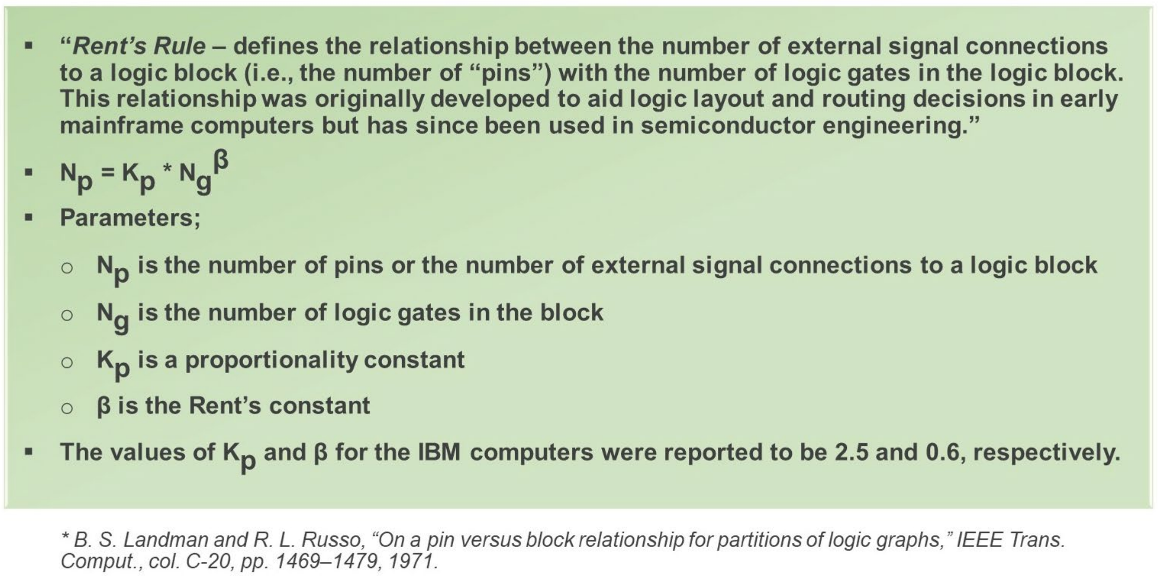
✓ High protype performance

✓ Fast FPGA prototype reconfiguration

Figure 2. Essential Debug Tool Characteristics

The implementation itself of each of these *essential debug* tool characteristics can create impediments to achieving the *Ideal* SoC design debug tools for FPGA prototyping previously described. For example, debug probes consume FPGA routing resources and will impact the logic utilization of the FPGAs and may impact prototype performance. Another example is when debug trace-data storage is implemented with FPGA storage resources, where the debug trace-data storage is limited by the on-FPGA storage resources, and the needed debug trace-data storage may compete with the storage required by the SoC design prototype itself. Further, if debug probes consume a large number of FPGA I/O, the useful FPGA logic utilization will be reduced, and the prototype performance may be impaired – and if the debug probes need to be recompiled for FPGA download every time the debug probes are reassigned, the FPGA reconfiguration times will be extended, stretching out the total debug time consumed by many reconfigurations.

Debug Probes – debug probes are "physical" connections that must be connected to design prototype "physical" nodes inside the FPGA, and therefor will compete with the SoC design itself for FPGA routing resources, both for internal FPGA interconnect, and for external FPGA interconnect in a multi-FPGA prototype. When implementing any multi-FPGA prototype of an SoC design, an empirical relationship has been described between the size of a design logic block (often referred to as "cut size" when partitioning a design for multi-FPGA implementation), and the number of I/O needed to access the logic contained in the "cut". One such empirical relationship is referred to as "Rent's Rule" (Figure 3.). Said differently, consuming FPGA I/O for debug probe connection will limit the amount of useful FPGA logic available for prototype implementation, and, to exacerbate the multi-FPGA partitioning challenge, the raw logic capacity of new generations of prototyping FPGAs is increasing faster than the number of FPGA I/O is increasing. When thousands of debug probes are desired, and considering that today's leading prototyping FPGAs are configured with less than 2,000 single-ended I/O (Xilinx's VU19P FPGA has 1,976 single-ended I/O), the impact on prototype implementation can be significant if FPGA I/O are allocated to accessing a large number of debug probes.



- "*Rent's Rule* – defines the relationship between the number of external signal connections to a logic block (i.e., the number of "pins") with the number of logic gates in the logic block. This relationship was originally developed to aid logic layout and routing decisions in early mainframe computers but has since been used in semiconductor engineering."
- $N_p = K_p * N_g^{\beta}$
- Parameters;
  - N_p is the number of pins or the number of external signal connections to a logic block
  - N_g is the number of logic gates in the block
  - K_p is a proportionality constant
  - β is the Rent's constant
- The values of K_p and β for the IBM computers were reported to be 2.5 and 0.6, respectively.

** B. S. Landman and R. L. Russo, "On a pin versus block relationship for partitions of logic graphs," IEEE Trans. Comput., col. C-20, pp. 1469–1479, 1971.*

Figure 3. Rent's Rule

The number of debug probes will also determine the "structure" of the trace-data storage (width and depth). For a fixed amount of trace-data storage, the trace-data width (the number of probes) will determine the number of trace-data "samples" that can be captured (trace-data storage depth) – kind of a "zero-sum game". If you are limited to 1,000 bits of trace-data storage, and the trace-data width is 100 bits (100 probes), the maximum number of trace-data stored samples will be 10. Alternately, if the trace-data width is 10 bits, the maximum number of trace-data stored samples will be 100. As the number of trace-data probes increases to large numbers, probe data multiplexing may be necessary to extract the trace-data from the FPGA through a very small number of I/O pins – leading to another important tradeoff between trace-data width, depth, and debug tool performance.

Available Debug Trace-Data Storage – depending on the implementation of the debug tools, trace-data storage may use FPGA internal storage resources. If "external" trace-data storage is used for implementation, no FPGA storage resources are consumed, and the trace-data storage depth can be scaled to meet the debug strategy requirements by adding more external storage memory. Alternatively, very "wide" trace-data (a large number of debug probes) will need to be multiplexed "down" to a few FPGA I/O. which will necessarily reduce debug tool performance. The FPGA prototype user should look for tradeoff flexibility to optimize the trace-data storage depth, width, and performance.

Debug Solution Performance – its challenging enough to get a complex SoC design running correctly on an FPGA prototype at the targeted performance – without being performance-limited by the implementation of the FPGA debug tools. Unfortunately, without *effective* debug tools, the FPGA prototype "value" is diminished in a verification environment. Similarly, if a large number of debug probes causes significant FPGA routing congestion (internal or external), the prototype performance will be degraded. Further, if implementing a large number of debug probes requires many levels of multiplexing to access the probe trace-data, the prototype performance will be reduced proportionally. The FPGA prototype may be capable of running at 50MHz with fewer debug probes, but the prototype performance may only be 25MHz with many more debug probes. FPGA debug tools that offer programmable options for configuring debug probe width, trace-data storage depth, and prototype performance will provide users with the flexibility to optimize the prototype debug tools for a given FPGA prototyping project.

Reconfiguration Time – SoC design debug is inherently iterative during the SoC verification process on the path to silicon tapeout, including the FPGA prototyping phase of SoC verification. Fix a design bug, then run until the next design bug stops further design verification progress. Then, debug some more by capturing key design signals (probes) during design operation up to the time when *"symptoms"* occur that indicate that a bug has occurred, and iterate prototype runs until the bug *"root-cause"* is identified. Clearly, this is an oversimplification of the process, but it should also be clear that the debug process is unavoidably iterative. The actual bug *root-cause* may have occurred many tens of thousands of clock cycles prior to the appearance of the bug *symptoms*, and the bug *root-cause* trace-data may be inaccessible by not having anticipated the placement of debug probes on critical design signals. As bugs are discovered, they will be fixed by the designers and a new version of the design RTL incorporating the fix will be released into the SoC verification environment. During this debug process, it may take several iterations of the FPGA debug tool configuration to isolate the bug *root-cause* so that the bug can be diagnosed and fixed. Throughout the SoC design verification process, there may be hundreds of reconfigurations of the design and/or the FPGA debug tools – in extreme cases, the design and/or the FPGA debug probes will need to be reconfigured multiple times a day. To the extent that the implementation of the FPGA debug tools can contribute to reducing the number of needed reconfigurations (therefor the reconfiguration time) – the total debug time will be reduced.

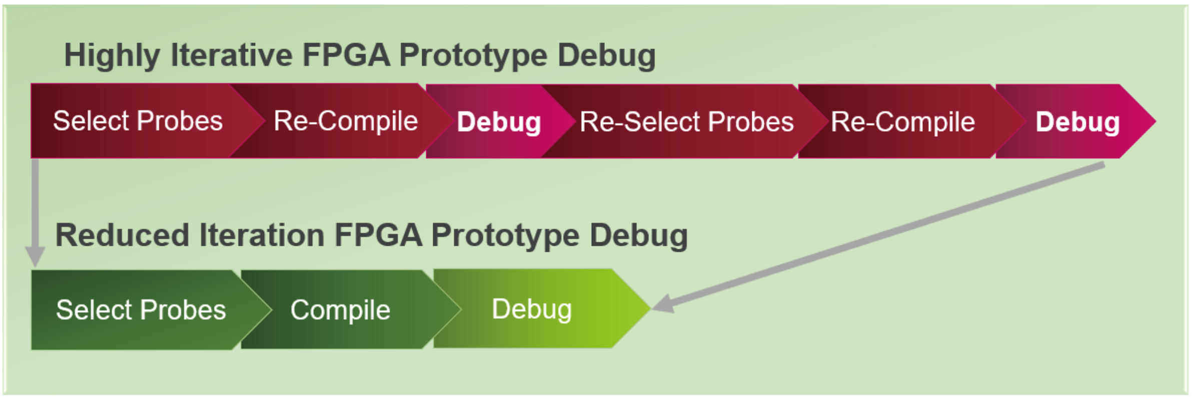


Figure 4. Soc Design Debug Is Inherently Iterative

A Word About Multi-FPGA Prototyping

Prototyping an SoC design with multiple FPGAs is not a novel concept. Early hardware "emulation" systems for SoC design verification and validation faced the same hardware implementation challenges as the FPGA prototyping systems of today. Over the past 2 decades, emulation has focused on scale and reducing *user involvement* in the prototype design implementation (higher automation) at the expense of *lower prototype performance* and *higher cost*, while FPGA prototyping has focused on the *highest prototype performance* and *lower cost* at the expense of *higher user involvement* in the prototype design implementation (more user involvement). Consequently, today's emulators achieve SoC design prototype performances of a few mega-Hertz, while FPGA prototyping can achieve SoC design prototype performances that are 10 times or more higher than the performance of emulators – consequently, today's SoC designers have evolved their verification methodologies to include a combination of software simulation, hardware emulation, and FPGA prototyping, to "shift-left" the SoC-based product time-to-market.

The point of this comparison between emulation and prototyping is that debugging a complex SoC design in an multi-FPGA prototype (or an emulator) implies extensive probing of internal design nodes during design prototype operation, and, as mentioned earlier in this white paper, the implementation of debug probes competes with the same FPGA resources used to implement the SoC design prototype itself, and impacts the number of FPGAs required, the prototype performance, and the prototype cost – for both emulation and FPGA prototyping. More probes consume more routing resources. Also, large amounts of trace-data storage requires significant storage resources – and, more of both impacts the performance and reconfiguration times of the prototype. This, in turn, impacts the effectiveness of the FPGA prototype debug tools for accelerating time-to-market. These considerations have driven FPGA prototype suppliers to optimize the implementation of debug probes, and trace-data storage – by multiplexing debug probes for access with a few high-speed FPGA I/Os, and external memory hardware.

Maximizing the number of "available" debug probes while considering the impact on prototype performance and reconfiguration time, and maximizing trace-data storage while considering the impact on prototype performance and consumption of internal FPGA storage resources, are two examples of FPGA prototyping debug tool optimizations.

Prodigy Multi-Debug Module Pro (MDM Pro™)

To optimize the implementation of its SoC design debug tools for FPGA prototyping, S2C has chosen a combination of external hardware, soft IP implemented in the FPGA, high-speed FPGA I/O, and debug configuration software for its MDM Pro. MDM Pro was designed specifically to support multi-FPGA prototype implementations – with support for *high probe-counts, deep-trace debug data storage, optimization of debug reconfiguration* compiles, and with the ability to choose debug configuration tradeoffs to optimize prototype performance. The MDM Pro debug capabilities are delivered through a combination of the *MDM Pro hardware,* S2C's *Prodigy™ FPGA prototyping platforms,* and S2C's *Player Pro™ software*. The *Player Pro* software supports user-friendly debug configuration, complex trace-data capture triggering, and single-window viewing on the user console of simultaneous streams of trace-data from multiple FPGAs. The *MDM Pro* hardware supports high-performance deep-trace debug data storage without consuming internal FPGA storage resources. S2C's *Prodigy FPGA prototyping platforms* support SoC design implementation with user-selected debug probes inserted, and high-speed transmission of the deep-trace debug data to the *MDM Pro* hardware through FPGA GTY transceivers over mini-SAS connectors between the *MDM Pro* hardware and the *Prodigy FPGA prototyping platforms.*

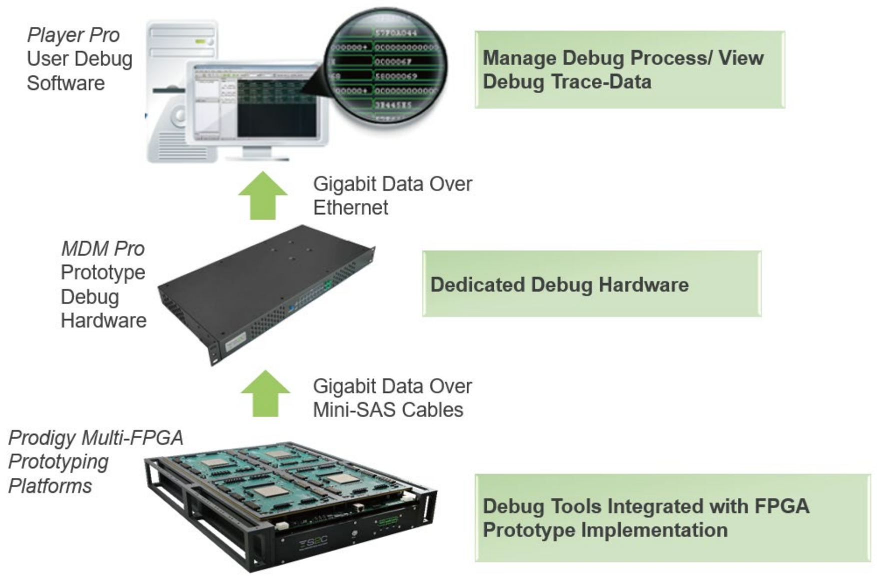

Player Pro
User Debug
Software
Manage Debug Process/ View Debug Trace-Data
Gigabit Data Over Ethernet
MDM Pro
Prototype
Debug
Hardware
Dedicated Debug Hardware
Gigabit Data Over Mini-SAS Cables
Prodigy Multi-FPGA
Prototyping
Platforms
Debug Tools Integrated with FPGA Prototype Implementation

Figure 5. MDM Pro Overview

MDM Pro Prototype Debug Hardware: the *MDM Pro debug hardware* is organized to optimize the implementation of debug probe connections, the trigger hardware for trace-data capture, the trace-data storage memory, and the transmission of the trace-data to the user console for waveform viewing – providing debug configuration flexibility, optimizable prototype performance, and minimized reconfiguration time.

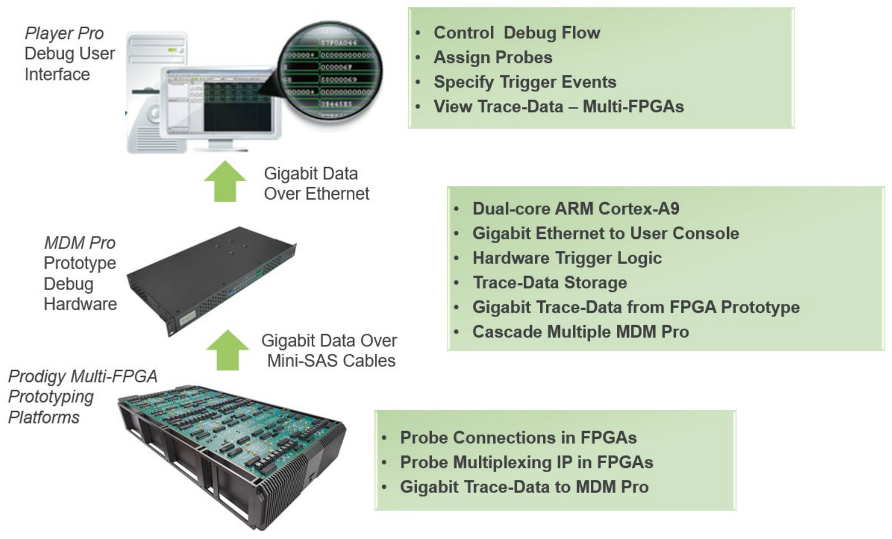


Figure 6. MDM Pro Debug Hardware Components

MDM Pro debug hardware supports FPGA prototype debug with the configurable features listed in Figure 7.

- Up To 16K Probes Per FPGA In 8 Groups of 2K Probes
- 64GB of Trace-data Storage Memory in the MDM Pro Hardware
- Advanced Trigger Conditions Implemented in the MDM Pro Hardware
- Probe Multiplexing In the FPGA to Conserve I/O
- High-Speed GTY I/O for Sending FPGA Trace-Data to the MDM Pro Hardware
- Simultaneous Viewing of Trace-Data from Up to 8 FPGAs Per MDM Pro
- Cascade MDM Pros to View Trace-Data from More Than 8 FPGAs

Figure 7. MDM Pro Prototype Debug Features

To minimize the use of FPGA I/O, and minimize the impact on FPGA logic utilization, debug probes are multiplexed for transmission from the *Prodigy FPGA prototyping platform* to the *MDM Pro hardware* with FPGA high-speed GTY transceivers over mini-SAS cables – as illustrated in Figure 8. MDM Pro IP is provided to implement the probe multiplexing and will be downloaded into the FPGA prototype with the SoC design and the debug probe connections. The "level" of debug probe multiplexing is configured to support the user-specified number debug probes for collecting debug trace-data – and, due to the multiplexing hardware delays, the level of probe multiplexing will impact prototype performance (see Figure 13).

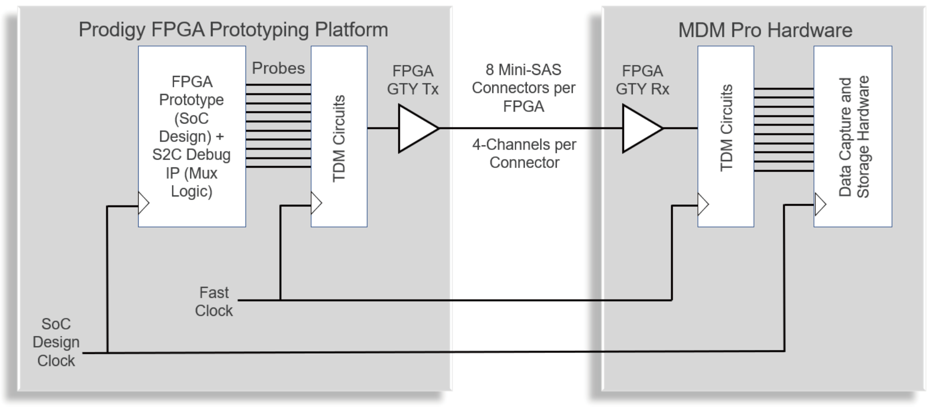
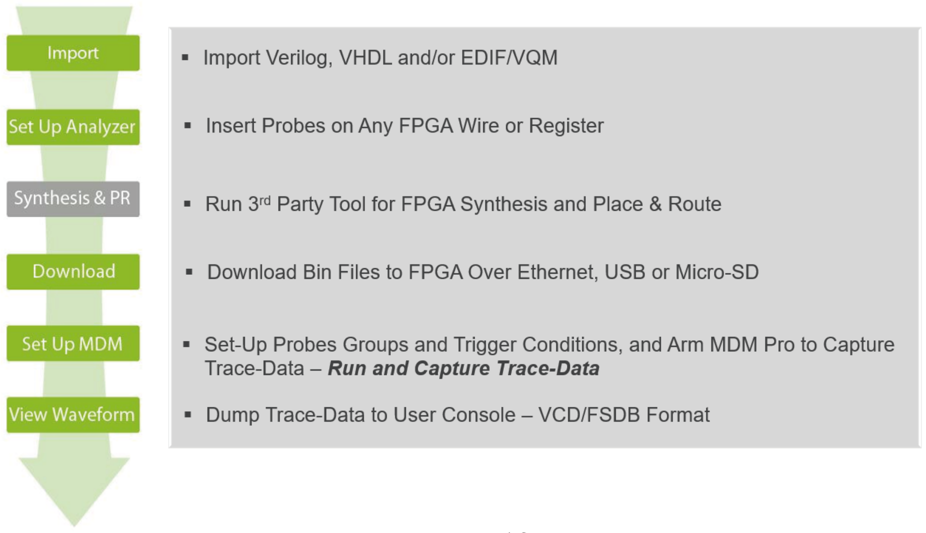

Prodigy FPGA Prototyping Platform
FPGA Prototype (SoC Design) + S2C Debug IP (Mux Logic)
Probes
TDM Circuits
FPGA GTY Tx
8 Mini-SAS Connectors per FPGA
4-Channels per Connector
MDM Pro Hardware
FPGA GTY Rx
TDM Circuits
Data Capture and Storage Hardware
Fast Clock
SoC Design Clock

Figure 8. MDM Pro Debug IP for Probe Multiplexing

MDM Pro Workflow: An overview of the MDM Pro workflow flow is provided in Figure 9.

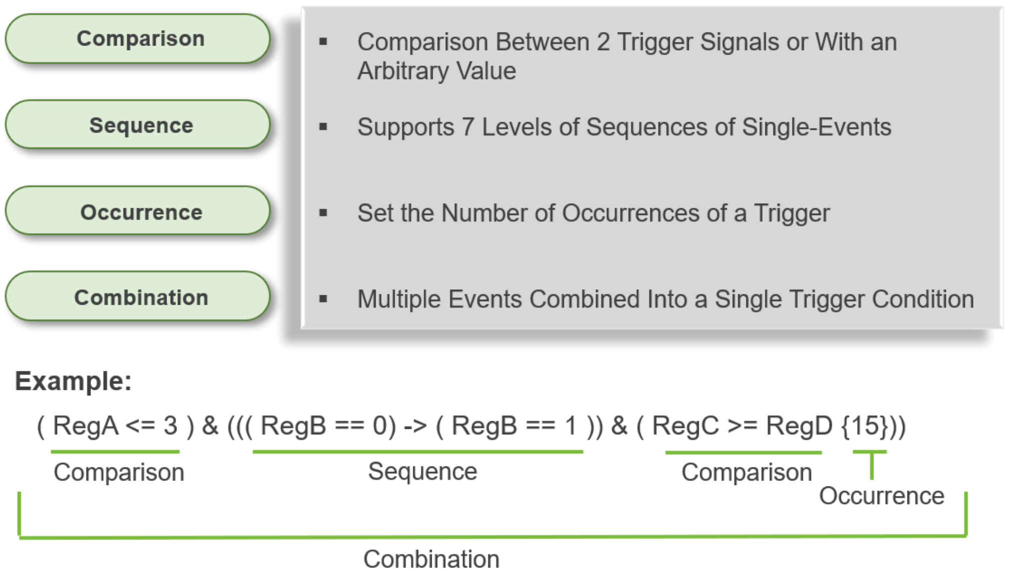


- Import Verilog, VHDL and/or EDIF/VQM
- Insert Probes on Any FPGA Wire or Register
- Run 3rd Party Tool for FPGA Synthesis and Place & Route
- Download Bin Files to FPGA Over Ethernet, USB or Micro-SD
- Set-Up Probes Groups and Trigger Conditions, and Arm MDM Pro to Capture Trace-Data – ***Run and Capture Trace-Data***
- Dump Trace-Data to User Console – VCD/FSDB Format

Figure 9. MDM Pro Workflow Overview

The user identifies a "generous" number of wires or registers, at the imported design-level, that "might" need to be viewed during debug – potentially more probes than the user expects to view during any single debug run of the FPGA prototype. *All* of these probes will later be programmed into the FPGA for debug. The advantage of configuring more probes than might be required for any single debug run is that capturing trace-data from *any* of the "pre-configured probes" will not require a re-compile of the FPGA to switch from one "set of active probes" to another set of probes – potentially reducing the number of FPGA re-compiles during a debug session and reducing the overall debug time.

MDM Trace-Data Capture Trigger: advanced debug triggering can reduce the time to converge on the root-cause of a bug, fix the problem, and move on to the next debug sequence. The features of MDM Pro's complex trace-data capture triggering are summarized in Figures 10 and 11.

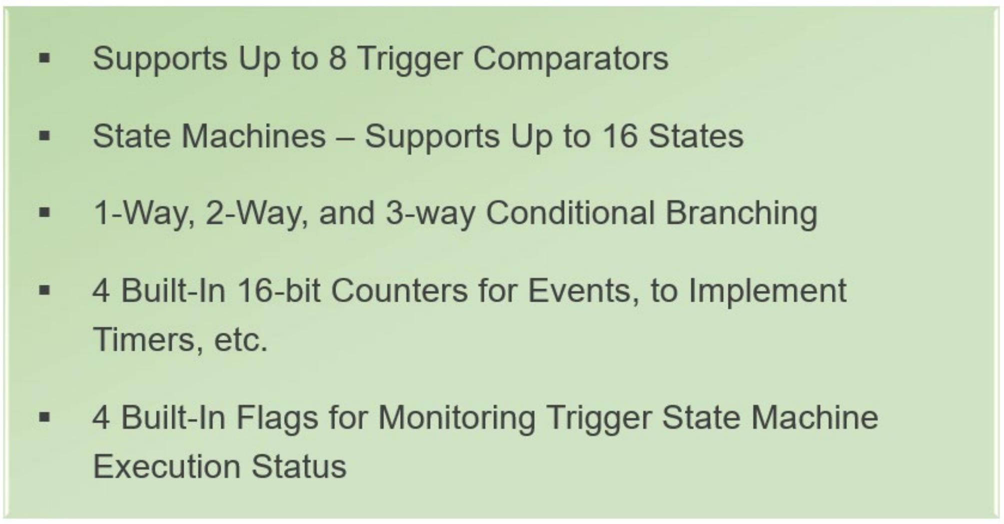


Comparison	▪ Comparison Between 2 Trigger Signals or With an Arbitrary Value
Sequence	▪ Supports 7 Levels of Sequences of Single-Events
Occurrence	▪ Set the Number of Occurrences of a Trigger
Combination	▪ Multiple Events Combined Into a Single Trigger Condition

Example:

(RegA <= 3) & (((RegB == 0) -> (RegB == 1)) & (RegC >= RegD {15}))

Figure 10. MDM Pro Trace-Data Capture Triggering



- Supports Up to 8 Trigger Comparators
- State Machines – Supports Up to 16 States
- 1-Way, 2-Way, and 3-way Conditional Branching
- 4 Built-In 16-bit Counters for Events, to Implement Timers, etc.
- 4 Built-In Flags for Monitoring Trigger State Machine Execution Status

Figure 11. MDM Pro Capture Trigger State Machine

MDM Trace-Data Storage Memory: MDM Pro trace-data storage is implemented with 64GBytes DDR4 memory in the MDM Pro hardware. When probes are inserted into the FPGA implementation of the SoC design, and the prototype design is allowed to "run", trace-data is "flowing" from the FPGA prototype to the MDM Pro debug hardware. Trace-data from the FPGA debug probes is continuously stored in a trace-data memory that is organized as a "circular buffer" where new data is allowed to continuously overwrite old data as the trace-data buffer fills – until the user-programmed trace-data trigger conditions are met (Figure 12.). When the trigger conditions are met, the trace-data memory begins storing trace-data until the trace-data buffer fills again, or until the programmed number of trace-data samples are captured, and then the trace-data memory stops storing new trace-data – and the captured/stored trace-data is uploaded to the user console for viewing. As mentioned earlier, trace-data may be simultaneously captured/stored from 1 FPGA, or several FPGAs, and viewed in a single viewing window with the *Player Pro* software.

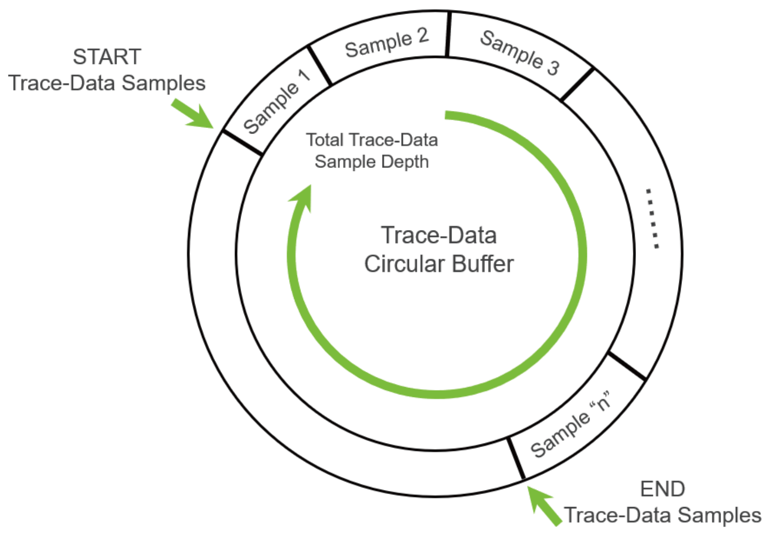


Figure 12. MDM Pro Trace-Data Storage Circular Buffer

For optimum flexibility, and optimum performance, the MDM Pro trace-data storage "structure" is automatically configured for different debug requirements – first, to support the number of user-specified probes (trace-data storage width), and then the trace-data storage depth (number of samples) is in turn determined by the total available hardware storage memory (64GBytes). The maximum MDM Pro performance is determined for each configured structure. Figure 13. summarizes the supported combinations of trace-data storage width and depth – and maximum MDM Pro performance.

- Max Speed: **256 signals per FPGA @ 125 MHz**
- Max Visibility: **2K signals per FPGA @ 15 MHz**

NUMBER OF FPGAS	PROBE WIDTH (BITS)	MAX SAMPLE FREQUENCY (MHZ)	MAX SAMPLE DEPTH (SAMPLES)
1-FPGA	1~ 256	125	2G
	257 ~ 512	62.5	1G
	513 ~1,024	31.25	512M
	1,025 ~ 2,048	15.625	256M
2-FPGAs	2 ~ 512	125	1G
	513 ~ 1,024	62.5	512M
	1,025 ~ 2,048	31.25	256M
	2,049 ~ 4,096	15.625	128M
4-FPGAs	4 ~ 1,024	125	512M
	1,025 ~ 2,048	62.5	256M
	2,049 ~ 4,096	31.25	128M
	4,097 ~ 8,192	15.625	64M
8-FPGAs	8 ~ 2,048	62.5	256M
	2,049 ~ 4,096	31.25	128M
	4,097 ~ 8,192	15.625	64M
	8,193 ~ 16,384	7.8125	32M

Figure 13. MDM Pro Trace-Data Width and Depth

Summary and Conclusions

S2C's MDM Pro hardware, together with S2C's Prodigy FPGA prototyping platforms, and S2C's Player Pro software, implements a rich set of debug features that provides SoC designers with the flexibility to optimize the FPGA prototype debug tools for a given FPGA prototyping project. MDM Pro combines off-FPGA hardware for "deep" trace-data storage and complex hardware trigger logic, in combination with probe multiplexing

IP in the FPGA to access a large number of debug probes over a few FPGA high-speed GTY connections to minimize the consumption of FPGA I/O, and the ability to setup more probe connections than need to be viewed at the same time so that more probes may be viewed when needed without recompiling the FPGA or degrading the debug performance. Player Pro software for debug compliments the debug hardware with a powerful user interface for managing the debug setup, configuring advanced trace-data trigger conditions, initiating debug runs of the FPGA prototype, and viewing the debug trace-data from multiple FPGAs in a single viewing window.