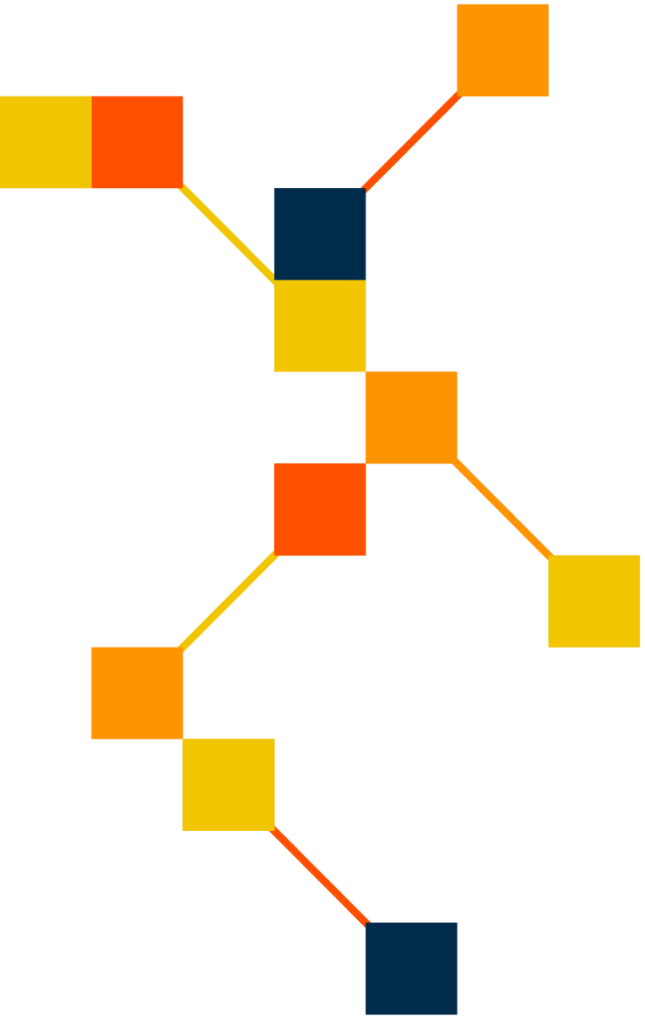
2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
MUNICH, GERMANY
OCTOBER 15-16, 2024



Modernizing the Hardware/Software Interface

Bringing your register design into the 21st Century

Insaf Meliane - Sr. Product Manager
Tim Schneider - Sr. Staff Applications Engineer

October, 2024

ARTERIS IP

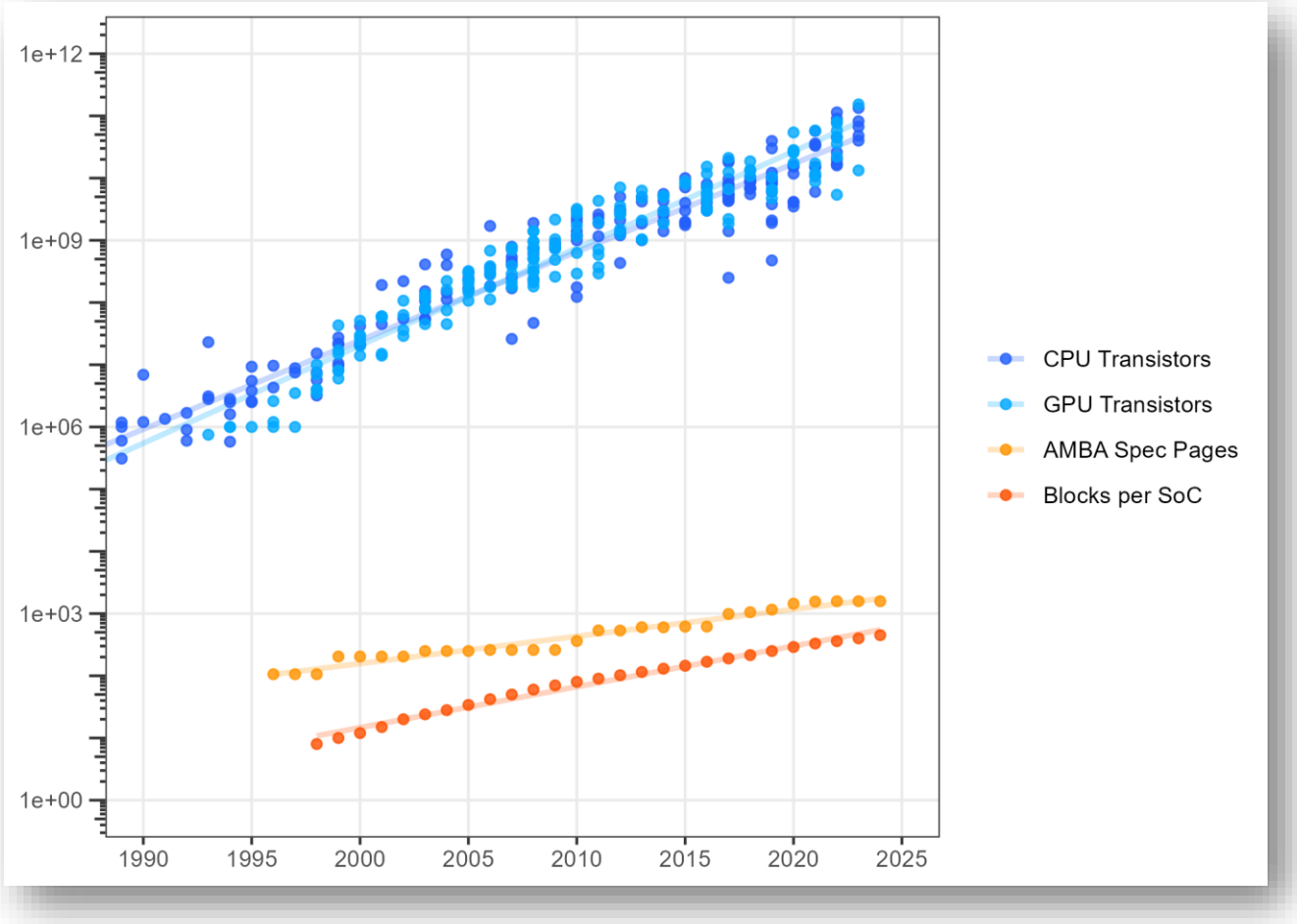
2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
MUNICH, GERMANY
OCTOBER 15-16, 2024

Industry Challenges

Industry Challenges

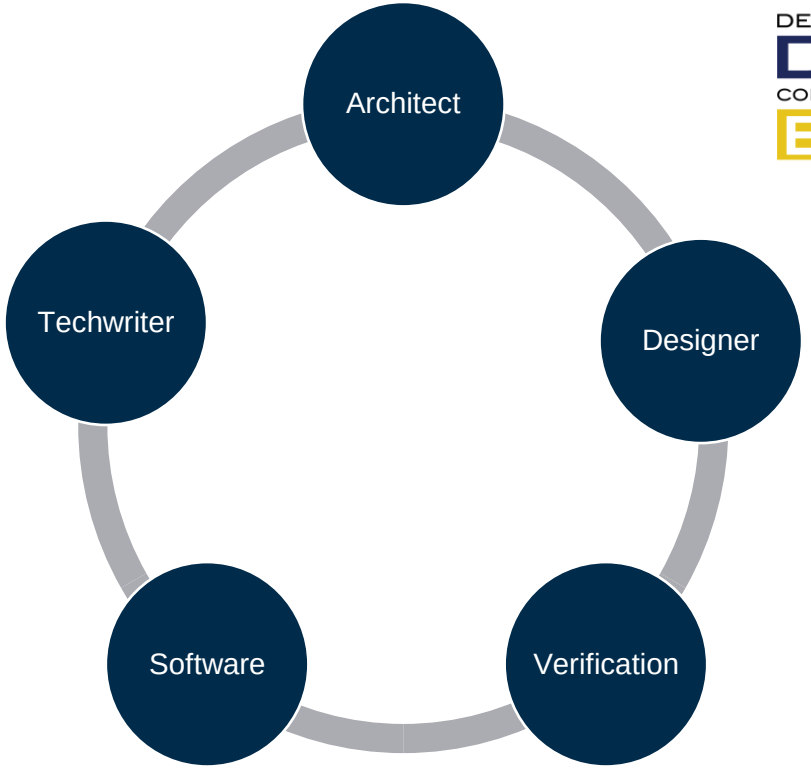
Hardware Software Interfaces (HSIs) are gaining in criticality!

- Designs continue to grow in size and complexity
  - 500+ up to 1K IP blocks
  - 200K+ up to 5M+ registers
- Content integration from various sources
  - Significantly higher portion of third-party IP hitting your designs
    - How to detect and resolve issues early in the design cycle
  - Ease of combining legacy design data into current designs



Industry Challenges

Different disciplines/teams need to interact



- HW/SW Interfaces (HSIs) are critical
  - Many disciplines involved in the development process
- Limited cross-team synergy
  - Team “silo’s” and parallel design process
- Miscommunication
  - Lead to specification misalignment
  - Can cause disaster and may require late fixes or even re-spins if issues cannot be resolved in software
    - Chance of a Respin Due to HSI Issues: ***1 in 7 Designs***
- Inconsistent data (HW/SW/documentation) across proprietary or legacy databases and flows
- Many different forms of definition exist - Spreadsheets, IP-XACT, SystemRDL…

Industry Challenges

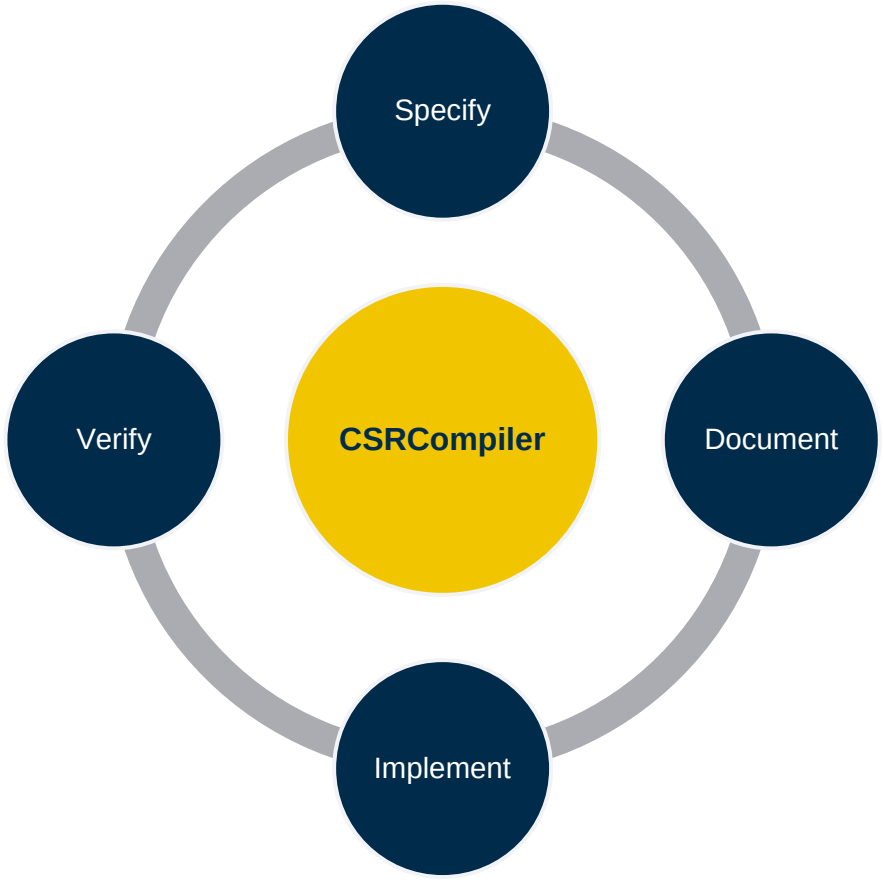
Traditional design methodologies reaching their limits

- Unproductive use of limited engineering resources
  - Tedious writing and maintaining homegrown scripts and solutions
  - Significant manual efforts to generate accurate designs
    - Across a large variety of design tasks
    - Not automated, error-prone
    - Missing capabilities/ have legacy limitations
    - To keep up with changing standards, enhancements & bug fixes
- Aggressive schedule

2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
MUNICH, GERMANY
OCTOBER 15-16, 2024

Arteris Solution

CSRCompiler
Automate Hardware/Software Interface (HSI)



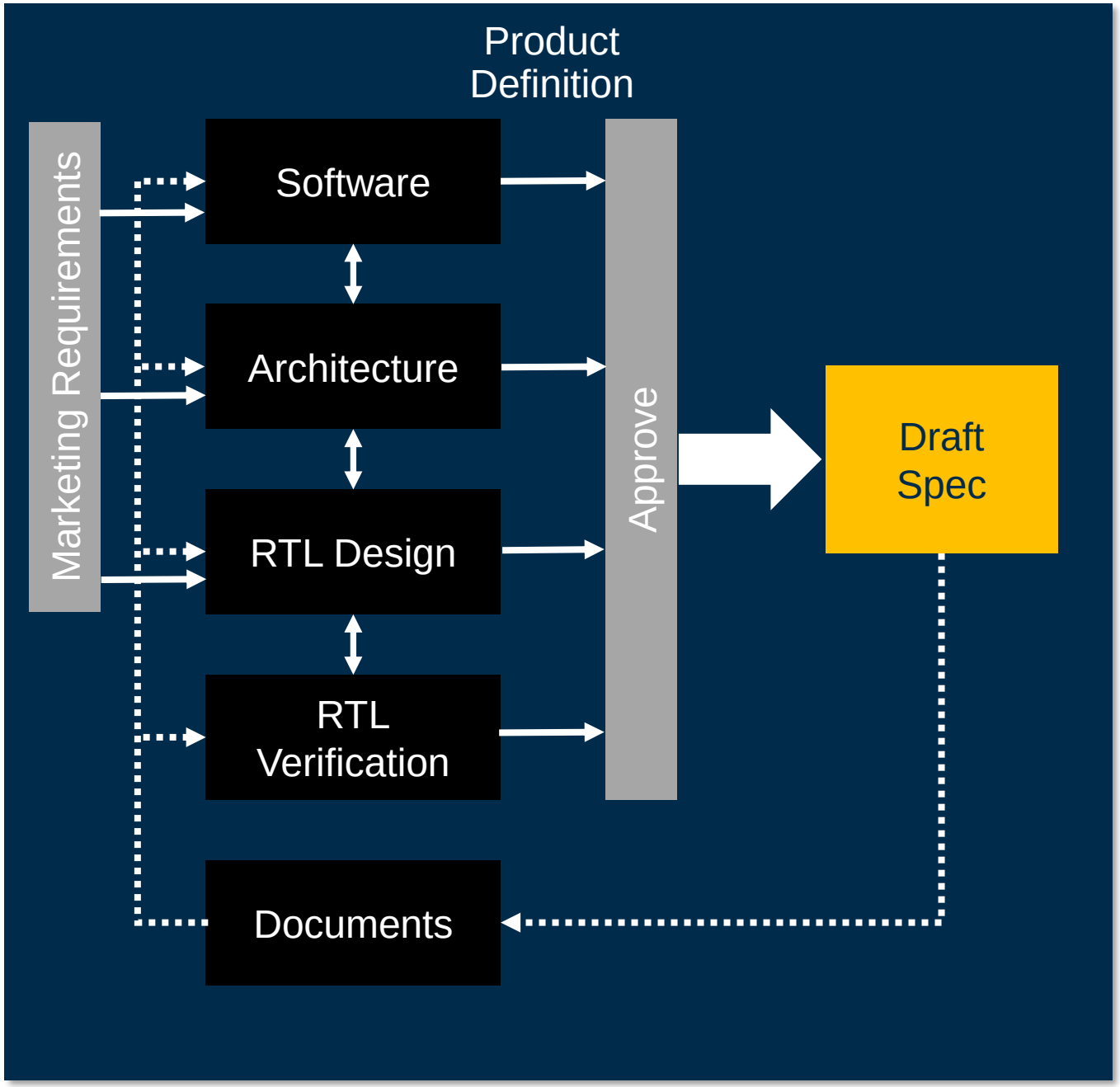
- Automate the generation of all outputs for hardware and software interface implementation (RTL, SW, DV, DOC)
- Eliminate time-consuming and error-prone manual scripting and editing of design data
- Provide a solution that has the features and flexibility to design the largest and most complex designs in the world today

CSRCompiler Design Flow

Product Definition

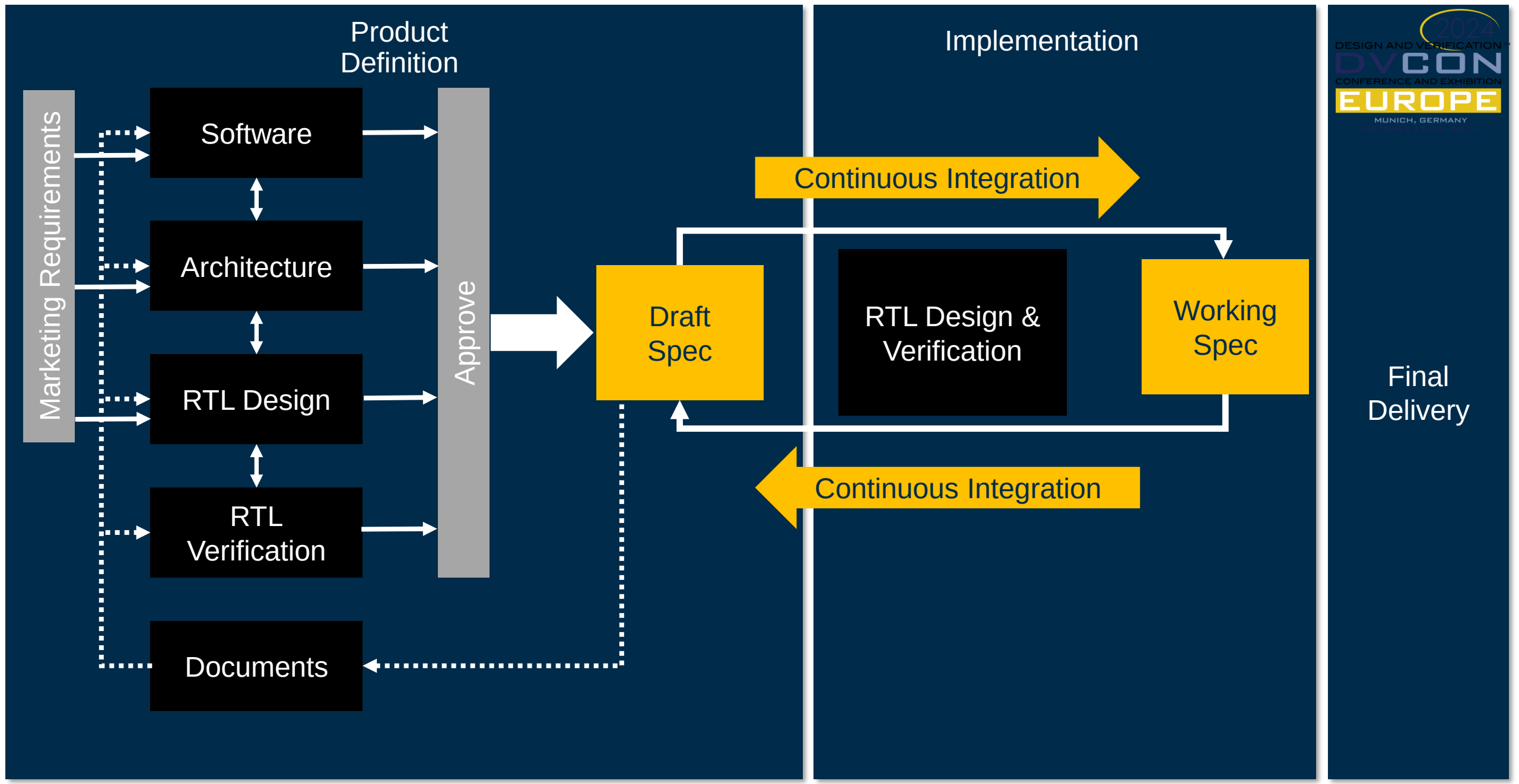
Implementation

Final Delivery



Implementation

Final Delivery



ARTERIS IP

Product Definition

Implementation

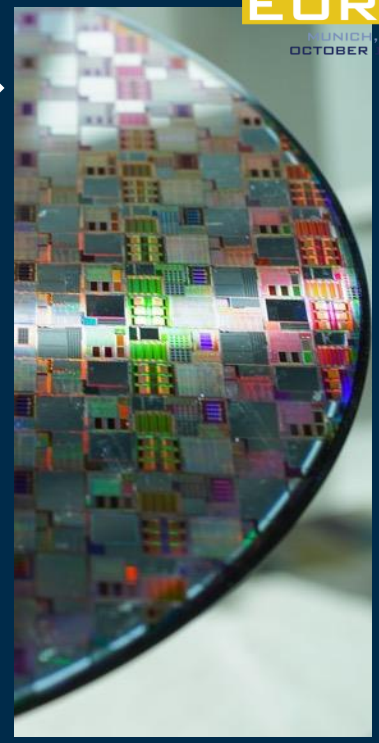
Working Spec

Approve

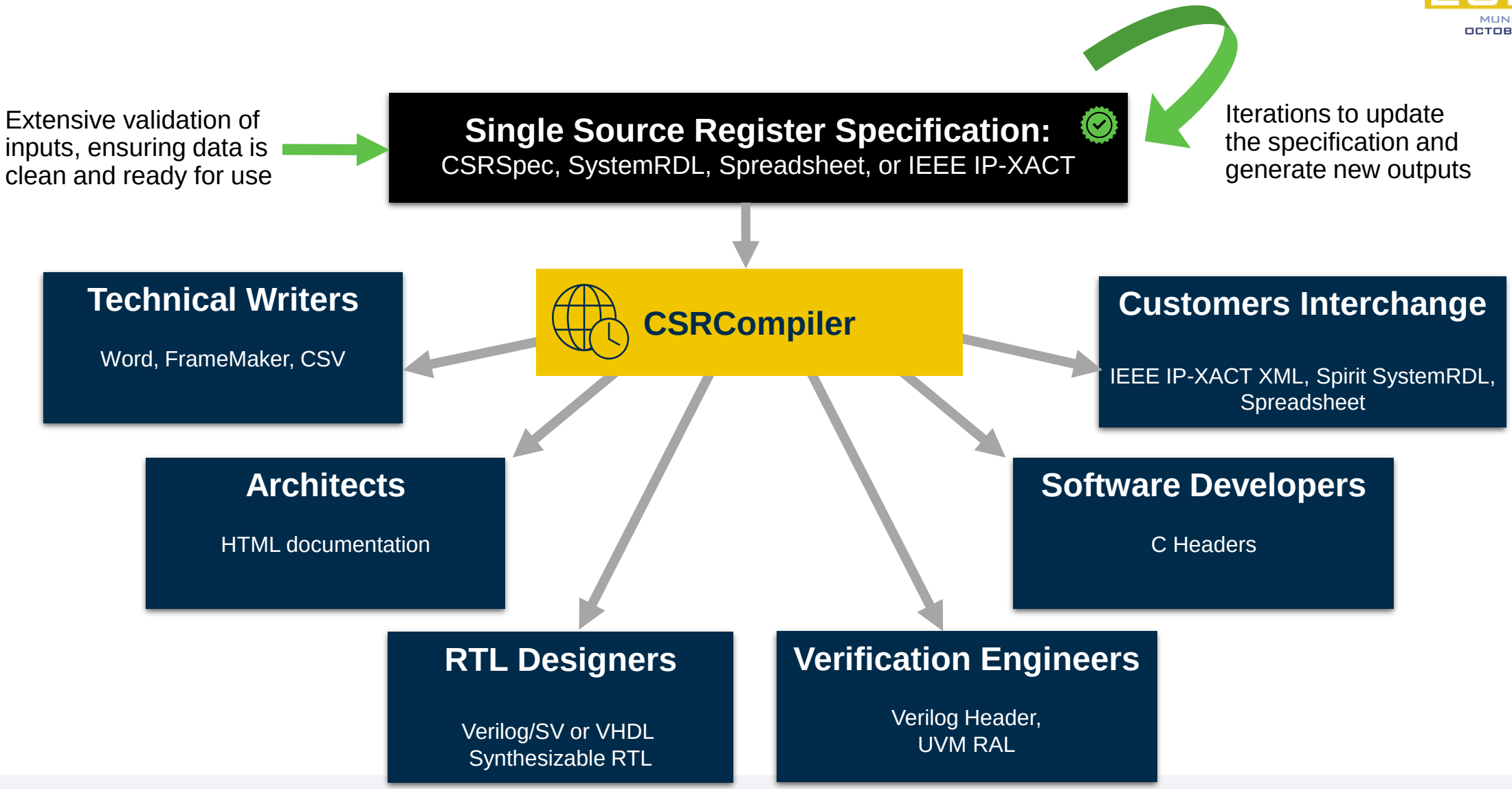
Final Delivery

Final RTL

Final Spec



Single Source of Truth
Unified Specification and Compilation Flow



2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
MUNICH, GERMANY
OCTOBER 15-16, 2024

Technical Capabilities

1. High capacity & fast performance

2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
MUNICH, GERMANY
OCTOBER 15-16, 2024

- Up to 5 Million registers
- Generates 100,000 registers in seconds
- Proven to compile 5 million registers in less than 10 min to RTL implementation

2. True cross compiler reading various formats

- IP-XACT
- SystemRDL
- Spreadsheets
- SRCSpec

3. High quality RTL

- Correct by construction RTL
- Extensive error checking
  - Over 1,000 Error Checks
  - Functional, behavioral, syntactic, and semantic error checks
- Built-in ability to validate intellectual property from third-party or internal legacy data
  - All inputs are verified for semantic and syntactic correctness
  - Ensuring data is clean, verified, and ready for use.

4. Powerful generator

- Produces all required views, formats
  - RTL register bank (vhdl, Verilog, systemVerilog)
  - Generates all versions of IP-XACT
  - SystemRDL
  - C Header Outputs
  - UVM ral
  - Docx/HTML documentation
  - FrameMaker

5. Highly Configurable

- Over 400 configuration properties
- Highly configurable parameterized templates
- Full outputs customization with no additional scripting or manual intervention needed
  - Automates user customization of compiler output
  - Eliminates scripts, post-processing

6. Advanced features support

- General
  - Re-usable Templates
  - Sharing of objects and registers
  - Python API
- RTL
  - More complete protocols support
  - AddressMap-level interconnect (for ex: APB Target to Multi Initiator bridge)
  - Memories
  - Wide registers atomic access
  - Indirect access, Register Alias, Broadcast
  - Comprehensive Interrupt support
  - Parity checks
  - Full coverage Verification suite
- UVM
  - Coverage bins
  - Back door path mapping

CSRCompiler Technical Capabilities

1. High capacity & fast performance: compiles up to 5 Million Registers in minutes
2. True cross compiler reading many formats: IPXACT, SystemRDL, spreadsheets…
3. High-quality RTL: Extensive error checking
4. Powerful generator: RTL reg bank, UVM, C-HAL, Documentation…
5. Highly configurable: Over 400 configuration properties
6. Advanced features support: Atomic access, wide memories…

Benefits

Benefits: It's all about Time & Accuracy!

2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
MUNICH, GERMANY
OCTOBER 15-16, 2024

- Shorten time to market (TTM) with improved productivity
  - Generation in seconds - Not hours
  - Right the first time and correct by construction
  - Ease of adjusting specifications with very fast iteration updating documentation across all design teams
- Consistent and 100% accurate design information
  - Design, verification, software, and documentation integrated with a single-source specification
  - All involved teams rely on a correctly implemented HSI ensuring a robust product
  - All the outputs are autogenerated, no need for additional custom scripting or manual post-processing
- Flexibility to design large and complex designs
  - Highly scalable
  - Highly configurable
  - Fast performance

CSRCompiler Key Benefits

For RTL Architects

Making your life easier: simple design capture and standardized, self-checking authoring; clean, standard RTL; 100% accurate.

For Software Developers

Benefits include: consistent documentation; single-source spec that generates both RTL and Header files; early software access.

For Verification Engineers

Take advantage of: consistent documentation; a single-source spec that generates the SystemVerilog testbench register model and register information.

For Tech Pubs

Always 100% accurate: 100% accurate documentation auto-generated in Word or FrameMaker; generate multiple document views and/or formats from the same source.

2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
MUNICH, GERMANY
OCTOBER 15-16, 2024

Tutorial / Demo

Agenda

2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
MUNICH, GERMANY
OCTOBER 15-16, 2024

1. **What is a CSR register**
2. CSR compiler : CSR Spec, imports and exports
3. Demonstration

Controlling a hardware module through a CSR register

enable, mode, interrupt-enable are Hardware Control signals that SW writes in order to enable specific hardware modes / functions

SW may like also to Read the register for full visibility

```c
#include <stdint.h>

// Define the Control/Status Register
typedef struct {
  uint8_t enable: 1;     // Bit 0: Enable bit
  uint8_t mode: 2;       // Bit 1-2: Mode selection
  uint8_t interrupt: 1;  // Bit 3: Interrupt enable bit
  uint8_t reserved: 4;   // Bit 4-7: Reserved
} CSR;

// Create an instance of the Control/Status Register
volatile CSR* controlStatusReg = (CSR*)0x12345678;

int main() {
  // Enable the chip
  controlStatusReg->enable = 1;

  // Set the mode to 2 (example value)
  controlStatusReg->mode = 2;

  // Enable interrupts
  controlStatusReg->interrupt = 1;

  // Check the status of the register
  if (controlStatusReg->enable) {
    // Chip is enabled
  }

  // Perform other operations...

  return 0;
}
```

CSR Register documentation – More complex Example

5.1.2. Statistics Counters (Dword Offset 0x18 – 0x38)

The following table describes the read-only registers that collect the statistics on the transmit and receive datapaths. A hardware reset clears these registers; a software reset also clears these registers except aMacID. The statistics counters roll up when the counter is full.

The register description uses the following definitions:

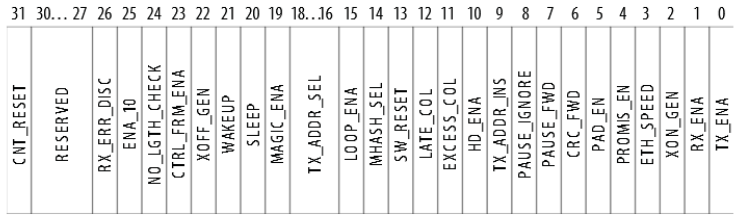
- Good frame—error-free frames with valid frame length.
- Error frame—frames that contain errors or whose length is invalid.
- Invalid frame—frames that are not addressed to the MAC function. The MAC function drops this frame.

Table 40. Statistics Counters

Dword Offset	Name	R/W	Description
0x18 – 0x19	aMacID	RO	The MAC address. This register is wired to the primary MAC address in the mac_0 and mac_1 registers.
0x1A	aFramesTransmittedOK	RO	The number of frames that are successfully transmitted including the pause frames.
0x1B	aFramesReceivedOK	RO	The number of frames that are successfully received including the pause frames.
0x1C	aFrameCheckSequenceErrors	RO	The number of receive frames with CRC error.
0x1D	aAlignmentErrors	RO	The number of receive frames with alignment error.

5.1.1.1. Command_Config Register (Dword Offset 0x02)

Figure 44. Command_Config Register Fields



At the minimum, you must configure the TX_ENA and RX_ENA bits to 1 to start the MAC operations. When configuring the command_config register, Intel® recommends that you configure the TX_ENA and RX_ENA bits the last because the MAC immediately starts its operations once these bits are set to 1.

Table 39. Command_Config Register Field Descriptions

Bit(s)	Name	R/W	Description	HW Reset
0	TX_ENA	RW	Transmit enable. Set this bit to 1 to enable the transmit datapath. The MAC clears this bit following a hardware or software reset. See the SW_RESET bit description.	0
1	RX_ENA	RW	Receive enable. Set this bit to 1 to enable the receive datapath. The MAC clears this bit following a hardware or software reset. See the SW_RESET bit description.	0
2	XON_GEN	RW	Pause frame generation. When you set this bit to 1, the MAC generates a pause frame with a pause quanta of 0, independent of the status of the receive FIFO buffer.	0
3	ETH_SPEED	RW	Ethernet speed control. • Set this bit to 1 to enable gigabit Ethernet operation. The set_1000 signal is masked and does not affect the operation. • If you set this bit to 0, gigabit Ethernet operation is enabled only	0

https://www.intel.com/content/www/us/en/docs/programmable/683402/22-1-20-0-0/command-config-register-dword-offset-0x02.html

Key properties of Registers ?

- From previous slides;
  What are some key properties of registers ?

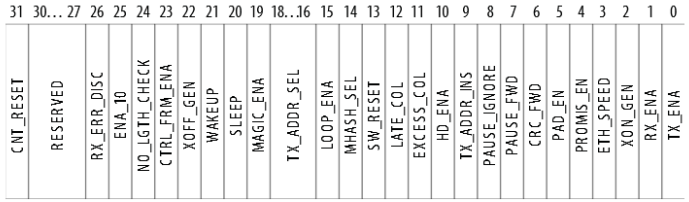
Key properties of Registers ?

- From previous slides;
  What are some key properties of registers ?

- Name
- Description
- Address
- Field layout
- Reset Value
- Access Types
  (RO, RW, WO, WRC, WRS, WC, WS, WSRC, WCRS, W1C, W1S, W1T, W0C, W0S, W0T, W1SRC, W1CRS, W0SRC)

5.1.1.1. Command_Config Register (Dword Offset 0x02)

Figure 44. Command_Config Register Fields



At the minimum, you must configure the TX_ENA and RX_ENA bits to 1 to start the MAC operations. When configuring the command_config register, Intel® recommends that you configure the TX_ENA and RX_ENA bits the last because the MAC immediately starts its operations once these bits are set to 1.

Table 39. Command_Config Register Field Descriptions

Bit(s)	Name	R/W	Description	HW Reset
0	TX_ENA	RW	Transmit enable. Set this bit to 1 to enable the transmit datapath. The MAC clears this bit following a hardware or software reset. See the SW_RESET bit description.	0
1	RX_ENA	RW	Receive enable. Set this bit to 1 to enable the receive datapath. The MAC clears this bit following a hardware or software reset. See the SW_RESET bit description.	0
2	XON_GEN	RW	Pause frame generation. When you set this bit to 1, the MAC generates a pause frame with a pause quanta of 0, independent of the status of the receive FIFO buffer.	0
3	ETH_SPEED	RW	Ethernet speed control. • Set this bit to 1 to enable gigabit Ethernet operation. The set_1000 signal is masked and does not affect the operation. • If you set this bit to 0, gigabit Ethernet operation is enabled only	0

Scope within a chip

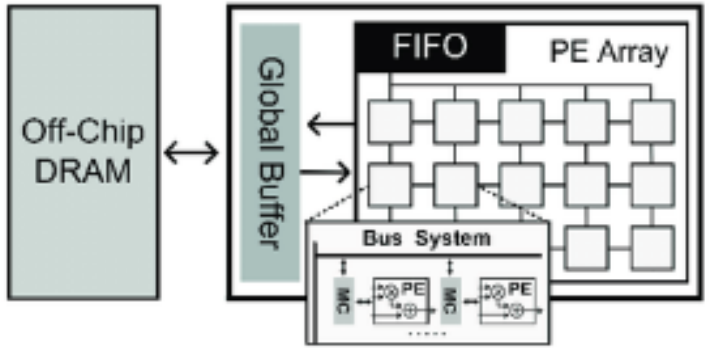
- We may have some SoC with very high count of CSR Registers, due to duplication of computing nodes

  (think FPGAs, or arrays of Process Elements for accelerators)

- In general, the CSR Registers are a small part of the chip circuitry
  < 10 % of the SoC Register

- Do not confuse with General Purpose Registers, which are registers used by the CPU in Assembly language for manipulating micro-operations

=> *So why bother generating them automatically ?*



Syncing Register information

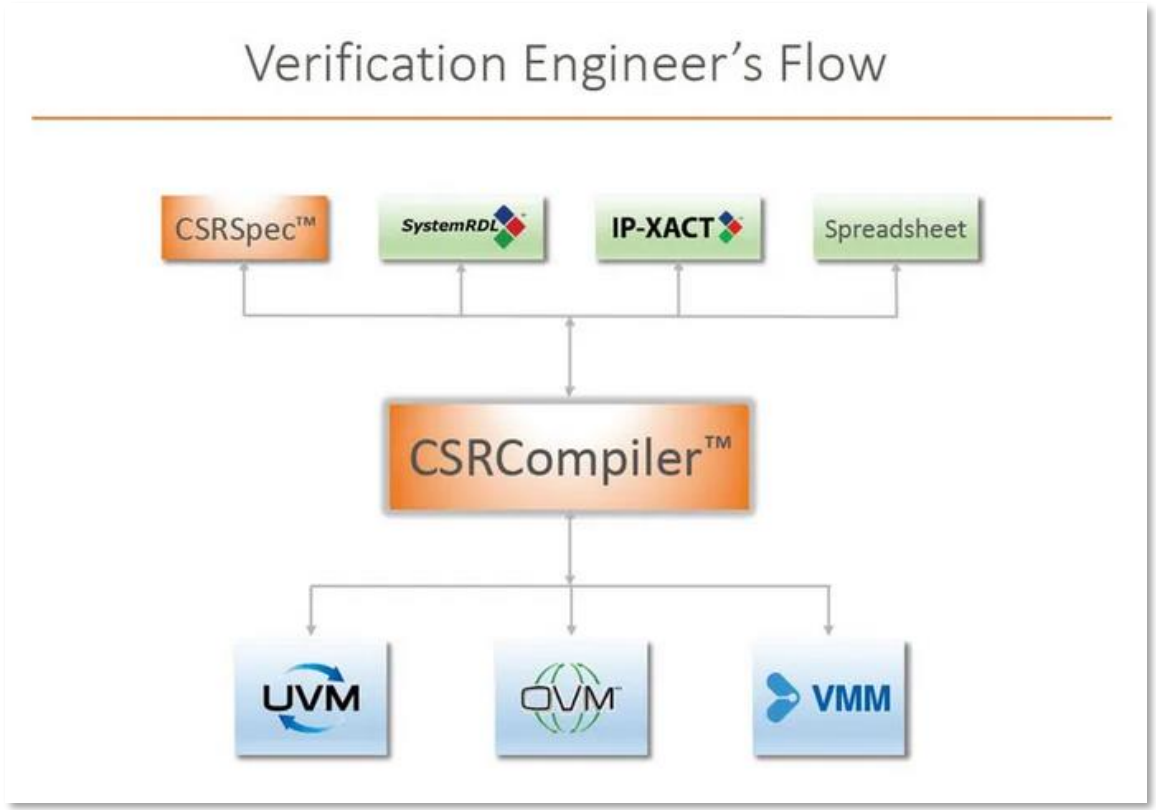
- The importance of CSR Registers comes from their crossroad position, Hardware – Software Interface
- Designers: Code the RTL for implementing the register in HW
- Verification engineers: Need a behavioral model to verify the HW
- Firmware software developers: Need
  - C Header files containing the memory map for low-level drivers
  - CMSIS file for debug
- Documentation is required by everybody
  - Design + Verif
  - Firmware Software
  - Customer using the chip

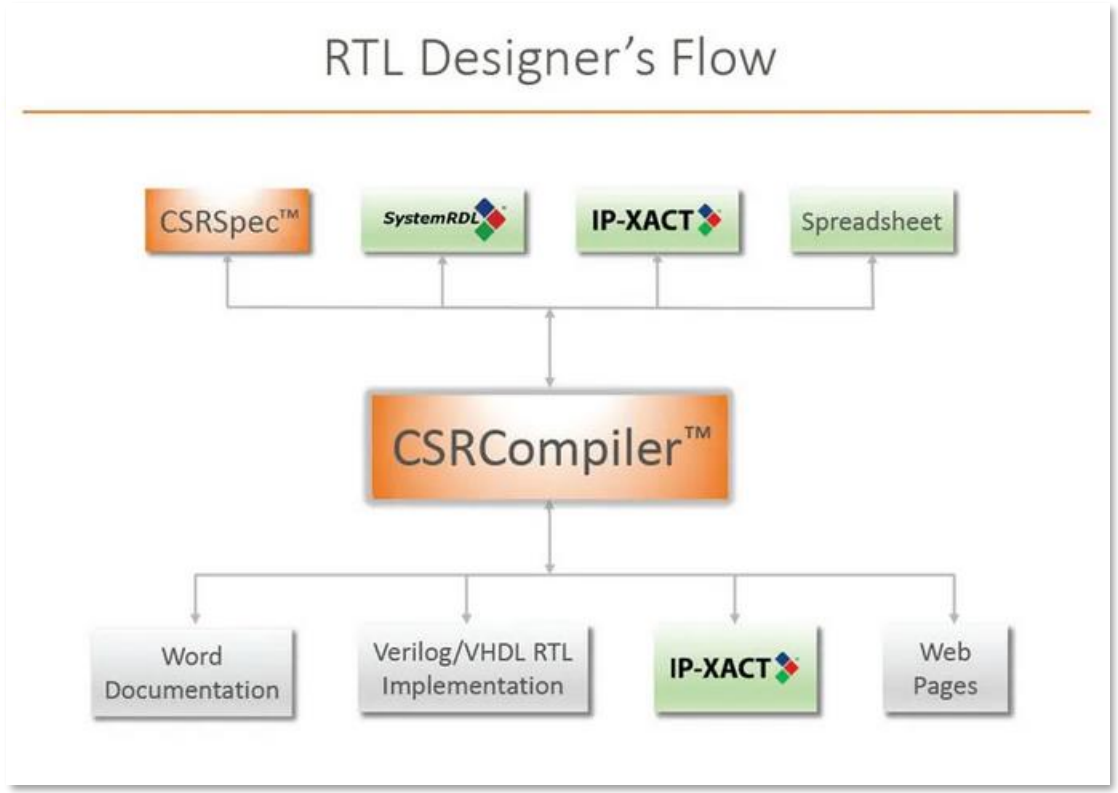
Syncing Register information

- Keeping all the team in-sync
  - Sync the team at earlier project phases
  - Avoids critical bugs due to bad communication
  - Avoids ‘exotic’ usages
- Limiting the supported CSR Register configurations
  - Reduces the number of cases -> reduces R&D effort

⇒Requirement for a tool centralizing Register information

CSR Compiler flows

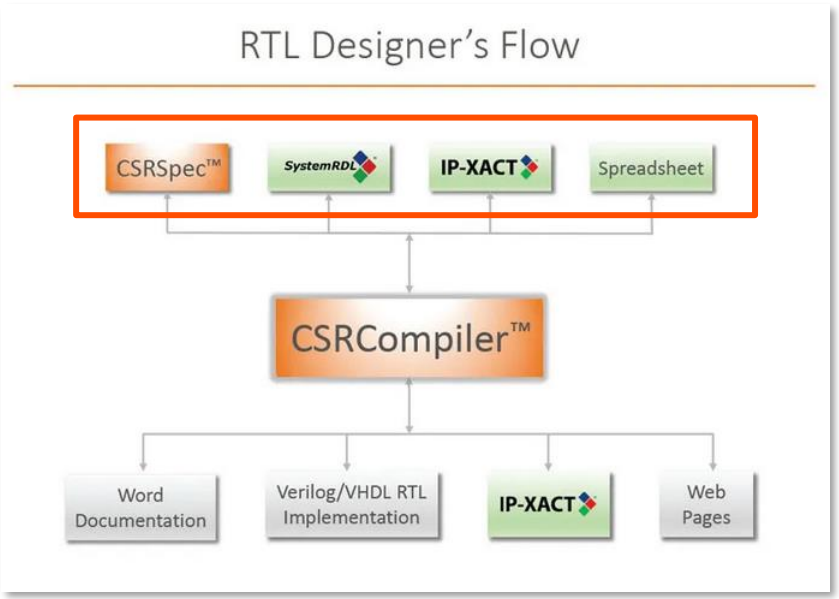




CSR Compiler - Input formats

- Spreadsheet
  - Practical, not standard, requires customization and manual effort
- IP-XACT
  - Standard, but focuses on HSI integration, and not on HW implementation
  - In MRV this is compensated with MRV Parameters and Mustache Templating
- System RDL
  - Dedicated standard for registers
  - Not complete and mature
- CSR Spec
  - Market defacto standard (but not official)
  - 200 unique properties and 6,000 register behavior combination :
  - Much more complete and mature when compared to System RDL with 15+ years of development

```
<spirit:parameters>
  <spirit:parameter>
    <spirit:name>_hwAccess_</spirit:name>
    <spirit:value>RO</spirit:value>
  </spirit:parameter>
</spirit:parameters>
```



CSR Spec Example

```
addressmap {
   property title = "Demo Addressmap";
   property description =
      "This addressmap contains several types"
      " of registers and fields.";
```

	A	B	C	D	E	F	G	H	I	J	K
1	**Offset**	**Position**	**Title**	**Identifier**	**Array**	**Access**	**Type**	**Reset Value**	**Reset Mask**	**Description**	
2			Demo Addressmap	demo		R/W	addressmap			This addressmap contains several types of registers and fields.	
3	0x0		Chip ID	chip_id		R	register	0x11234469	0xffffffff	This register contains the chip ID.	
4		[0]		one		R	constant	0x1			
5		[11:1]	Manufacturer ID	manufacturer_id		R	constant	0x234			
6		[27:12]	Part Number	part_number		R	constant	0x1234			
7		[31:28]	Version	version		R	constant	0x1			
8	0x4		Configuration Registe	chip_configuration		R/W	register	0x00000000	0xffffffff	This register has configuration bits.	
9		[1:0]		chip_mode		R/W	configuration	0x0			
10		[31]		chip_enable		R/W	configuration	0x0			
11	0x8		Status Register	chip_status		R	register	0x00000000	0xFD	This register has status bits.	
12		[0]		port_active		R	status				
13	0xc		Data Counter Registe	chip_data_count		R/W	register			This register has a counter.	
14		[31:0]		bytes_received		R/W	counter				
15											

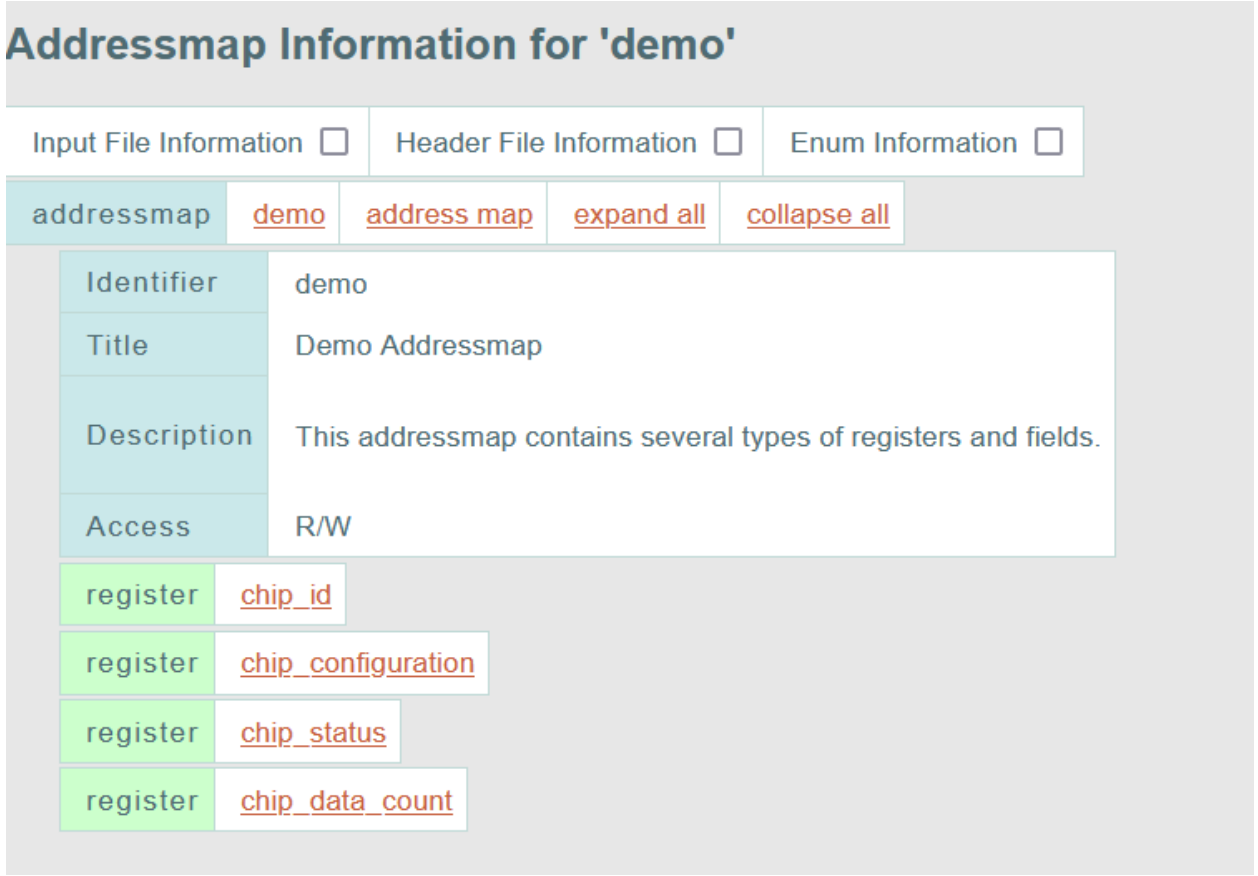
CSR Spec Example

```
register {
    property title = "Chip ID";
    property description = "This register contains the chip ID."
    field {
        property title = "Version";
        property field_type = "constant";
        property output_port = true;
        property reset_value = 0x1;
    } [31:28] version;
    field {
        property title = "Part Number";
        property field_type = "constant";
        property output_port = true;
        property reset_value = 0x1234;
    } [27:12] part_number;
    field {
        property {
            title = "Manufacturer ID";
            field_type = "constant";
            output_port = true;
            reset_value = 0x234;
        };
    } [11:1] manufacturer_id;
    field {
        property field_type = "constant";
        property output_port = true;
        property reset_value = 0x1;
    } [0] one;
} chip_id;
```

```
register {
    property title = "Configuration Register";
    property description = "This register has configuration bits.";
    field {
        property field_type = "configuration";
        property output_port = true;
        property reset_value = NORMAL;
    } [1:0] chip_mode;
    field {
        property field_type = "configuration";
        property output_port = true;
        property reset_value = 0x0;
    } [31] chip_enable;
} chip_configuration;

register {
    property title = "Status Register";
    property description = "This register has status bits.";
    field {
        property field_type = "status";
    } [0] port_active;
} chip_status;

register {
    property title = "Data Counter Register";
    property description = "This register has a counter.";
    field {
        property field_type = "counter";
    } [31:0] bytes_received;
} chip_data_count;

} demo;
```

CSR Compiler HTML Export

> csrCompile demo.csr -t html



- In what address offset is ‘chip status’ ?
- What is the manufacturer ID of this chip ?

CSR Compiler C Header

> csrCompile demo.csr -t h

```
/* ########################################################################## */
/*        TEMPLATE MACROS                                                     */
/* ########################################################################## */

/* Register type: demo::chip_configuration                                    */
/* Register template: demo::chip_configuration                                */
/* Source filename: demo.csr, line: 87                                        */
/* Field member: demo::chip_configuration.chip_enable                         */
/* Source filename: demo.csr, line: 96                                        */
#define DEMO_CHIP_CONFIGURATION_CHIP_ENABLE_MSB 31u
#define DEMO_CHIP_CONFIGURATION_CHIP_ENABLE_LSB 31u
#define DEMO_CHIP_CONFIGURATION_CHIP_ENABLE_WIDTH 1u
#define DEMO_CHIP_CONFIGURATION_CHIP_ENABLE_READ_ACCESS 1u
#define DEMO_CHIP_CONFIGURATION_CHIP_ENABLE_WRITE_ACCESS 1u
#define DEMO_CHIP_CONFIGURATION_CHIP_ENABLE_RESET 0x0u
#define DEMO_CHIP_CONFIGURATION_CHIP_ENABLE_FIELD_MASK 0x80000000ul
#define DEMO_CHIP_CONFIGURATION_CHIP_ENABLE_GET(x) \
   (((x) & 0x80000000ul) >> 31)
#define DEMO_CHIP_CONFIGURATION_CHIP_ENABLE_SET(x) \
   (((x) << 31) & 0x80000000ul)
#define DEMO_CHIP_CONFIGURATION_CHIP_ENABLE_MODIFY(r, x) \
   ((((x) << 31) & 0x80000000ul) | ((r) & 0x7fffffful))
/* Field member: demo::chip_configuration.chip_mode                           */
/* Source filename: demo.csr, line: 90                                        */
#define DEMO_CHIP_CONFIGURATION_CHIP_MODE_MSB 1u
#define DEMO_CHIP_CONFIGURATION_CHIP_MODE_LSB 0u
#define DEMO_CHIP_CONFIGURATION_CHIP_MODE_WIDTH 2u
#define DEMO_CHIP_CONFIGURATION_CHIP_MODE_READ_ACCESS 1u
#define DEMO_CHIP_CONFIGURATION_CHIP_MODE_WRITE_ACCESS 1u
#define DEMO_CHIP_CONFIGURATION_CHIP_MODE_RESET 0x0u
#define DEMO_CHIP_CONFIGURATION_CHIP_MODE_FIELD_MASK 0x00000003ul
#define DEMO_CHIP_CONFIGURATION_CHIP_MODE_GET(x) ((x) & 0x00000003ul)
#define DEMO_CHIP_CONFIGURATION_CHIP_MODE_SET(x) ((x) & 0x00000003ul)
#define DEMO_CHIP_CONFIGURATION_CHIP_MODE_MODIFY(r, x) \
   (((x) & 0x00000003ul) | ((r) & 0xfffffffcul))
```

```
/* ########################################################################## */
/*        ADDRESS MACROS                                                      */
/* ########################################################################## */

/* Address Space for Addressmap: demo                                         */
/* Source filename: demo.csr, line: 119                                       */
/* Register: demo.chip_id                                                     */
#define DEMO_CHIP_ID_ADDRESS 0x0u
#define DEMO_CHIP_ID_BYTE_ADDRESS 0x0u
/* Register: demo.chip_configuration                                          */
#define DEMO_CHIP_CONFIGURATION_ADDRESS 0x4u
#define DEMO_CHIP_CONFIGURATION_BYTE_ADDRESS 0x4u
/* Register: demo.chip_status                                                 */
#define DEMO_CHIP_STATUS_ADDRESS 0x8u
#define DEMO_CHIP_STATUS_BYTE_ADDRESS 0x8u
/* Register: demo.chip_data_count                                             */
#define DEMO_CHIP_DATA_COUNT_ADDRESS 0xcu
#define DEMO_CHIP_DATA_COUNT_BYTE_ADDRESS 0xcu
```

```
/* ########################################################################## */
/*        TYPE DEFINITIONS                                                    */
/* ########################################################################## */

/* Typedef for Addressmap: demo                                               */
/* Source filename: demo.csr, line: 119                                       */
typedef struct {
   uint32_t chip_id; /**< Offset 0x0 (R) */
   volatile uint32_t chip_configuration; /**< Offset 0x4 (R/W) */
   volatile uint32_t chip_status; /**< Offset 0x8 (R) */
   volatile uint32_t chip_data_count; /**< Offset 0xc (R/W) */
} Demo, *PTR_Demo;
```

CSR Compiler Verilog RTL

> csrCompile demo.csr -t v

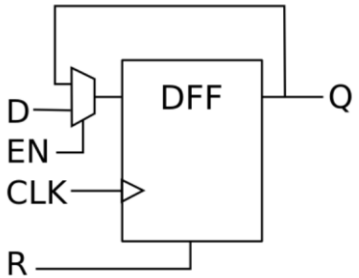
```
assign csr_internal_reset_value_chip_configuration_chip_enable =
  1'h0;
```

reset value

- Why is reset not in the sensitivity list ?

```
always @(posedge clock)
  if (reset)
    csr_internal_field_chip_configuration_chip_enable <=
      csr_internal_reset_value_chip_configuration_chip_enable;
  else
    csr_internal_field_chip_configuration_chip_enable <=
      csr_internal_next_field_chip_configuration_chip_enable;
```

Flip Flop




```
assign csr_internal_next_field_chip_configuration_chip_enable =
  (csr_internal_write_access_chip_configuration_chip_enable) ?
    csr_internal_bus_write_data[31]:
    csr_internal_field_chip_configuration_chip_enable;
```

Mux : (write data) or (keep data)

ARTERIS CONFIDENTIAL

CSR Compiler UVM RAL

> csrCompile demo.csr -t uvm

Register class

```
// Register: demo::chip_data_count
// Source filename: demo.csr, line: 111
class csr_reg_demo_chip_data_count extends uvm_reg;
  rand uvm_reg_field bytes_received;

  function new (string name = "csr_reg_demo_chip_data_count");
    super.new(name, 32, UVM_NO_COVERAGE);
  endfunction: new

  virtual function void build ();
    this.bytes_received = uvm_reg_field::type_id::create(
      "bytes_received", null, get_full_name());
    this.bytes_received.configure(this, 32, 0, "RW",
      1, 0, 0, 1, 0);
  endfunction: build

  `uvm_object_utils(csr_reg_demo_chip_data_count)

endclass : csr_reg_demo_chip_data_count
```

No coverage in this example

Register block

Register configuration in the block

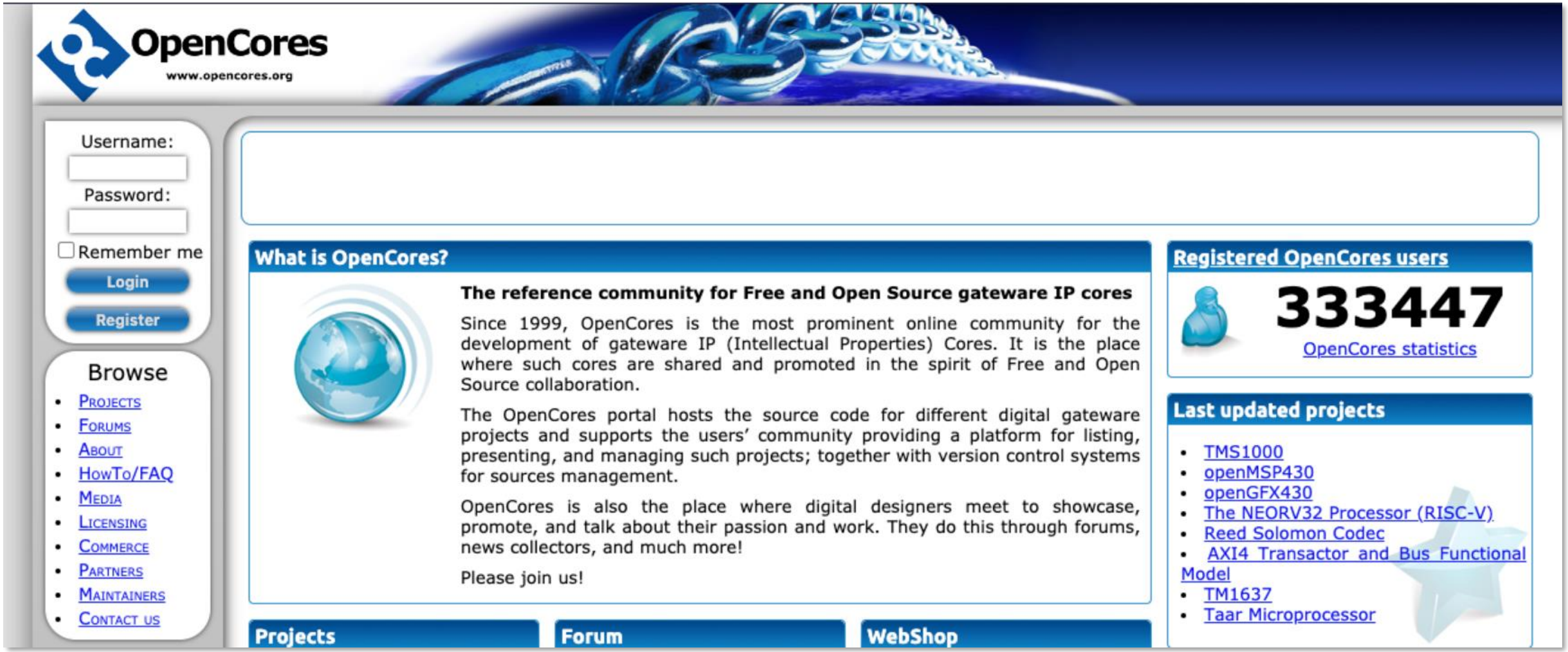
Backdoor paths

```
class csr_block_demo extends uvm_reg_block;
  rand csr_reg_demo_chip_id chip_id;
  rand csr_reg_demo_chip_configuration chip_configuration;
  rand csr_reg_demo_chip_status chip_status;
  rand csr_reg_demo_chip_data_count chip_data_count;

  function new (string name = "csr_block_demo");
    super.new(name, UVM_NO_COVERAGE);
  endfunction: new

  virtual function void build ();
    this.default_map = create_map(
      "csr_reg_map_demo", 0, 4, UVM_LITTLE_ENDIAN, 1);
    this.chip_id =
      csr_reg_demo_chip_id::type_id::create(
        "chip_id", null, get_full_name());
    this.chip_id.configure(this, null);
    this.chip_id.build();
    this.default_map.add_reg(chip_id,
      `UVM_REG_ADDR_WIDTH'h0, "RO");
    this.chip_configuration =
      csr_reg_demo_chip_configuration::type_id::create(
        "chip_configuration", null, get_full_name());
    this.chip_configuration.configure(this, null);
    this.chip_configuration.build();
    this.default_map.add_reg(chip_configuration,
      `UVM_REG_ADDR_WIDTH'h4, "RW");
    this.chip_status =
      csr_reg_demo_chip_status::type_id::create(
        "chip_status", null, get_full_name());
    this.chip_status.configure(this, null);
    this.chip_status.build();
    this.default_map.add_reg(chip_status,
      `UVM_REG_ADDR_WIDTH'h8, "RO");
    this.chip_data_count =
      csr_reg_demo_chip_data_count::type_id::create(
        "chip_data_count", null, get_full_name());
    this.chip_data_count.configure(this, null);
    this.chip_data_count.build();
    this.default_map.add_reg(chip_data_count,
      `UVM_REG_ADDR_WIDTH'hc, "RW");

    // Backdoor paths
    this.chip_id.add_hdl_path_slice("csr_internal_field_chip_id_one", 0, 1, 1);
    this.chip_id.add_hdl_path_slice("csr_internal_field_chip_id_manufacturer_id", 1, 11, 0);
    this.chip_id.add_hdl_path_slice("csr_internal_field_chip_id_part_number", 12, 16, 0);
    this.chip_id.add_hdl_path_slice("csr_internal_field_chip_id_version", 28, 4, 0);
    this.chip_configuration.add_hdl_path_slice("csr_internal_field_chip_configuration_chip_mode", 0, 2, 1);
    this.chip_configuration.add_hdl_path_slice("csr_internal_field_chip_configuration_chip_enable", 31, 1, 0);
    this.chip_status.add_hdl_path_slice("csr_internal_field_chip_status_port_active", 0, 1, 1);
    this.chip_data_count.add_hdl_path_slice("csr_internal_field_chip_data_count_bytes_received", 0, 32, 1);
  endfunction: build

  `uvm_object_utils(csr_block_demo)

endclass : csr_block_demo
```

2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
MUNICH, GERMANY
OCTOBER 15-16, 2024

CSR Compiler Demo

Ethernet MAC 10/100 Mbps



Too Much Code To Scroll Through

```
register {
  property title = "MII Mode Register";
  property register_reset_value = 0x00000064;
  property register_reset_mask = 0x000001FF;

  field {
    property title = "No Preamble";
    property description =
      "0 = 32-bit preamble sent\n"
      "1 = No preamble send";
    property output_port;
  } [8] MIINOPRE;

  field {
    property title = "Clock Divider";
    property description =
      "The field is a host clock divider factor."
      " The host clock can be divided by an even number,"
      " greater then 1. The default value is 0x64 (100)";
    property output_port;
  } [7:0] CLKDIV;
} [0x28] MIIMODER;
```

Example: MII Mode Register

```
register {
  property title = "Interrupt Source Register";
  property register_reset_value = 0x00000000;
  property register_reset_mask = 0x0000007F;
  property register_interrupt_output_port;

  field {
    property title = "Receive Control Frame";
    property description =
      "This bit indicates that the control frame was received."
      " It is cleared by writing 1 to it. Bit RXFLOW"
      " (CTRLMODER register) must be set to 1 in order to get"
      " the RXC bit set.";
    property field_type = "interrupt";
    property write_function = "one_clear";
    property interrupt_enable = field(INT_MASK.RXC_M);
  } [6] RXC;

...// Some Fields Not Shown

} [0x4] INT_SOURCE;
```

Example: Interrupt Field

Running the CSRCompiler

SystemVerilog RTL Output

```
output wire [31:0] csr_read_data,
input  wire csr_write_access,
input  wire csr_read_access,
input  wire [6:0] csr_address,
input  wire [31:0] csr_write_data,
input  wire reset,
input  wire clock
```

CSRSpec
Basic Bus Ports

CSRCompiler
Supported Protocols

AMBA 2 APB
AMBA 3 APB
AMBA 4 APB
AMBA 3 AHB-Lite
AMBA 4 AXI-Lite
AMBA 4 AXI
Avalon 3.2
OCP 2.1
Wishbone

Output Ports

```
output wire MIIMODER_MIINOPRE,
output wire [7:0] MIIMODER_CLKDIV,
```

RTL: MII Mode

Read Logic

```
assign csr_internal_read_value_MIIMODER =
    {
      23'h0,
      csr_internal_field_MIIMODER_MIINOPRE,
      csr_internal_field_MIIMODER_CLKDIV
    };
 assign csr_internal_read_bus_MIIMODER =
   csr_internal_read_value_MIIMODER &
   {32{csr_internal_decode_MIIMODER}};
```

```
assign csr_internal_write_access_MIIMODER_MIINOPRE =
  csr_internal_decode_MIIMODER &
  csr_internal_write_access;

assign csr_internal_next_field_MIIMODER_MIINOPRE =
  (csr_internal_write_access_MIIMODER_MIINOPRE) ?
    csr_internal_bus_write_data[8]:
    csr_internal_field_MIIMODER_MIINOPRE;

assign csr_internal_reset_value_MIIMODER_MIINOPRE =
      1'h0;

always @(posedge clock)
  if (reset)
    csr_internal_field_MIIMODER_MIINOPRE <=
      csr_internal_reset_value_MIIMODER_MIINOPRE;
  else
    csr_internal_field_MIIMODER_MIINOPRE <=
      csr_internal_next_field_MIIMODER_MIINOPRE;

assign MIIMODER_MIINOPRE =
  csr_internal_field_MIIMODER_MIINOPRE;
```

RTL: MIINOPRE Field

Interrupt Source

```
input wire INT_SOURCE_RXC_source,
```

```
assign csr_internal_write_access_INT_SOURCE_RXC =
  csr_internal_decode_INT_SOURCE &
  csr_internal_write_access;

assign csr_internal_source_INT_SOURCE_RXC =
  INT_SOURCE_RXC_source;
assign csr_internal_event_INT_SOURCE_RXC =
  csr_internal_source_INT_SOURCE_RXC;

assign csr_internal_interrupt_INT_SOURCE_RXC =
  (
    csr_internal_field_INT_SOURCE_RXC &
    csr_internal_field_INT_MASK_RXC_M
  );
```

RTL:
RXC Interrupt

Interrupt Logic

```
assign csr_internal_next_field_INT_SOURCE_RXC =
  (csr_internal_write_access_INT_SOURCE_RXC) ?
    (
      (~csr_internal_bus_write_data[6]) &
      csr_internal_pre_write_INT_SOURCE_RXC
    ):
    csr_internal_pre_write_INT_SOURCE_RXC;

assign csr_internal_pre_write_INT_SOURCE_RXC =
  (
    csr_internal_event_INT_SOURCE_RXC |
    csr_internal_field_INT_SOURCE_RXC
  );

assign csr_internal_reset_value_INT_SOURCE_RXC =
  1'h0;

always @(posedge clock)
  if (reset)
    csr_internal_field_INT_SOURCE_RXC <=
      csr_internal_reset_value_INT_SOURCE_RXC;
  else
    csr_internal_field_INT_SOURCE_RXC <=
      csr_internal_next_field_INT_SOURCE_RXC;
```

Continued RTL: RXC Interrupt

UVM Output

```
class csr_reg_ethmac_MIIMODER extends uvm_reg;
  rand uvm_reg_field CLKDIV;
  rand uvm_reg_field MIINOPRE;

  function new (string name = "csr_reg_ethmac_MIIMODER");
    super.new(name, 32, UVM_NO_COVERAGE);
  endfunction: new

  virtual function void build ();
    this.CLKDIV = uvm_reg_field::type_id::create(
      "CLKDIV", null, get_full_name());
    this.CLKDIV.configure(this, 8, 0, "RW",
      0, 8'h64, 1, 1, 0);
    this.MIINOPRE = uvm_reg_field::type_id::create(
      "MIINOPRE", null, get_full_name());
    this.MIINOPRE.configure(this, 1, 8, "RW",
      0, 1'h0, 1, 1, 0);
  endfunction: build

  `uvm_object_utils(csr_reg_ethmac_MIIMODER)

endclass : csr_reg_ethmac_MIIMODER
```

Fields

Build

MII Mode Definition

MII Mode Instance


```
class csr_block_ethmac extends uvm_reg_block;
  ...
  rand csr_reg_ethmac_MIIMODER MIIMODER;
  ...

  this.MIIMODER =
    csr_reg_ethmac_MIIMODER::type_id::create(
      "MIIMODER", null, get_full_name());
  this.MIIMODER.configure(this, null);
  this.MIIMODER.build();
  this.default_map.add_reg(MIIMODER,
    `UVM_REG_ADDR_WIDTH'h28, "RW");
  ...

  this.MIIMODER.add_hdl_path_slice("csr_internal_field_MIIMODER_CLKDIV", 0, 8, 1);
  this.MIIMODER.add_hdl_path_slice("csr_internal_field_MIIMODER_MIINOPRE", 8, 1, 0);
  ...

endclass : csr_block_ethmac
```

Declaration

Build

Backdoor Path

C Header Output

MII Mode Reg

Address Macros

```
#define ETHMAC_MIIMODER_ADDRESS 0x28u
#define ETHMAC_MIIMODER_BYTE_ADDRESS 0x28u
```

Template Macros

```
#define ETHMAC_MIIMODER_OFFSET 0x28u
#define ETHMAC_MIIMODER_BYTE_OFFSET 0x28u
#define ETHMAC_MIIMODER_READ_ACCESS 1u
#define ETHMAC_MIIMODER_WRITE_ACCESS 1u
#define ETHMAC_MIIMODER_RESET_VALUE 0x00000064ul
#define ETHMAC_MIIMODER_RESET_MASK 0x000001fful
#define ETHMAC_MIIMODER_READ_MASK 0xffffffful
#define ETHMAC_MIIMODER_WRITE_MASK 0x000001fful
```

MIINOPRE Field

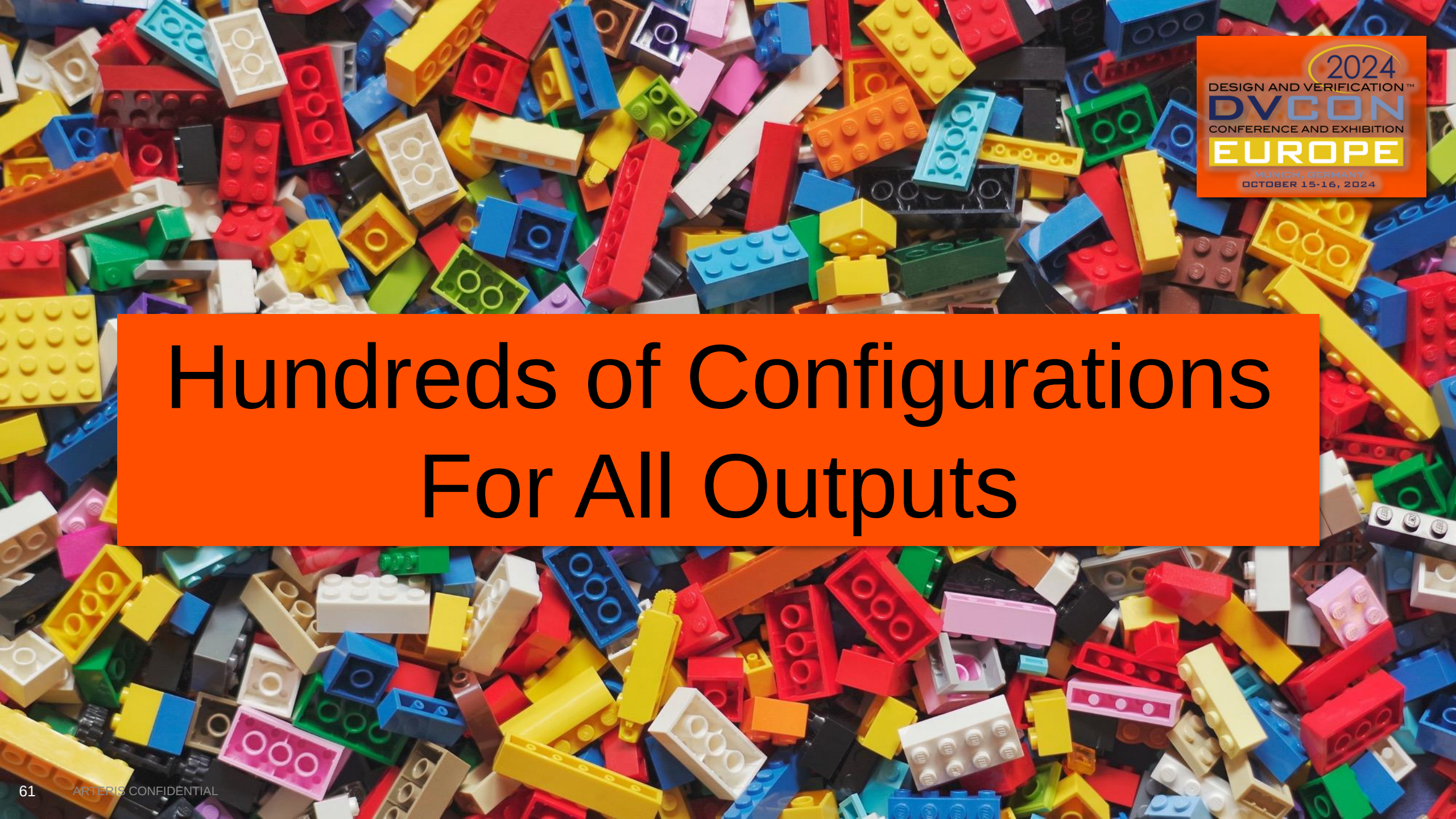
Template Macros

```
#define ETHMAC_MIIMODER_MIINOPRE_MSB 8u
#define ETHMAC_MIIMODER_MIINOPRE_LSB 8u
#define ETHMAC_MIIMODER_MIINOPRE_WIDTH 1u
#define ETHMAC_MIIMODER_MIINOPRE_READ_ACCESS 1u
#define ETHMAC_MIIMODER_MIINOPRE_WRITE_ACCESS 1u
#define ETHMAC_MIIMODER_MIINOPRE_RESET 0x0u
#define ETHMAC_MIIMODER_MIINOPRE_FIELD_MASK 0x00000100ul
#define ETHMAC_MIIMODER_MIINOPRE_GET(x) (((x) & 0x00000100ul) >> 8)
#define ETHMAC_MIIMODER_MIINOPRE_SET(x) (((x) << 8) & 0x00000100ul)
#define ETHMAC_MIIMODER_MIINOPRE_MODIFY(r, x) \
  ((((x) << 8) & 0x00000100ul) | ((r) & 0xfffffefful))
```

```
typedef struct {
  volatile uint32_t MODER; /**< Offset 0x0 (R/W) */
  volatile uint32_t INT_SOURCE; /**< Offset 0x4 (R/W) */
  volatile uint32_t INT_MASK; /**< Offset 0x8 (R/W) */
  volatile uint32_t IPGT; /**< Offset 0xc (R/W) */
  volatile uint32_t IPGR1; /**< Offset 0x10 (R/W) */
  volatile uint32_t IPGR2; /**< Offset 0x14 (R/W) */
  volatile uint32_t PACKETLEN; /**< Offset 0x18 (R/W) */
  volatile uint32_t COLLCONF; /**< Offset 0x1c (R/W) */
  volatile uint32_t TX_BD_NUM; /**< Offset 0x20 (R/W) */
  volatile uint32_t CTRLMODER; /**< Offset 0x24 (R/W) */
  volatile uint32_t MIIMODER; /**< Offset 0x28 (R/W) */
  volatile uint32_t MIICOMMAND; /**< Offset 0x2c (R/W) */
  volatile uint32_t MIIADDRESS; /**< Offset 0x30 (R/W) */
  volatile uint32_t MIITX_DATA; /**< Offset 0x34 (R/W) */
  volatile uint32_t MIIRX_DATA; /**< Offset 0x38 (R) */
  volatile uint32_t MIISTATUS; /**< Offset 0x3c (R) */
  volatile uint32_t MAC_ADDR0; /**< Offset 0x40 (R/W) */
  volatile uint32_t MAX_ADDR1; /**< Offset 0x44 (R/W) */
  volatile uint32_t ETH_HASH0_ADR; /**< Offset 0x48 (R/W) */
  volatile uint32_t ETH_HASH1_ADR; /**< Offset 0x4c (R/W) */
  volatile uint32_t ETH_TXCTRL; /**< Offset 0x50 (R/W) */
  uint8_t _pad0[0x2c];
} Ethmac, *PTR_Ethmac;
```

Struct of the Addressmap

HTML OUTPUT

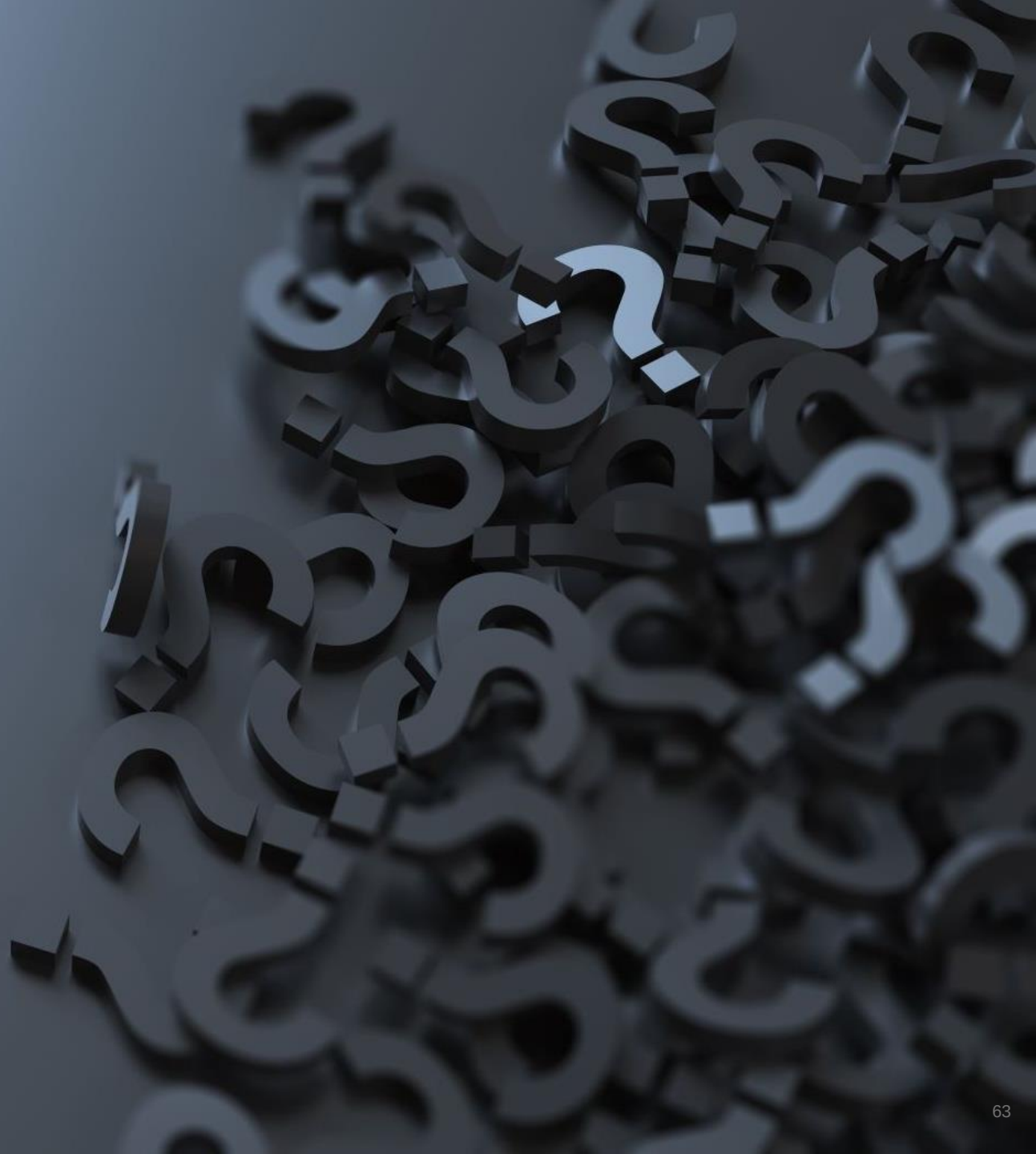


Hundreds of Configurations For All Outputs

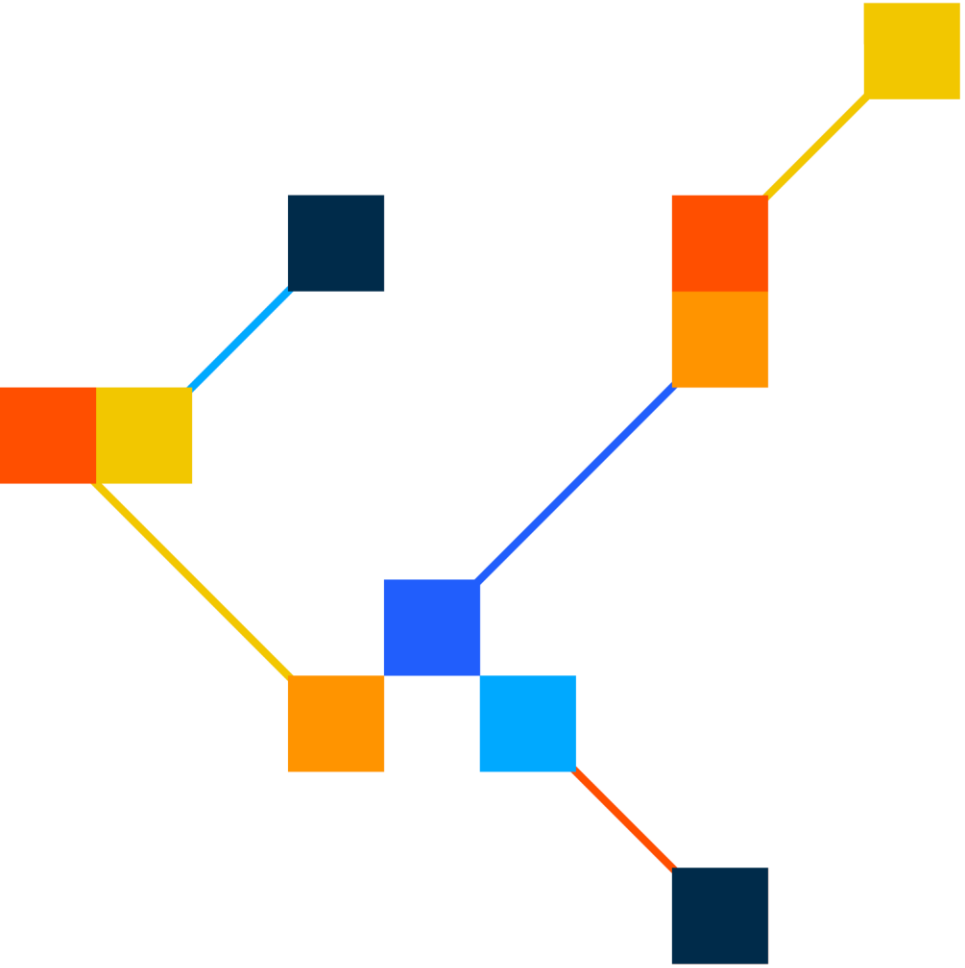
Lets Run a Bigger CSRSpec

2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
MUNICH, GERMANY
OCTOBER 15-16, 2024

Q&A



2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
MUNICH, GERMANY
OCTOBER 15-16, 2024



ARTERIS IP

Thank you