# AKER: A Design and Verification Framework for Safe and Secure SoC Access Control


Francesco Restuccia, Andres Meza, and Ryan Kastner

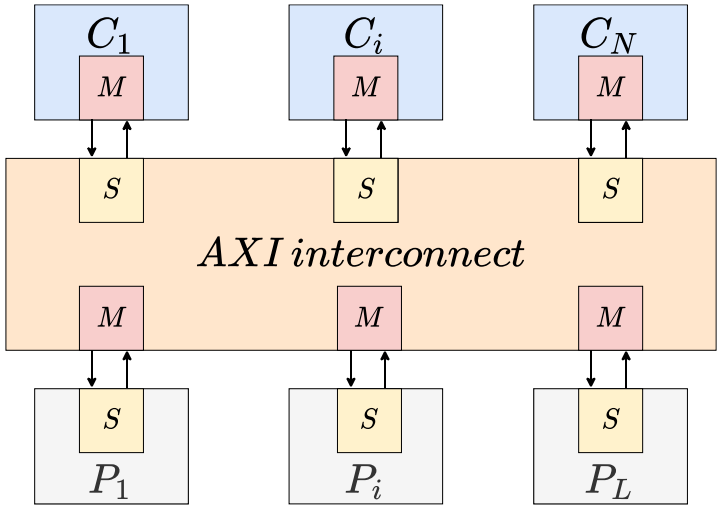
Dept. of Computer Science and Engineering

UC San Diego

http://kastner.ucsd.edu

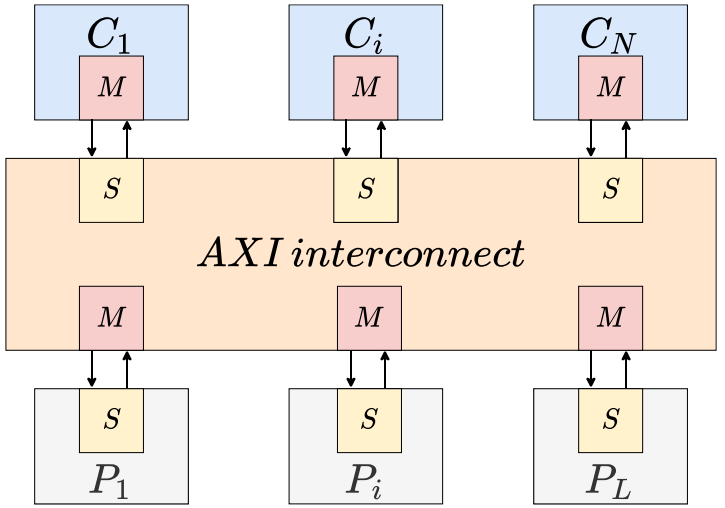
# Motivations

- Generic multi-manager, multi-subordinate architecture
- C -> Controllers, able to initiate requests (Processors, Hardware accelerators, etc)
- P -> Shared peripherals, serving requests (I/O,  DRAM, ROM, etc )
- Popular architecture in modern SoCs



# Motivations

- Each controller (C) accesses a subset of the resources (P)
- Different levels of trustworthiness, criticality, privacy, …
- Access control changes over the SoC lifetime




**Access control must be flexible, fast, and efficient**

**Verified to work correctly and securely**

# The AKER framework: overview

**The Access Control Wrapper (ACW)**

Universal hardware module

Easy-to-integrate

Open-source

Flexible: reconfigurable policies

Secure configuration: integrated with SoA open HRoT (openTitan)

Extensive Security Verification

Property-driven

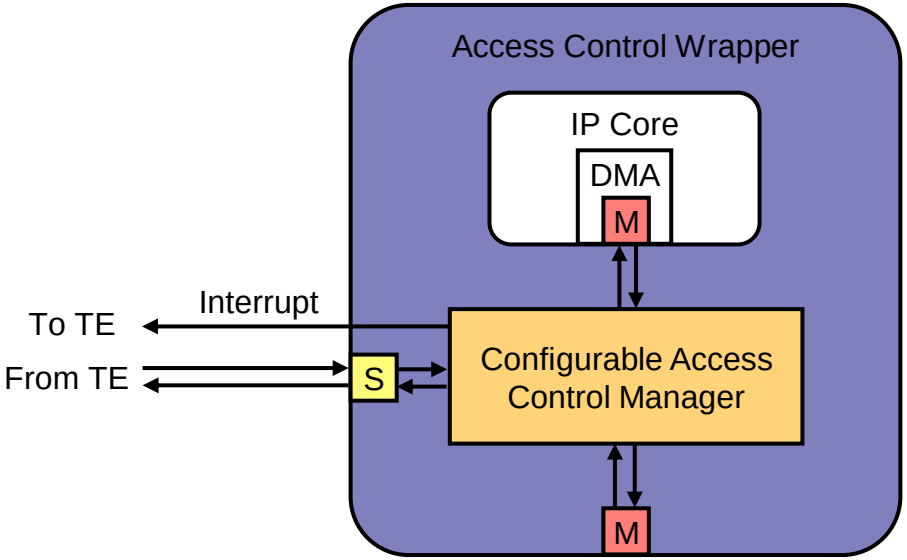
Multi-level verification process

Evaluates the MITRE's common weaknesses (CWEs)

Easily extendable (e.g., for the requirements of industrial designs)

# Access Control Wrapper (ACW): Overview

- Universal wrapper for SoC C cores (IP core)
- Managed by a trusted entity (HRoT, trusted processor, etc)
- Runtime monitoring of reads/writes
  - * Allows all legal requests
  - * Stops any illegal requests
- Interrupt on any illegal request
- Logs data about illegal request

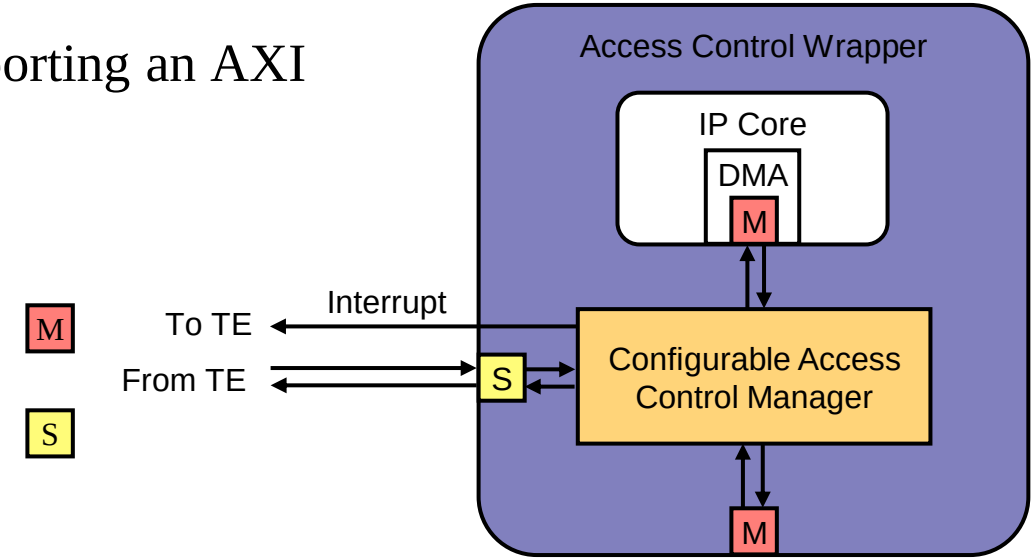


# Access Control Wrapper: Interface

Compatible with any module exporting an AXI manager interface

External interface

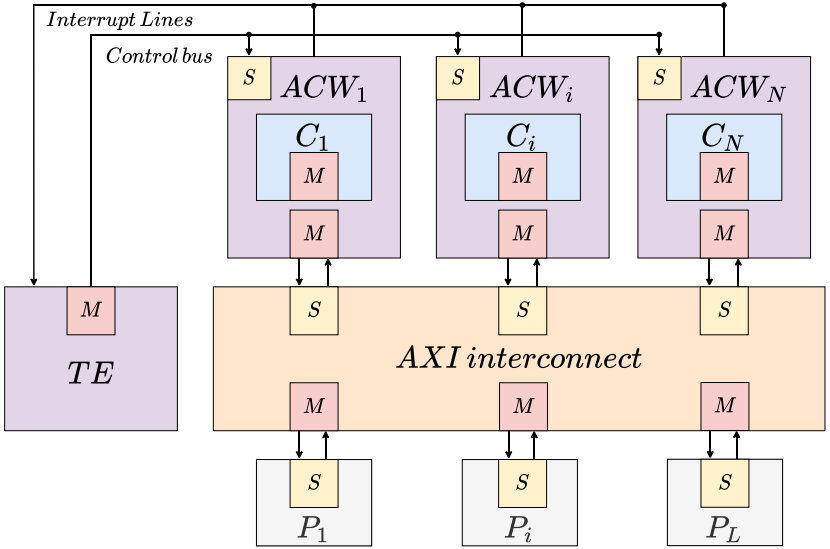
- AXI manager interface
- AXI subordinate interface
- Interrupt line



# Access Control Wrapper: How it works

- Each untrusted Controller C (IP core) is wrapped by an ACW
- Trusted Entity (TE) configures the ranges of allowed addresses for each C (HRoT, trusted processor, etc)
- Silently monitors the request issued by the hardware module
- Individually block the illegal requests
- Internally saves info about anomalies
- Interrupts to TE on illegal attempt

Extended Architecture



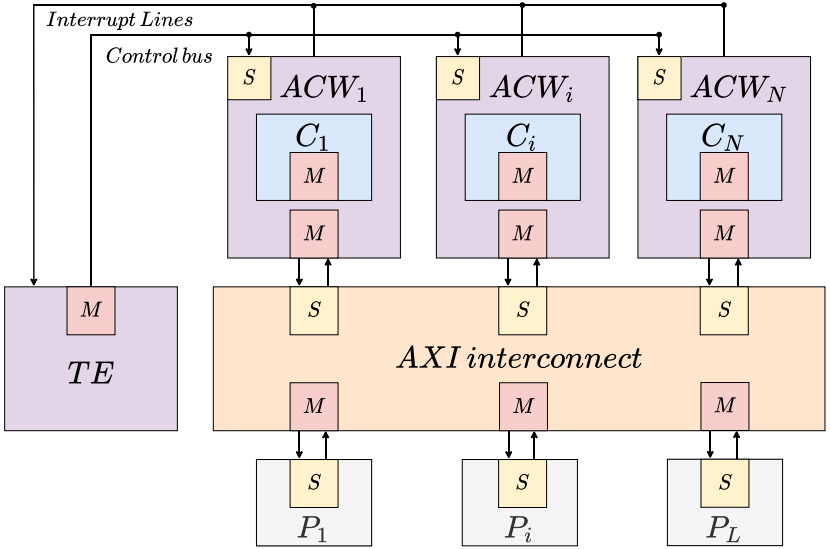
# Access Control Wrapper: Setup

The module is configured through its configuration interface by the Trusted Entity (HRoT, trusted processor, etc.)

Configured at bootup (firmware) -> static configuration

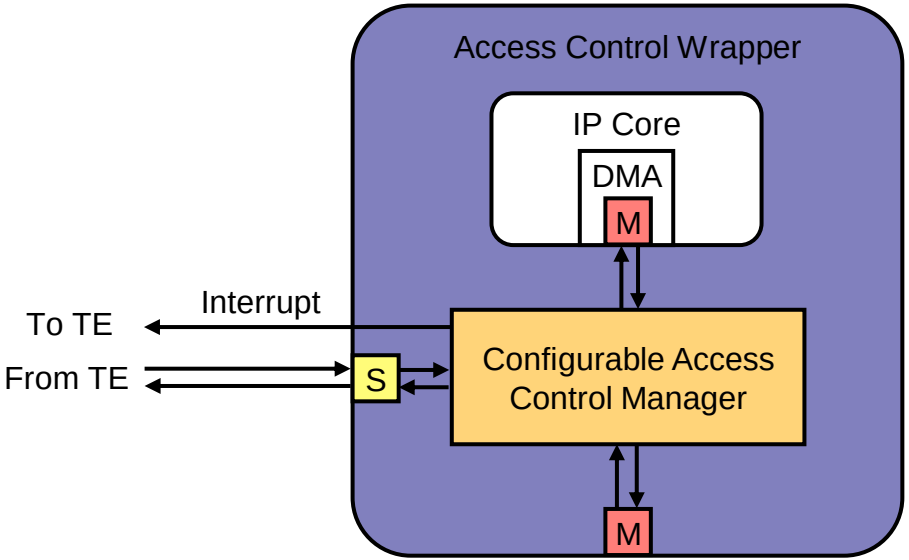
Configured at runtime (Hypervisor or OS kernel) -> initial configuration and then maintained by the TE

Extended Architecture



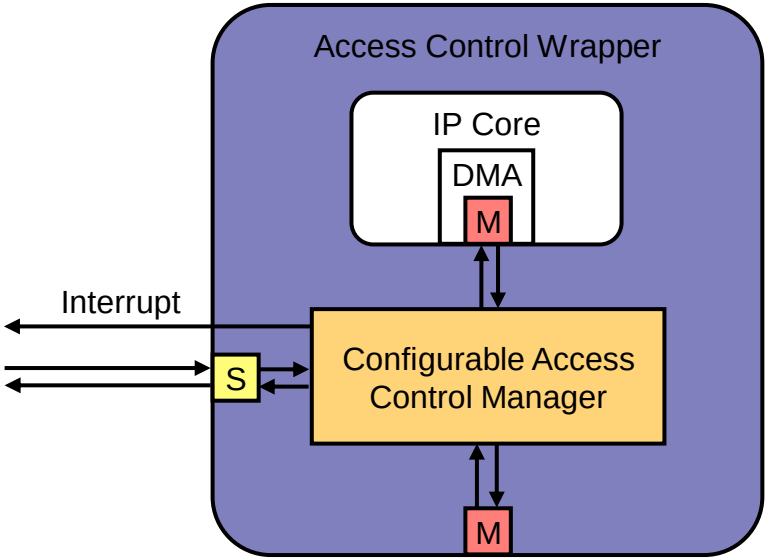
# Example: Legal Address Request

- IP Core issues a RW/WR request
- Access control machine checks requested address against the local configuration
- If address is valid, the request is propagated without modification to the external manager port
- The manager port sends the request to the AXI interconnect
- The request is completed following the AXI standard



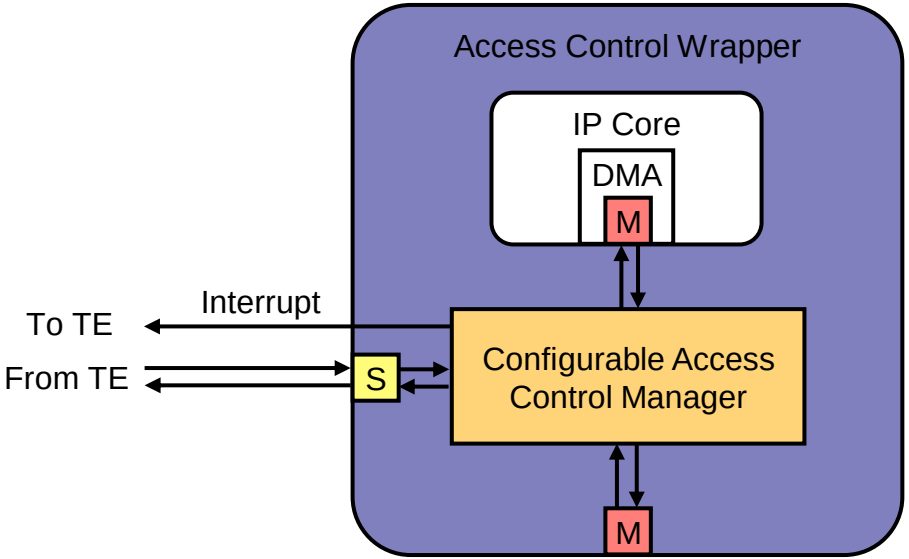
# Example: Illegal Address Request

- IP Core issues a RW/WR request
- Access control machine checks requested address against the local configuration
- If address is illegal, the request is not sent to the manager port
- Access control machine replies with an error
- Information about the illegal attempt saved into the anomalies register
- An interrupt raised to the HRoT
- The hardware module is isolated from the shared bus until HRoT readmits it



# Readmission Policy

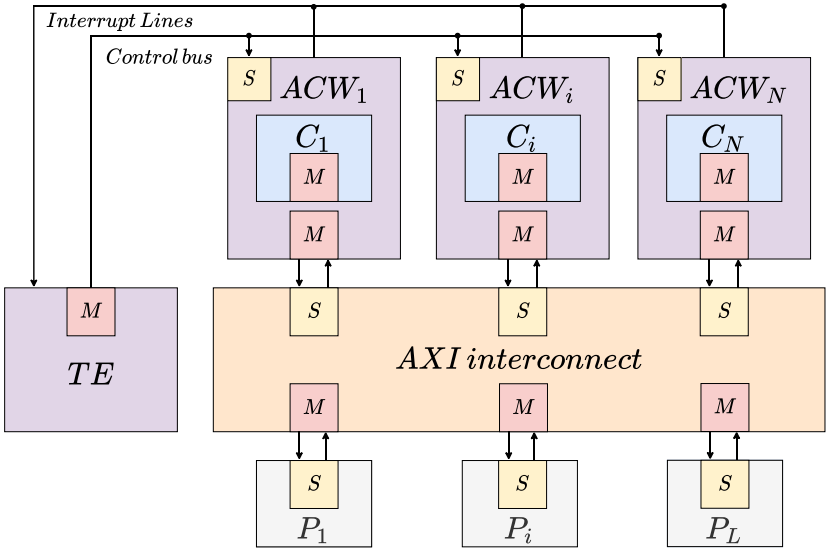
- Decoupled IP core cannot issue any RW/WR transactions
- Transactions issued and pending before the illegal attempt are correctly served
- TE makes readmission decision (or takes other action, e.g., reset the module, change access control policy, reprogram the module, …)
- Readmission performed by writing a configuration register through the subordinate configuration port



# Access Control Wrapper: Features

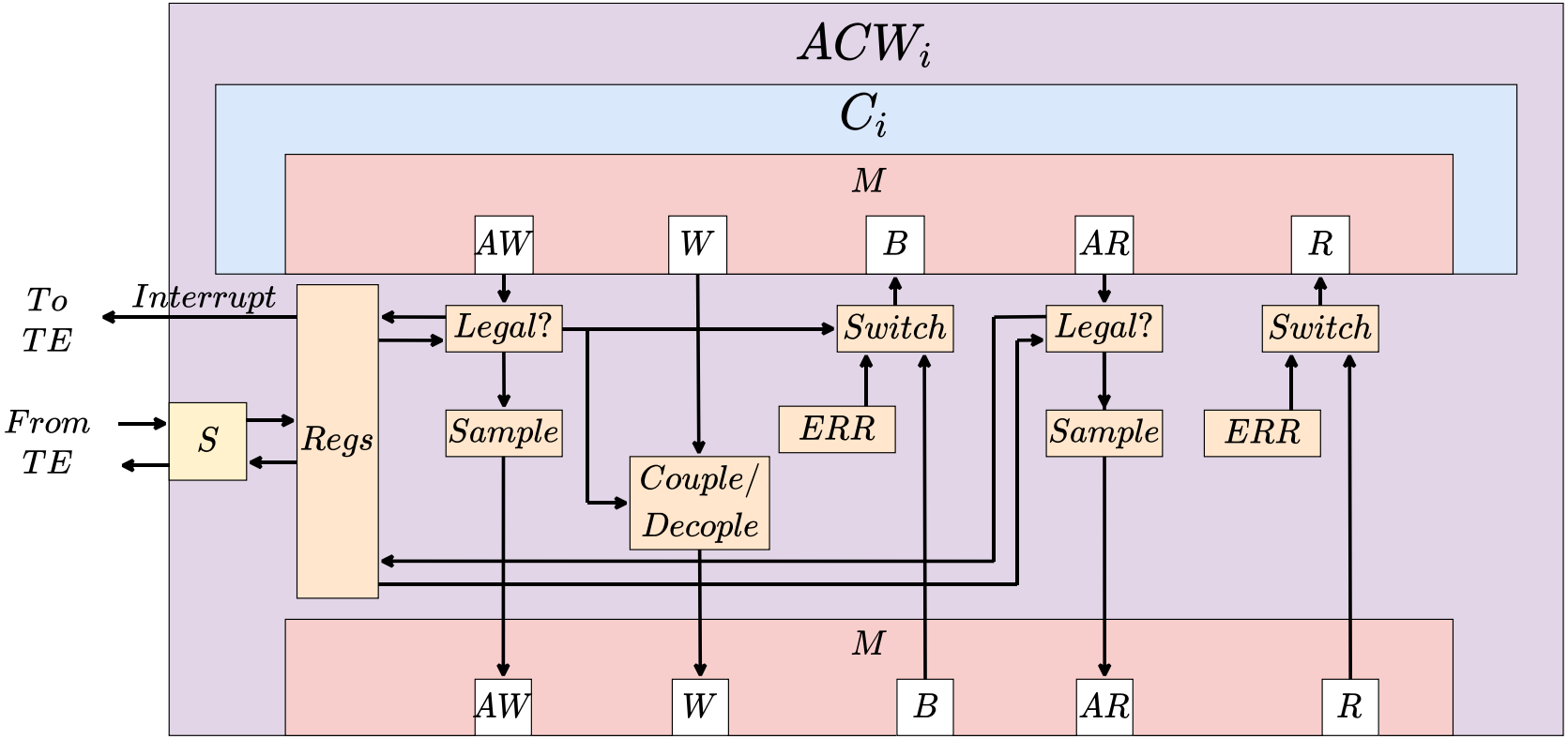
- Security verifiable: open-source design + security properties
- Interoperability: compatible to wrap any AXI manager module and transparent to the network (AXI interconnect)
- Immediate filtering of illegal requests (they never enter the network)
- Standard configuration AXI interface
- Low performance impact
- Low (customizable) resource consumption
- Diagnostic information for flexible recovery

Extended Architecture




Good for security- and safety-critical applications

# Access Control Wrapper: Internal Architecture



# Performance on FPGA SoC

- Implemented three designs -> performance comparison
- Comparison against two access control systems

    (a) AXI SmartConnect

    (b) Xilinx Memory Protection Unit (XMPU)

    (c) AKER-based Access control System with ACWs

# (a) Access control into the AXI interconnect

Advantages:

No additional hardware needed

Integrated solution

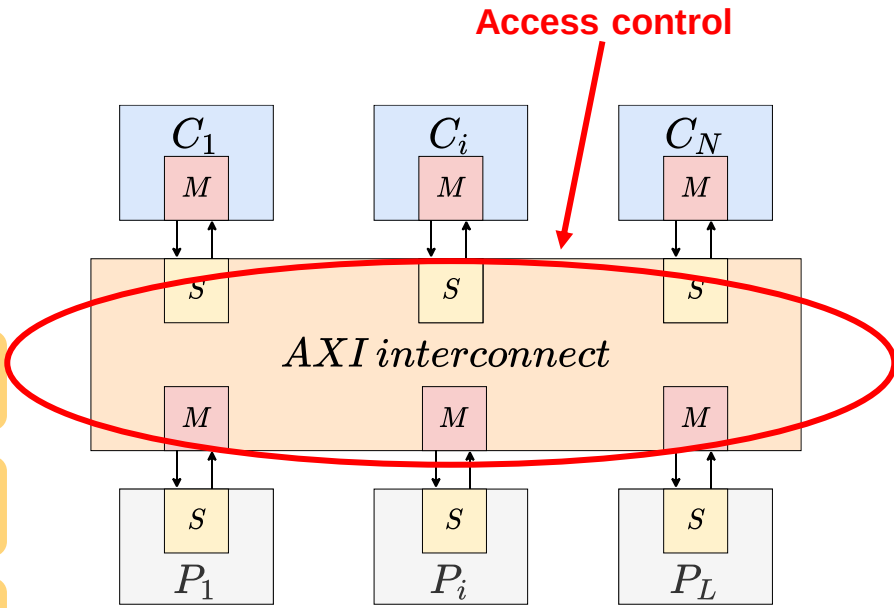
Limitations:

Lack of flexibility! Cannot change the routing (static configuration)

AXI Interconnect is a critical component for performance/footprint. Many times is outsourced.

If we don't have the code, we cannot formally verify the AXI Interconnect

Cannot be sure that the AC is absent of bugs and/or is enforced in any condition!




→ Concern for security-critical systems!

# Access control on the secondaries (XMPU)

Limitations:

Managers must be identified → AXI do not provide any signal to identify the managers (AXI ID out-of-order)

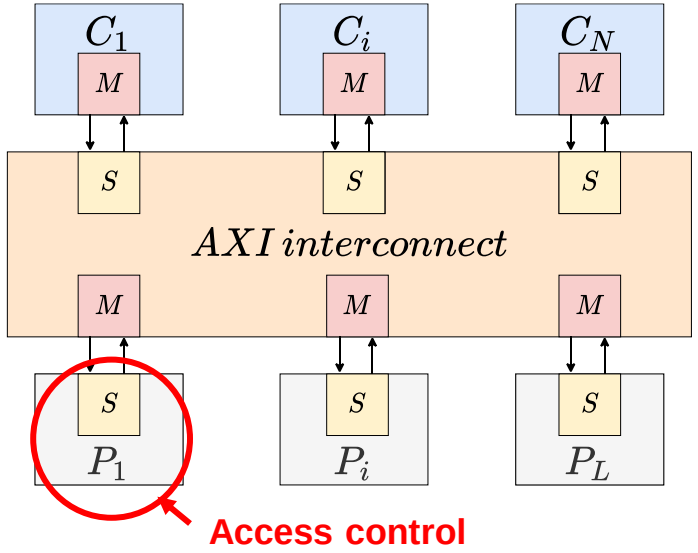
Faulty transactions are routed to the subordinates

AXI transactions cannot be aborted (from spec.)

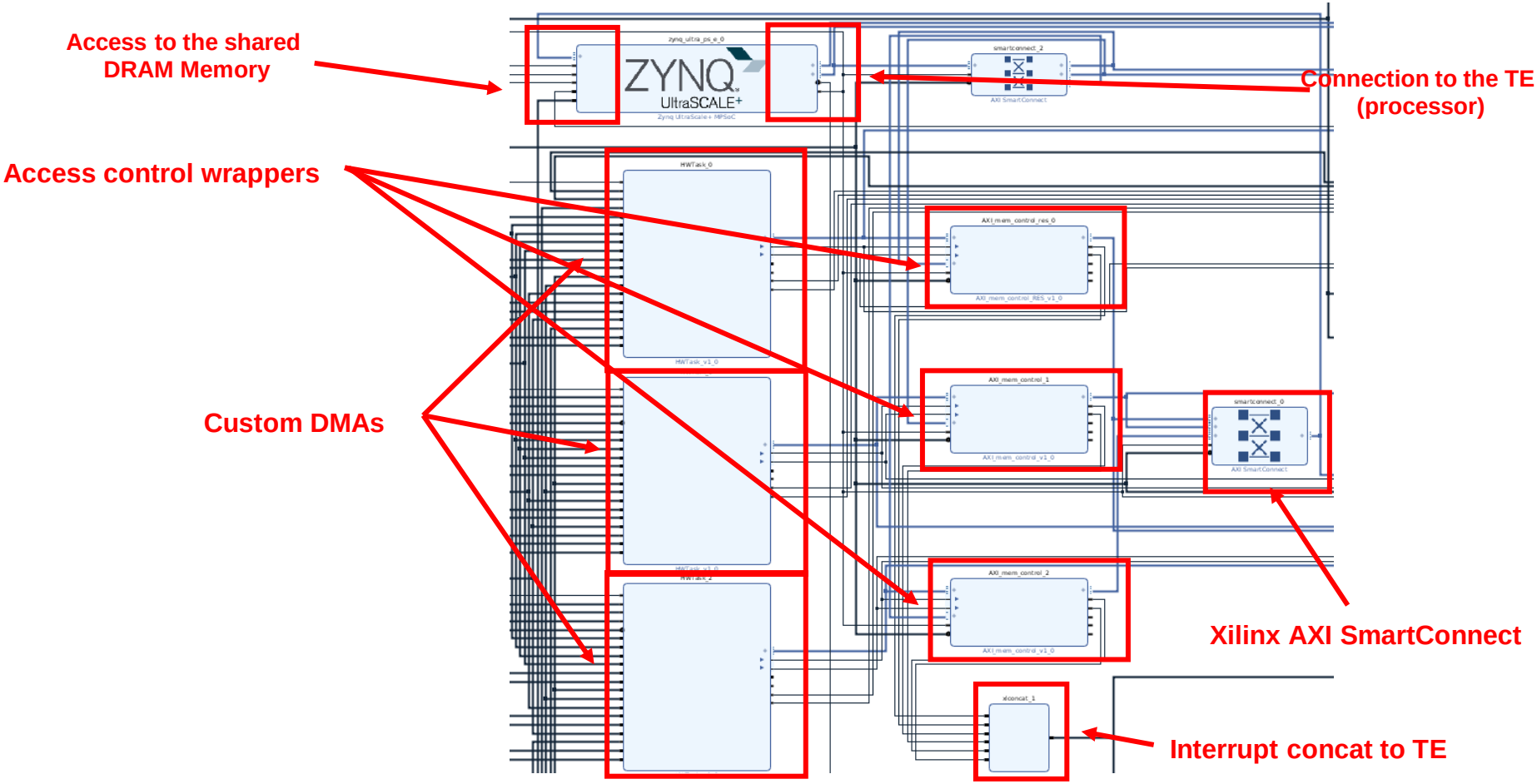
Each faulty trans. must be replied with a SLVERR

Faulty data are actually propagated and cause delays and increase the power consumption

Faulty transactions impact the response time of the other HW modules → Concern for safety-critical systems



# AKER-based access control system on FPGA SoC



# Performance on FPGA SoC

- Comparison against two access control systems

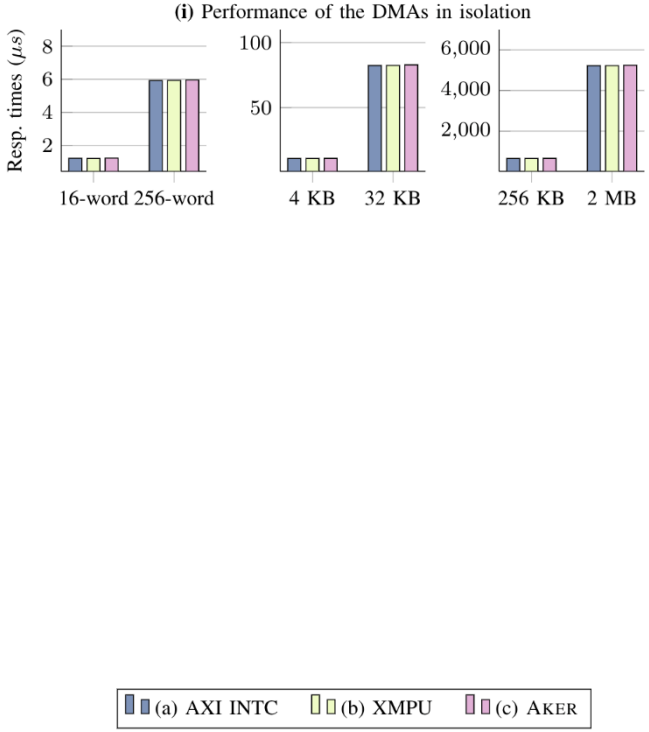
  (a) AXI SmartConnect

  (b) Xilinx Memory Protection Unit (XMPU)

  (c) Access control System with ACWs

- Running DMAs in isolation:

  Measured performance impact ~1% (worst case)



# Performance on FPGA SoC

- Comparison against two access control systems

  (a) AXI SmartConnect

  (b) Xilinx Memory Protection Unit (XMPU)

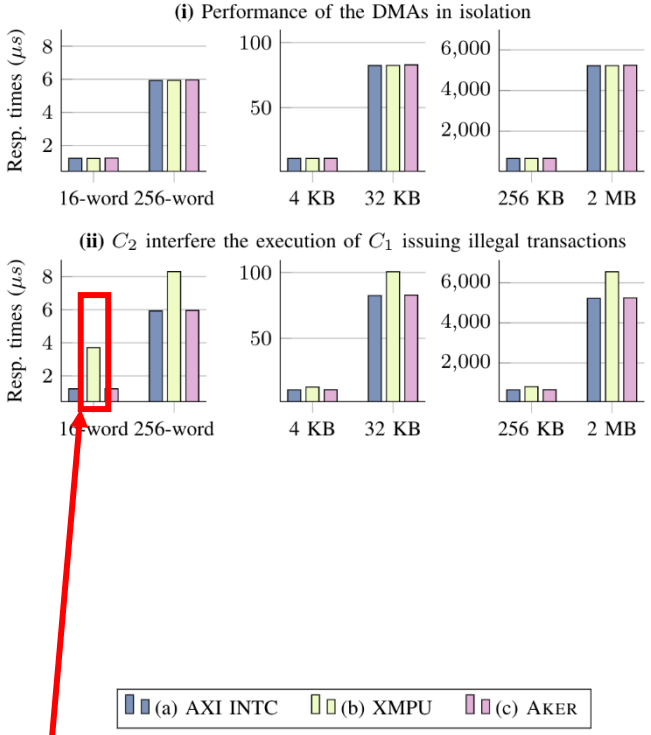
  (c) Access control System with ACWs

- Running DMAs in isolation:

  Measured performance impact ~1% (worst case)

- DMAs under contention

  No performance losses due to serving illegal requests




**Response time +200% in this scenario with XMPU**

# Performance on FPGA SoC

- Implemented three designs -> performance comparison
- Comparison against two access control systems

  (a) AXI SmartConnect

  (b) Xilinx Memory Protection Unit (XMPU)

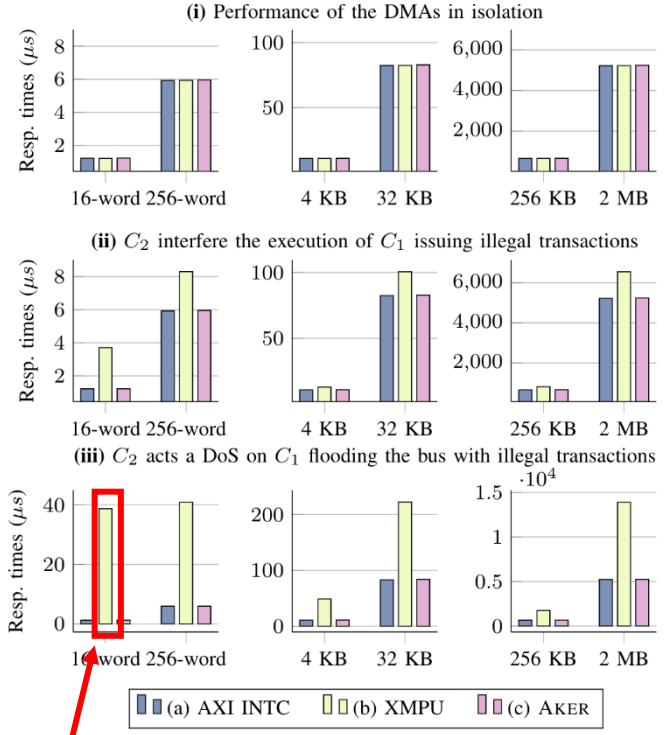
  (c) Access control System with ACWs

- Running DMAs in isolation:

  Measured performance impact ~1% (worst case)

- DMAs under contention

  No performance losses due to serving illegal requests




**Response time +3074% in this scenario with XMPU**

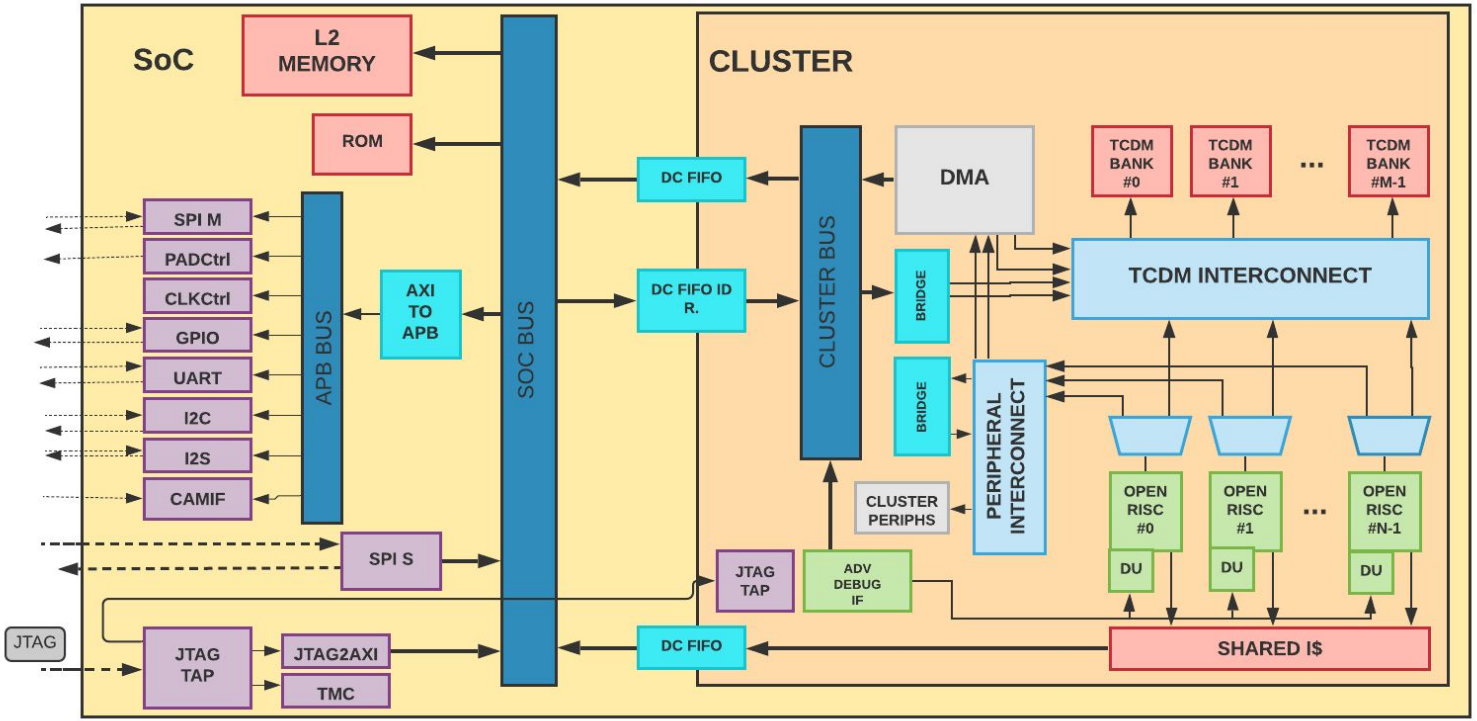
# Resource consumption on FPGA SoC

- Resource consumptions for ACW implemented on FPGA SoC
- Comparison with the Xilinx AXI SmartConnect (AXI SMTC)
- Reported results for ACWs implementing 2 regions, 4 regions, 8 regions, and 16 regions.
- The resource consumption are customizable according to the requirements of the application

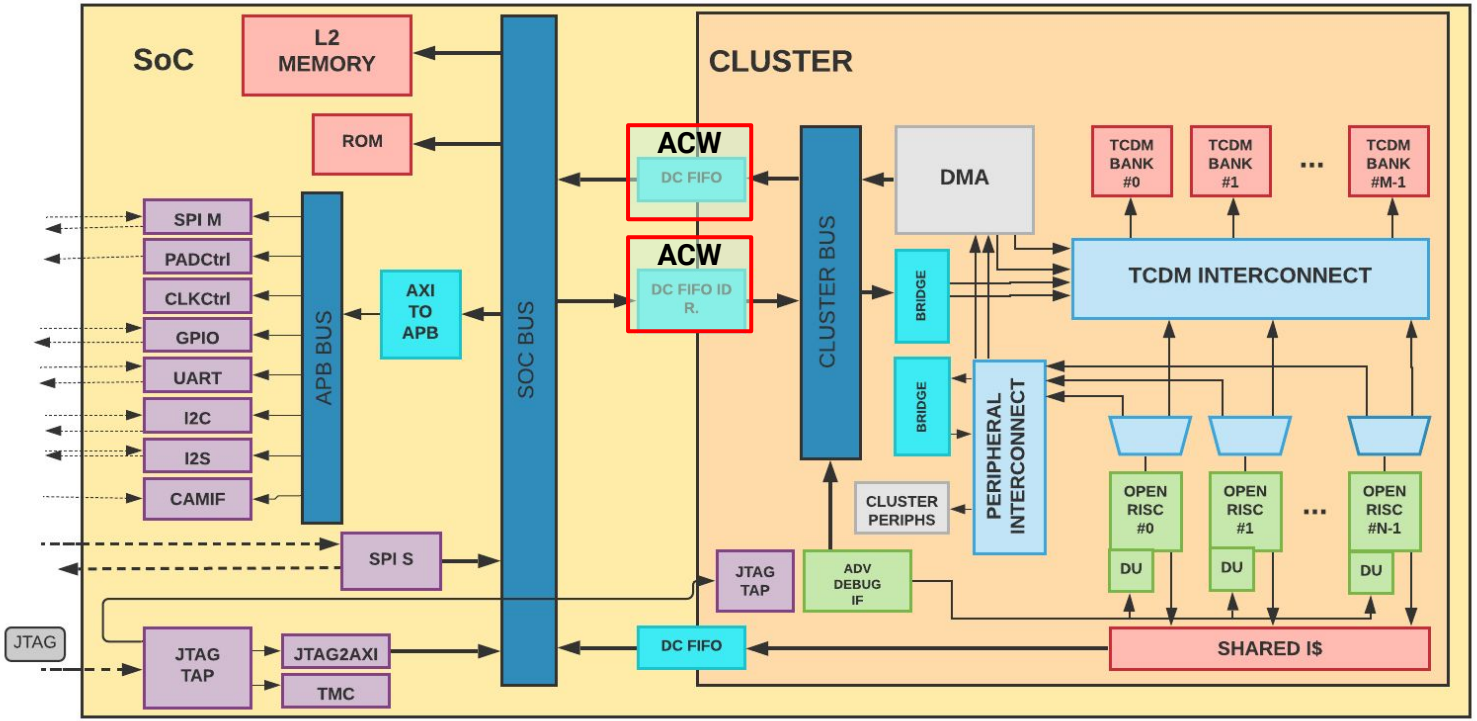
| Resources | **AXI SMTC** | **2 regs** | **4 regs** | **8 regs** | **16 regs** |
|---|---|---|---|---|---|
| **LUT** | 5626 | 263 | 326 | 467 | 730 |
| **FF** | 6688 | 294 | 358 | 486 | 744 |

# AKER Integration Efforts

# AKER + OpenPULP - The Default OpenPULP

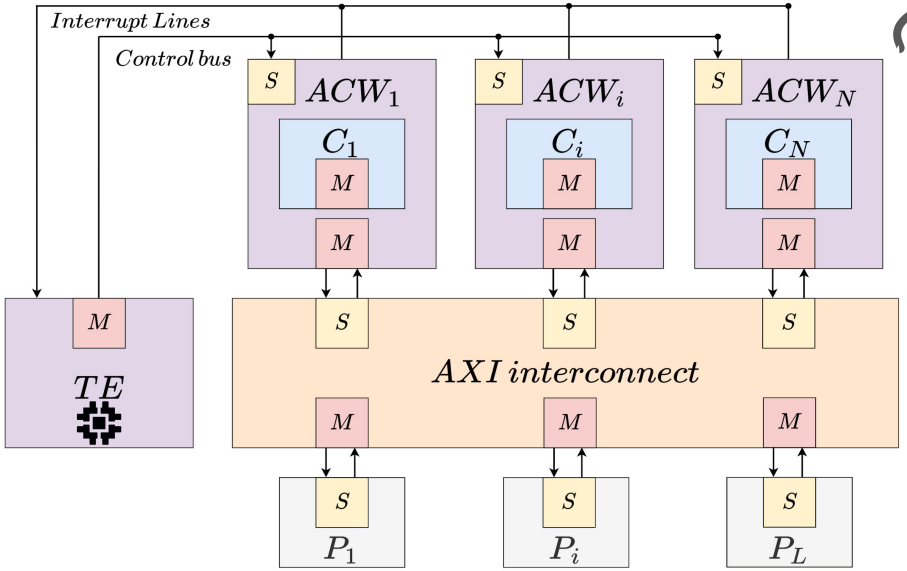


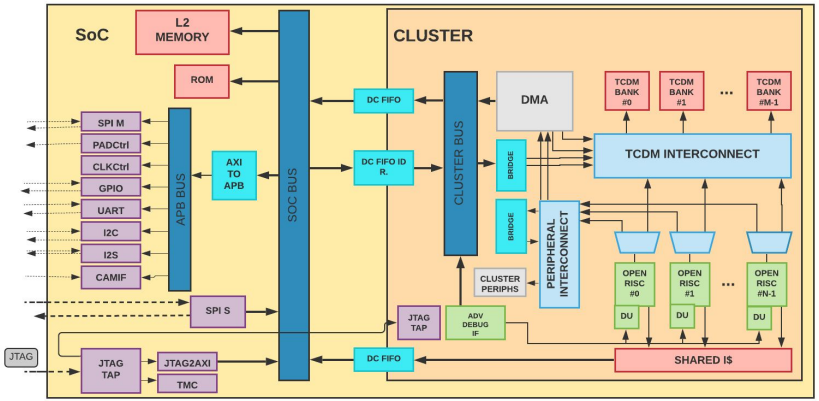
# AKER + OpenPULP - The Modified OpenPULP



# AKER + OpenPULP + OpenTitan

opentitan



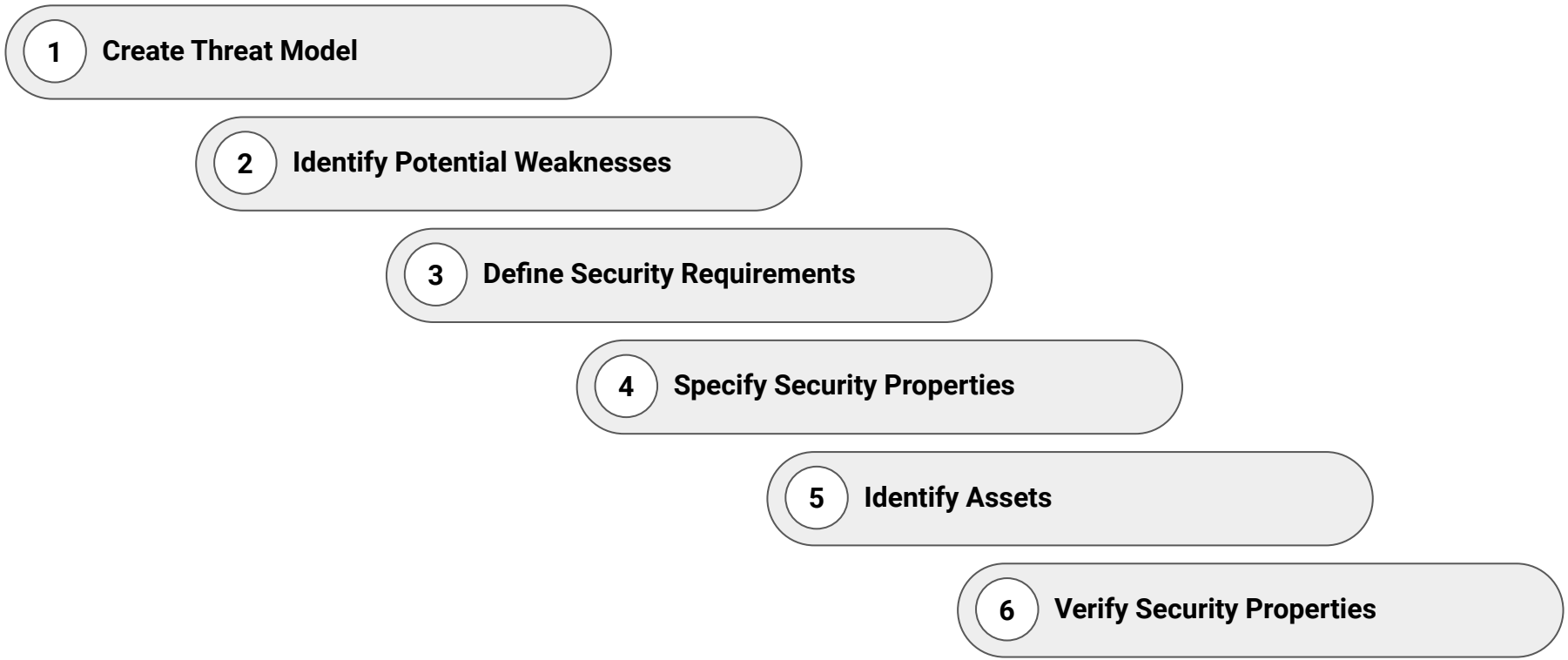



# AKER + OpenPULP + OpenTitan - Functional Verification

| PULP Configuration | C Program (no Cluster) | Expect | C Program (Cluster) | Expect |
|---|---|---|---|---|
| Default | Pass | Pass | Pass | Pass |
| ACW (2SoC) (2Cluster) (Open) (Open) | Pass | Pass | Pass | Pass |
| ACW (2SoC) (2Cluster) (Closed) (Closed) | Pass | Pass | Fail | Fail |
| ACW (2SoC) (2Cluster) (Open) (Closed) | Pass | Pass | Fail | Fail |
| ACW (2SoC) (2Cluster) (Closed) (Open) | Pass | Pass | Fail | Fail |
| ACW (2SoC) (2Cluster) (Partial) (Partial) | Pass | Pass | Pass/Fail | Pass/Fail |

# AKER Security Verification

# AKER Security Verification Process



# AKER Security Verification Process

- The security verification of the ACW is broken into three levels:
  - IP Level
  - Firmware Level
  - System Level
- All of the properties verified from lower levels are still applicable to higher levels

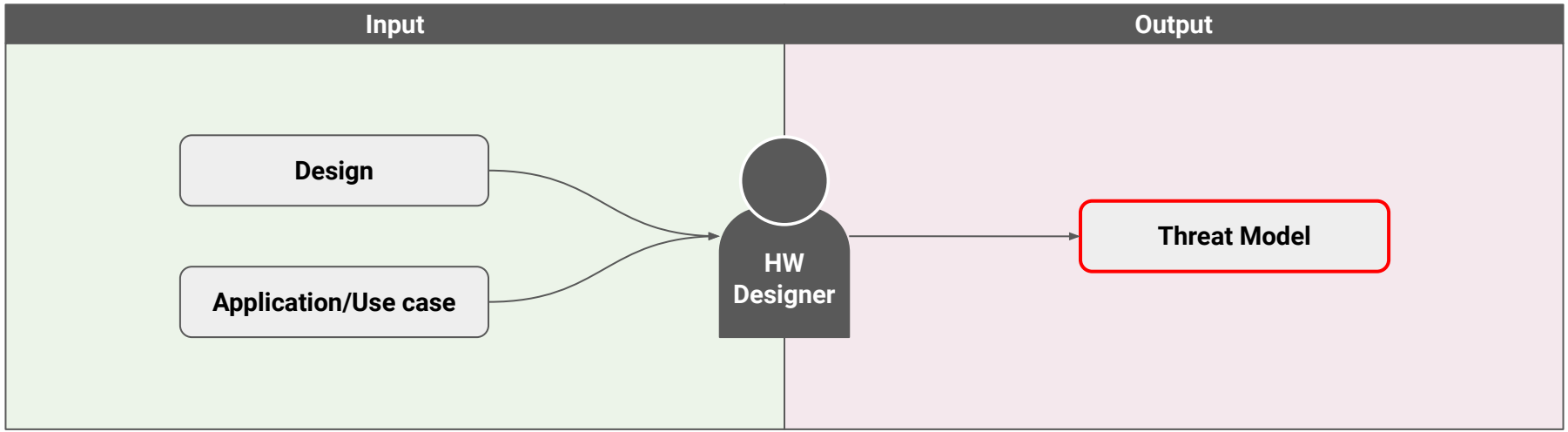
# ACW

IP Level Security Verification

*Verifying the security of a single statically-configured access control wrapper.*

# 1. Create Threat Model

- The first step in the security verification process develops the threat model.
- Hardware threats are vast and must be assessed based upon the usage of the hardware under design.
- The threat model will help determine which security threats are relevant.



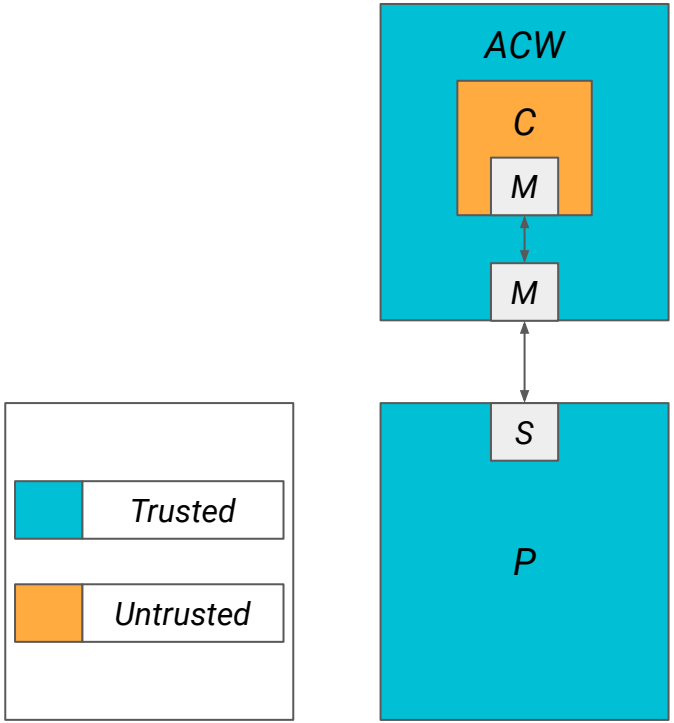
# 1. Create Threat Model

At the IP level, there are three entities that we are concerned with:

- the controller (C)
- the access control wrapper (ACW)
- the peripheral (P)

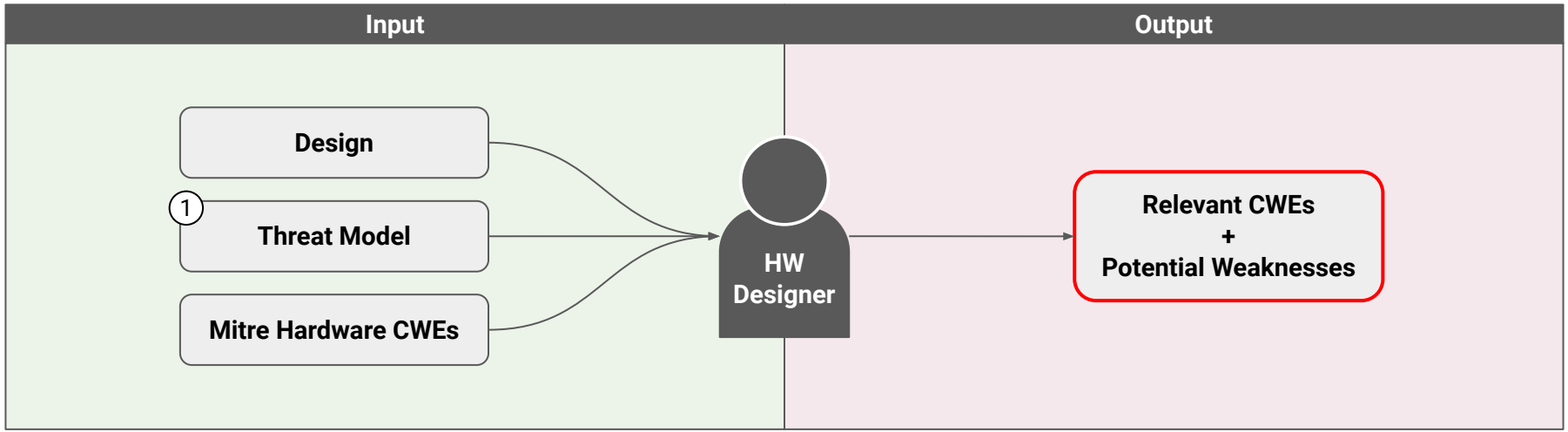
We consider an integrity scenario where C is untrusted and the ACW and P are trusted.

Therefore, the threat model considers C's ability to communicate with P via the ACW in a manner which does not adhere to the statically-configured LACP.



# 2. Identify Potential Weaknesses

- The second step identifies potential weaknesses in the design. A potential weakness is any mechanism that could introduce a security vulnerability relevant to the threat model.
- Identifying these weaknesses is often challenging and time-consuming.
- In an effort to increase the chance of identifying security critical weaknesses, we use the threat model and design to find relevant CWEs from MITRE's extensive database.



# CWE-1280: Access Control Check Implemented After Asset is Accessed

**Weakness ID: 1280**
**Abstraction:** Base
**Structure:** Simple

**Status:** Incomplete

*Presentation Filter:* Complete

## Description

A product's hardware-based access control check occurs after the asset has been accessed.

## Extended Description

The product implements a hardware-based access control check. The asset should be accessible only after the check is successful. If, however, this operation is not atomic and the asset is accessed before the check is complete, the security of the system may be compromised.

## Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that the user may want to explore.

***Relevant to the view "Research Concepts" (CWE-1000)***

| Nature | Type | ID | Name |
|---|---|---|---|
| ChildOf | P | 284 | Improper Access Control |
| ChildOf | | 696 | Incorrect Behavior Order |

***Relevant to the view "Hardware Design" (CWE-1194)***

| Nature | Type | ID | Name |
|---|---|---|---|
| MemberOf | C | 1198 | Privilege Separation and Access Control Issues |

## Modes Of Introduction

The different Modes of Introduction provide information about how and when this weakness may be introduced. The Phase identifies a point in the life cycle at which introduction may occur, while the Note provides a typical scenario related to introduction during the given phase.

| Phase | Note |
|---|---|
| Implementation | |

## Applicable Platforms

The listings below show possible areas for which the given weakness could appear. These may be for specific named Languages, Operating Systems, Architectures, Paradigms, Technologies, or a class of such platforms. The platform is listed along with how frequently the given weakness appears for that instance.

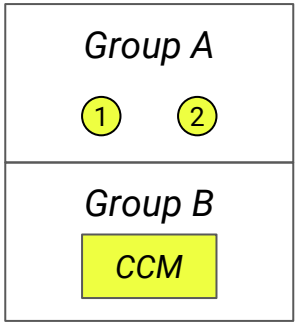
# 2. Identify Potential Weaknesses

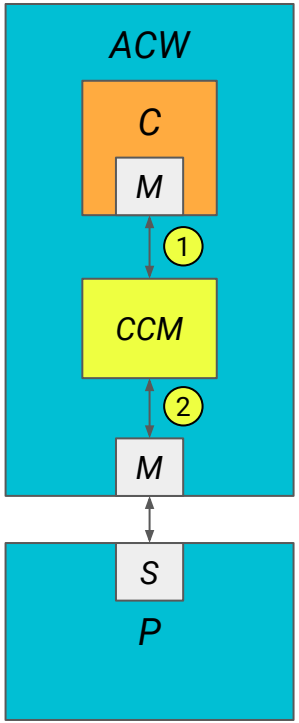
We identified the following 17 relevant CWEs:

1220, 1221, 1244, 1258, 1259, 1264, 1266, 1267, 1268, 1269, 1270, 1271, 1272, 1274, 1280, 1282, 1326

These aided in identifying two groups of weaknesses:

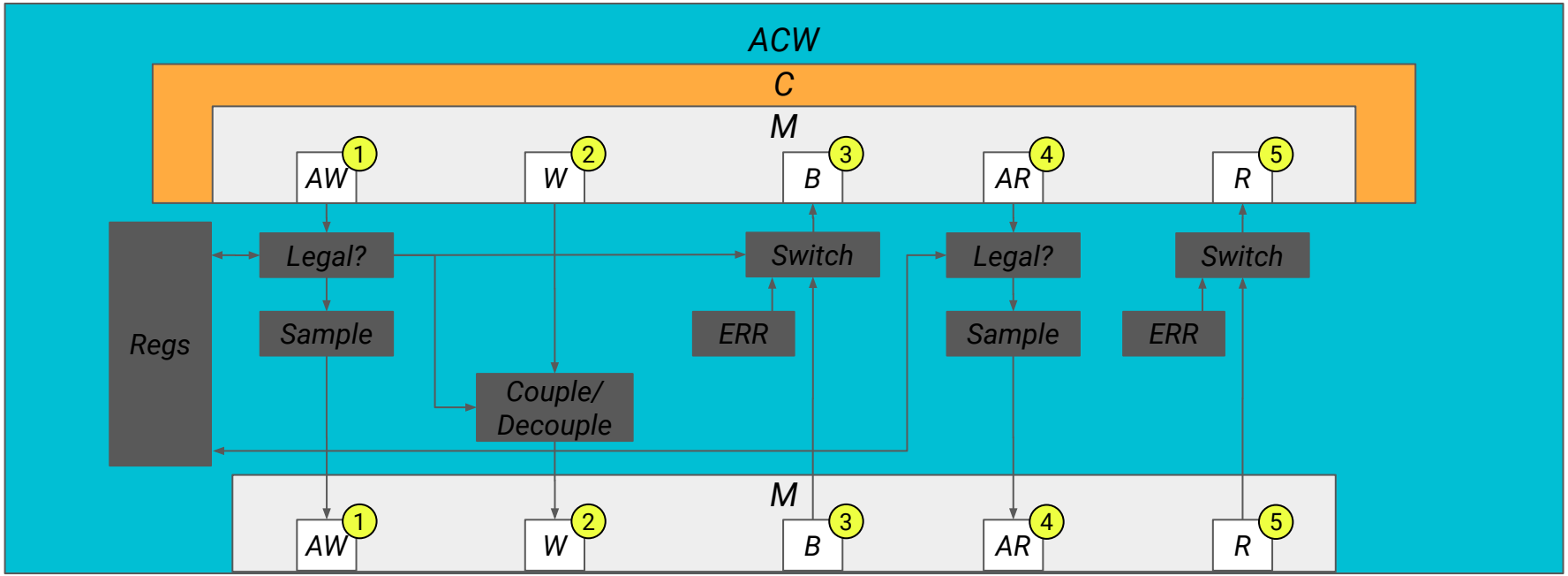
- Group A: C's read/write access points
- Group B: the ACW's configuration and control mechanisms (CCM).



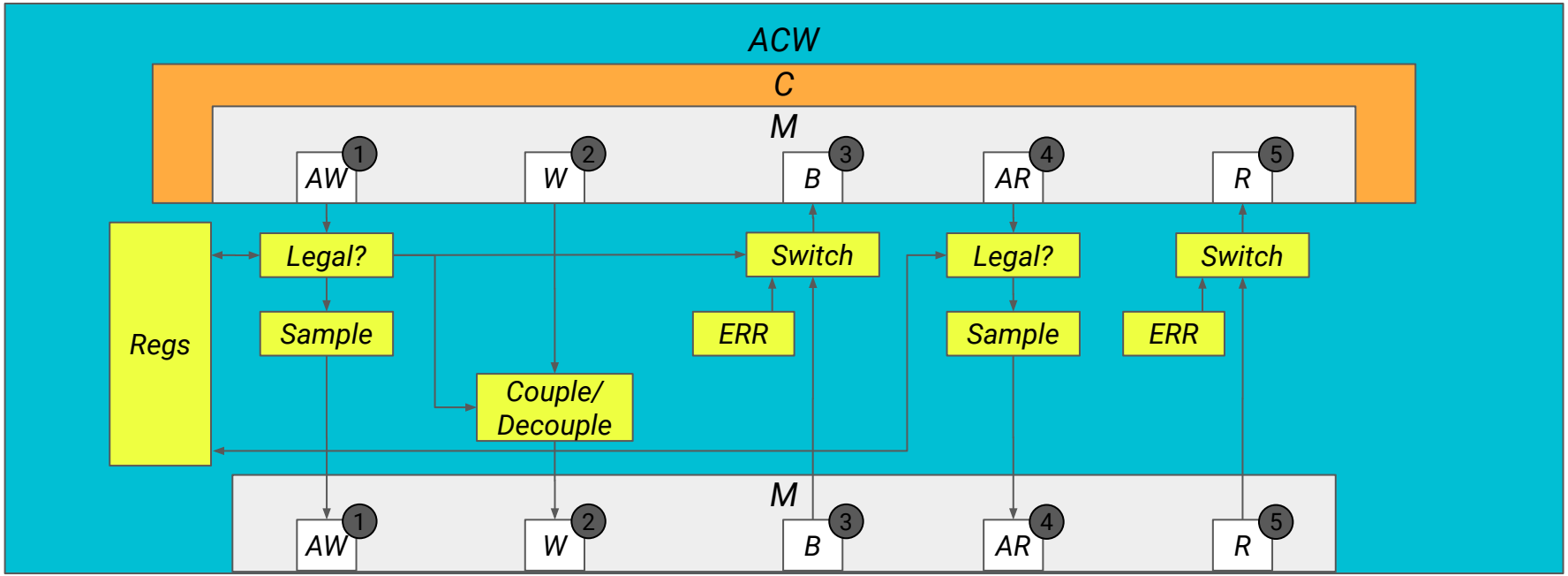
# 2. Identify Potential Weaknesses *(Group A)*

The five read/write AXI channels which make up Group A are shown below.



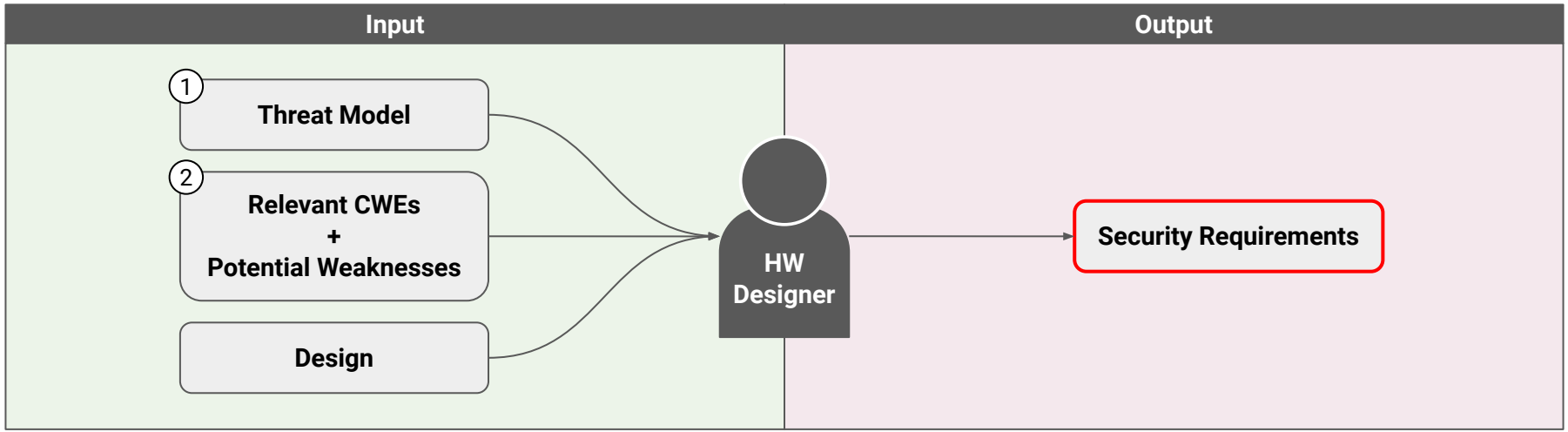
# 2. Identify Potential Weaknesses *(Group B)*

The configuration and control mechanisms which make up Group B are shown below.

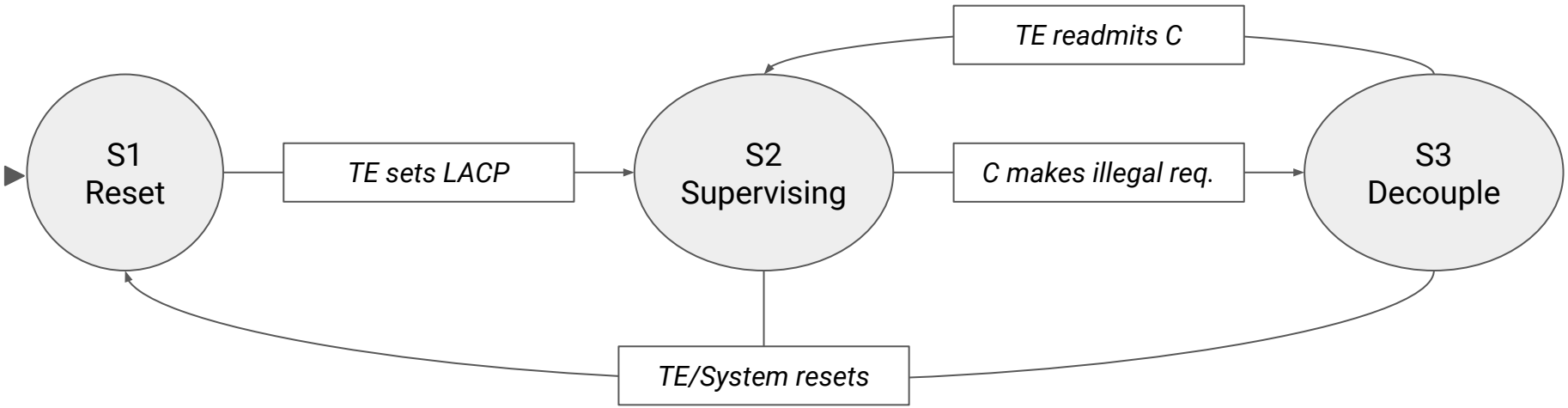


# 3. Define Security Requirements

- The third step defines plain-language security requirements for the identified weaknesses.
- Once a mechanism is identified as a potential weakness, designers can articulate a security requirement for it.
- Requirements should addresses how that mechanism could fail as determined from the relevant CWEs and an analysis of the design.



# 3. Define Security Requirements *(ACW Modes)*




*LACP = local access control policy*

# 3. Define Security Requirements *(ACW Modes)*

Below is an overview of the intended behavior of the ACW design along with its modes

| Mode | Description | Security Implication for C |
|---|---|---|
| Active Reset (AR) | The ACW is being reset to its default state and is not serving requests. | The ACW should block controller C from sending or receiving information |
| Reset (R) | The ACW does not have a valid configuration and is not serving requests. | The ACW should block controller C from sending or receiving information. |
| Supervising (S) | The ACW does have a valid configuration set and is serving/monitoring requests. | The ACW should allow controller C to send/receive information according to the set configuration. |
| Decouple (D) | The ACW does have a valid configuration set and is not serving (new) requests because an illegal request has been detected. | The ACW should block controller C from sending or receiving information. |

# 3. Define Security Requirements *(AR)*

Requirements related to when the ACW is in active reset mode.

| SP No. | Related CWEs | SP Requirement | ACW Mode | Weakness Group |
|---|---|---|---|---|
| 1 | 1258, 1266, 1270, 1271, 1272, 1280 | C cannot receive/send data from/to P which originates while the ACW is actively being reset. | AR | A |
| 2 | 1221, 1258, 1266, 1269, 1271, 1280 | C receives the default AXI signals while the ACW is actively being reset. | AR | A |
| 3 | 1221, 1258, 1266, 1269, 1271, 1280 | The ACW outputs the default AXI signals to P while the ACW is actively being reset. | AR | A |
| 4 | 1221, 1258, 1259, 1266, 1267, 1269, 1271, 1274, 1280, 1282 | The configuration/anomaly registers are cleared and set to contain the default values while the ACW is actively being reset. | AR | B |

# 3. Define Security Requirements *(R)*

Requirements related to when the ACW is in reset mode.

| SP No. | Related CWEs | SP Requirement | ACW Mode | Weakness Group |
|---|---|---|---|---|
| 6 | 1258, 1270, 1272, 1280 | C cannot receive/send data from/to P which originates while the ACW is in reset mode. | R | A |
| 7 | 1221, 1258, 1269, 1272, 1280 | C receives the default AXI signals while the ACW is in reset mode. | R | A |
| 8 | 1221, 1258, 1269, 1272, 1280 | The ACW outputs the default AXI signals to P while the ACW is in reset mode. | R | A |

# 3. Define Security Requirements *(S)*

Requirements related to when the ACW is in supervising mode.

| SP No. | Related CWEs | SP Requirement | ACW Mode | Weakness Group |
|---|---|---|---|---|
| 10 | 1270, 1272, 1280 | C cannot receive/send data associated with an illegal address from/to P which originates while the ACW is in supervising mode. | S | A |

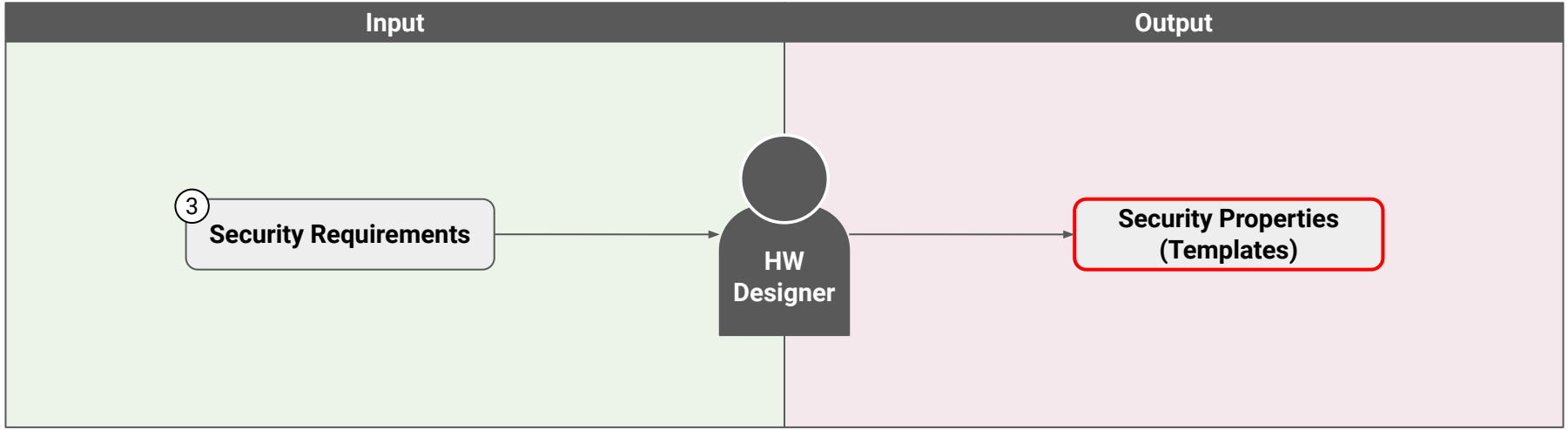
# 3. Define Security Requirements *(D)*

Requirements related to when the ACW is in decouple mode.

| SP No. | Related CWEs | SP Requirement | ACW Mode | Weakness Group |
|---|---|---|---|---|
| 11 | 1270, 1272, 1280 | C cannot receive/send data from/to P which originates while the ACW is in decouple mode. | D | A |
| 12 | 1221, 1269, 1272, 1280 | C receives the default AXI signals while the ACW is in decouple mode. | D | A |
| 13 | 1221, 1269, 1272, 1280 | The ACW outputs the default AXI signals to the P while the ACW is in decouple mode. | D | A |
| 14 | 1272, 1280, 1283 | The anomaly registers are updated with illegal request metadata after the ACW detects an illegal request. | D | B |

# 4. Specify Security Properties

- The fourth step specifies a security property template for each of the security requirements.
- In order to verify a security requirement, a formally specified security property which uses explicit values, design signals, and operators to form an evaluable expression is needed.
- Rather than specifying nearly identical security properties for every design signal that should adhere to a given security requirement, we specify security property templates with placeholder signals.



# 4. Specify Security Properties *(SP Types)*

Security properties fall into one of two categories: information flow* and trace properties.

**Sample Information Flow Property**

Information from C originating during reset mode will never flow to P.

```
‘sig_from_C‘     (source)
when (RST == 0) (tainting condition)
=/=>            (no-flow operator)
‘sig_to_P‘       (destination)
```

**Sample Trace Property**

C receives the baseline AXI signals while the ACW is in reset mode.

```
‘sig_to_C‘ == ‘val‘ (value check)
|| (RST != 1)       (condition)
```

** Requires information flow tools**

# 4. Specify Security Properties *(SP 1)*

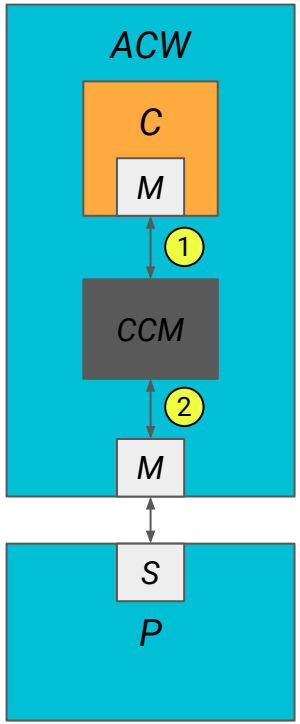
**Security Property - 1:**

C cannot receive/send data from/to P which originates while the ACW is actively being reset.

```
//# of Generic Signals to Replace = 2
SP01_RECEIVE_GENERIC: assert iflow(
  `signal_from_P`
   when (ARESETN == 0)
   =/=>
  `signal_to_C`
);

//# of Generic Signals to Replace = 2
SP01_SEND_GENERIC: assert iflow(
  `signal_from_C`
   when (ARESETN == 0)
   =/=>
  `signal_to_P`
);
```

| ACW Mode |
| --- |
| AR |
| **SP Type** |
| IFT |



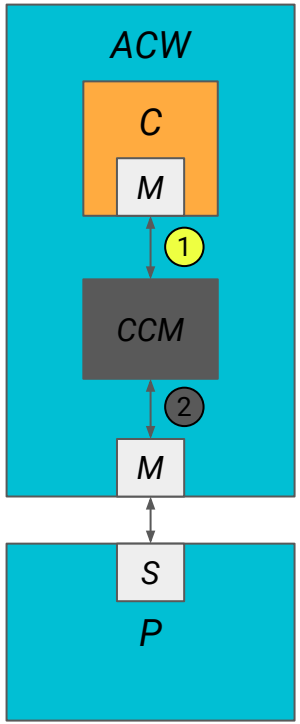
# 4. Specify Security Properties *(SP 2)*

**Security Property - 2:**

C receives the default AXI signals while the ACW is actively being reset.

```
//# of Generic Signals to Replace = 2
SP02_DEFAULT_GENERIC: assert iflow(
  `signal_to_C` == `default_AXI_value`
   unless (ARESETN != 0)
);
```

| ACW Mode |
| --- |
| AR |
| **SP Type** |
| Trace |



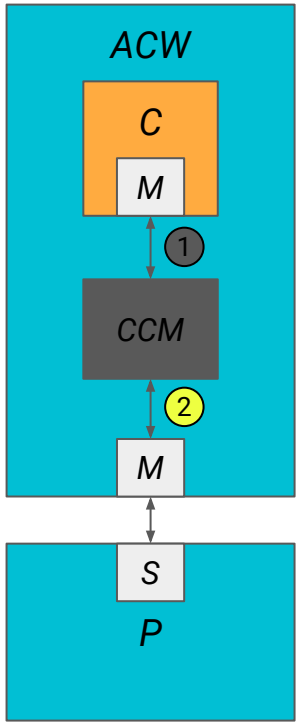
# 4. Specify Security Properties *(SP 3)*

**Security Property - 3:**

The ACW outputs the default AXI signals to P while the ACW is actively being reset.

```
//# of Generic Signals to Replace = 2
SP03_DEFAULT_GENERIC: assert iflow(
  `signal_to_P` == `default_AXI_value`
   unless (ARESETN != 0)
);
```

| ACW Mode |
| --- |
| AR |
| **SP Type** |
| Trace |



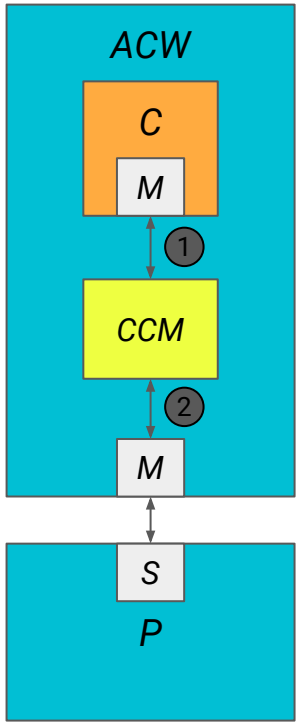
# 4. Specify Security Properties *(SP 4)*

**Security Property - 4:**

The configuration/anomaly registers are cleared and set to contain the default values while the ACW is actively being reset.

```
//# of Generic Signals to Replace = 2
SP04_DEFAULT_GENERIC: assert iflow(
  `reg` == `default_value`
   unless (ARESETN != 0 && `acw_w/r_state` != 2'b00)
);
```

| ACW Mode |
| --- |
| AR |
| **SP Type** |
| Trace |



# 4. Specify Security Properties *(SP 6)*

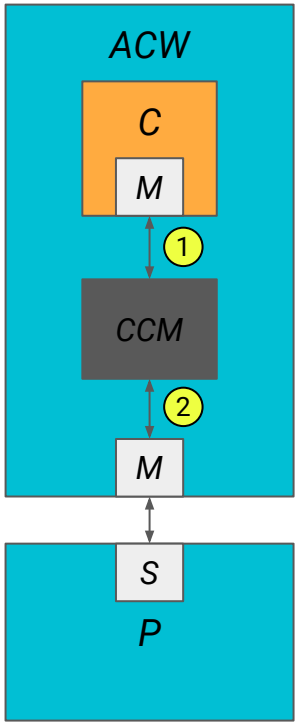
**Security Property - 6:**

C cannot receive/send data from/to P which originates while the ACW is in reset mode.

```
//# of Generic Signals to Replace = 3
SP06_RECEIVE_GENERIC: assert iflow(
  `signal_from_P`
   when (`acw_w/r_state` == 2'b00)
   =/=>
  `signal_to_C`
);

//# of Generic Signals to Replace = 3
SP06_SEND_GENERIC: assert iflow(
  `signal_from_C`
   when (`acw_w/r_state` == 2'b00)
   =/=>
  `signal_to_P`
);
```

| ACW Mode |
| --- |
| R |
| **SP Type** |
| IFT |



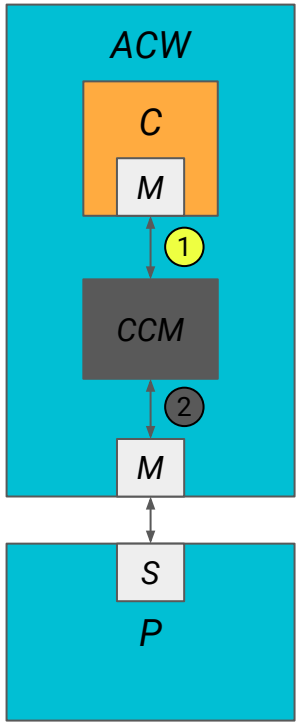
# 4. Specify Security Properties *(SP 7)*

**Security Property - 7:**

C receives the default AXI signals while the ACW is in reset mode.

```
//# of Generic Signals to Replace = 3
SP07_DEFAULT_GENERIC: assert iflow(
  `signal_to_C` == `default_AXI_value`
   unless (`acw_w/r_state` != 2'b00)
);
```

| ACW Mode |
| --- |
| R |
| **SP Type** |
| Trace |



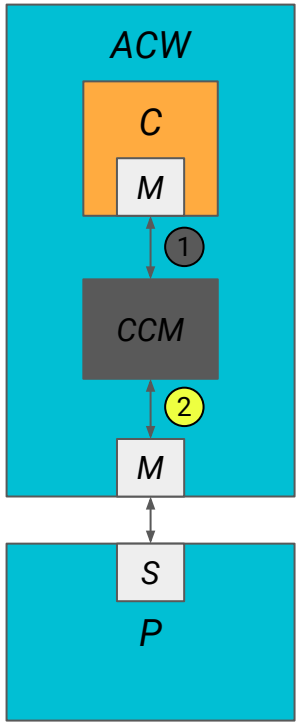
# 4. Specify Security Properties *(SP 8)*

**Security Property - 8:**

The ACW outputs the default AXI signals to P while the ACW is in reset mode.

```
//# of Generic Signals to Replace = 3
SP08_DEFAULT_GENERIC: assert iflow(
  `signal_to_P` == `default_AXI_value`
   unless (`acw_w/r_state` != 2'b00)
);
```

| ACW Mode |
| --- |
| R |
| **SP Type** |
| Trace |



# 4. Specify Security Properties *(SP 10)*

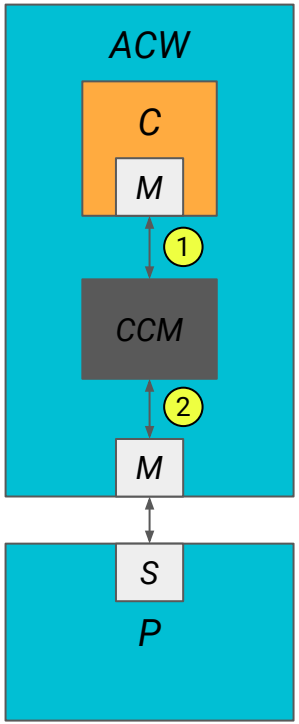
**Security Property - 10:**

C cannot receive/send data associated with an illegal address from/to P which originates while the ACW is in supervising mode.

```
//# of Generic Signals to Replace = 4
SP10_RECEIVE_GENERIC: assert iflow(
  `signal_from_P`
   when (`acw_w/r_state` == 2'b01) &&
        (`AR/AW_ADDR_VALID_FLAG` == 0)
   =/=>
  `signal_to_C`
);

//# of Generic Signals to Replace = 4
SP10_SEND_GENERIC: assert iflow(
  `signal_from_C`
   when (`acw_w/r_state` == 2'b01) &&
        (`AR/AW_ADDR_VALID_FLAG` == 0)
   =/=>
  `signal_to_P`
);
```

| ACW Mode |
| --- |
| S |
| **SP Type** |
| IFT |



# 4. Specify Security Properties *(SP 11)*

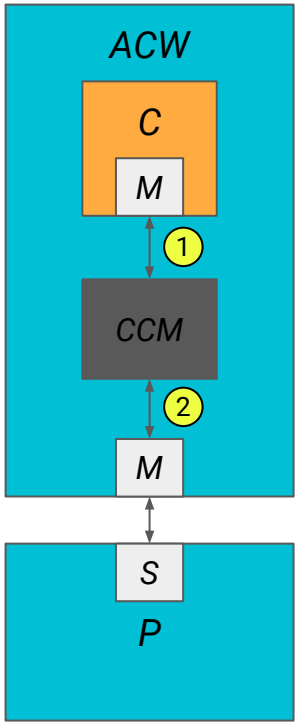
**Security Property - 11:**

C cannot receive/send data from/to P which originates while the ACW is in decouple mode.

```
//# of Generic Signals to Replace = 3
SP11_RECEIVE_GENERIC: assert iflow(
  `signal_from_P`
   when (`acw_w/r_state` == 2'b10)
   =/=>
  `signal_to_C`
);

//# of Generic Signals to Replace = 3
SP11_SEND_GENERIC: assert iflow(
  `signal_from_C`
   when (`acw_w/r_state` == 2'b10)
   =/=>
  `signal_to_P`
);
```

| ACW Mode |
| --- |
| D |
| **SP Type** |
| IFT |



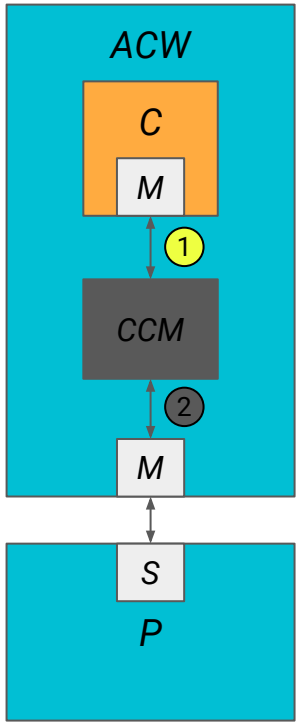
# 4. Specify Security Properties *(SP 12)*

**Security Property - 12:**

C receives the default AXI signals while the ACW is in decouple mode.

```
//# of Generic Signals to Replace = 3
SP12_DEFAULT_GENERIC: assert iflow(
  `signal_to_C` == `default_AXI_value`
   unless (`acw_w/r_state` != 2'b10)
);
```

| ACW Mode |
| --- |
| D |
| **SP Type** |
| Trace |



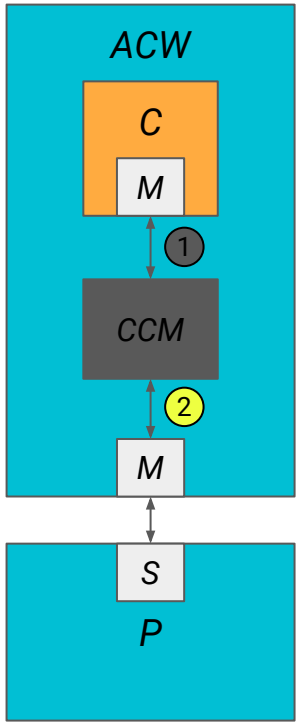
# 4. Specify Security Properties *(SP 13)*

**Security Property - 13:**

The ACW outputs the default AXI signals to the P while the ACW is in decouple mode.

```
//# of Generic Signals to Replace = 3
SP13_DEFAULT_GENERIC: assert iflow(
  `signal_to_P` == `default_AXI_value`
   unless (`acw_w/r_state` != 2'b10)
);
```

| ACW Mode |
| --- |
| D |
| **SP Type** |
| Trace |



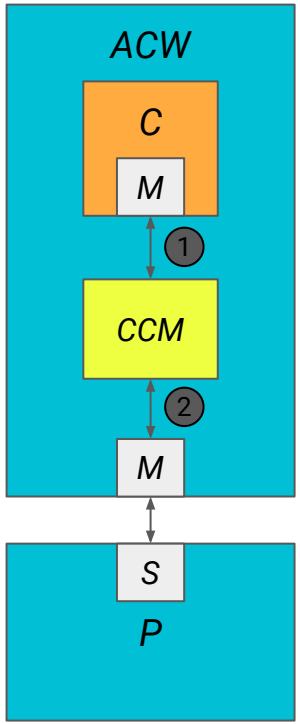
# 4. Specify Security Properties *(SP 14)*

**Security Property - 14:**

The anomaly registers are updated with illegal request metadata after the ACW detects an illegal request.

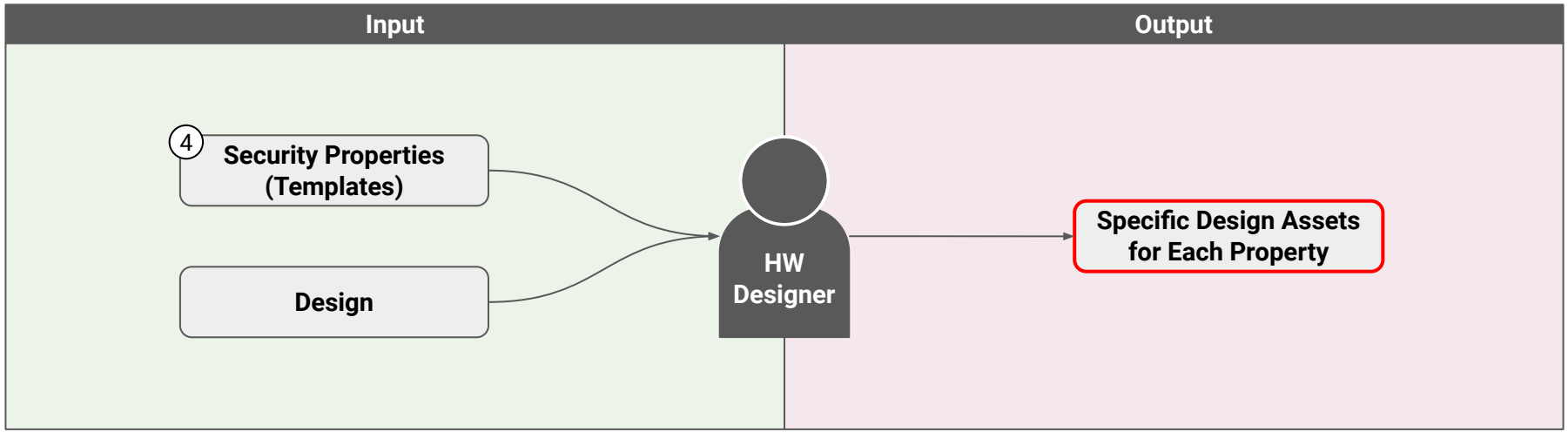
```
//# of Generic Signals to Replace = 3
SP14_DEFAULT_GENERIC: assert iflow(
  `authorized_signal`
   when (`acw_w/r_state` == 2'b01)
   =/=>
  `anomaly_reg`
   unless (`acw_w/r_state` == 2'b10)
);
```

| ACW Mode |
| --- |
| D |
| **SP Type** |
| IFT |



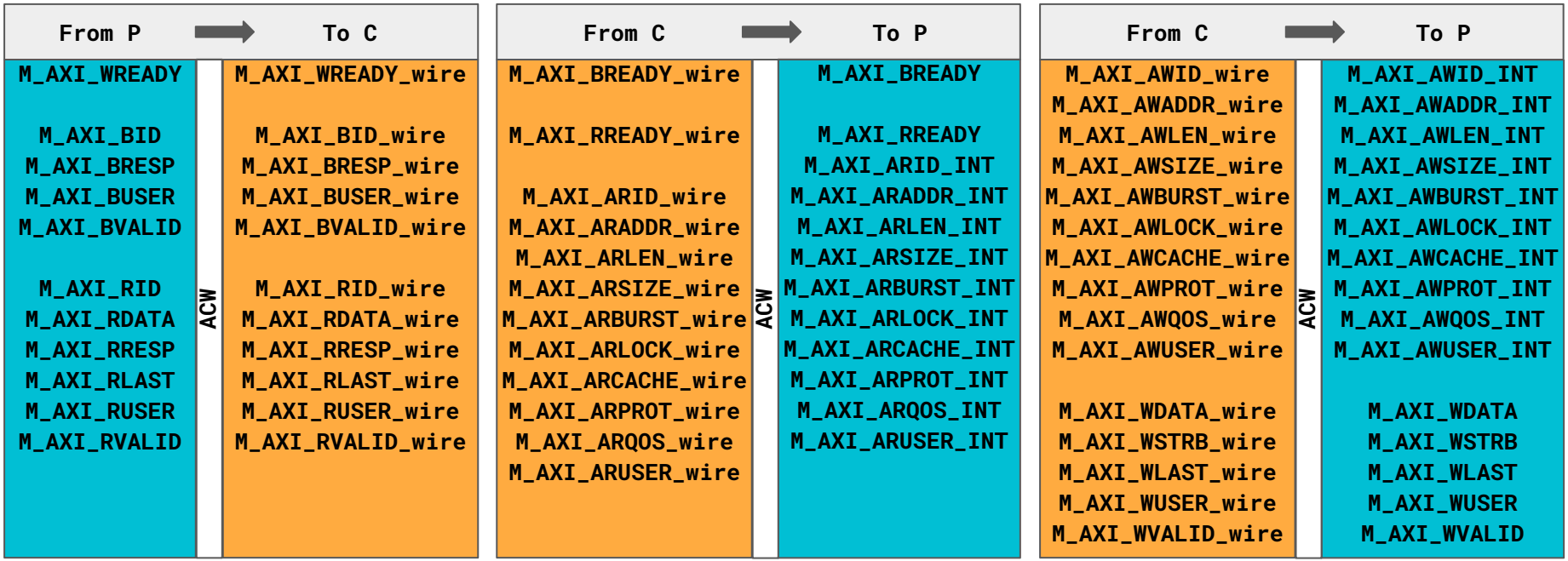
# 5. Identify Assets

- The fifth step identifies the specific assets that should adhere to each security property created in Step 4.
- Each security property will apply to at least one design signal (i.e., an asset).
- Designers must identify all of the specific assets relevant to each property. A single asset may be relevant for multiple security properties.



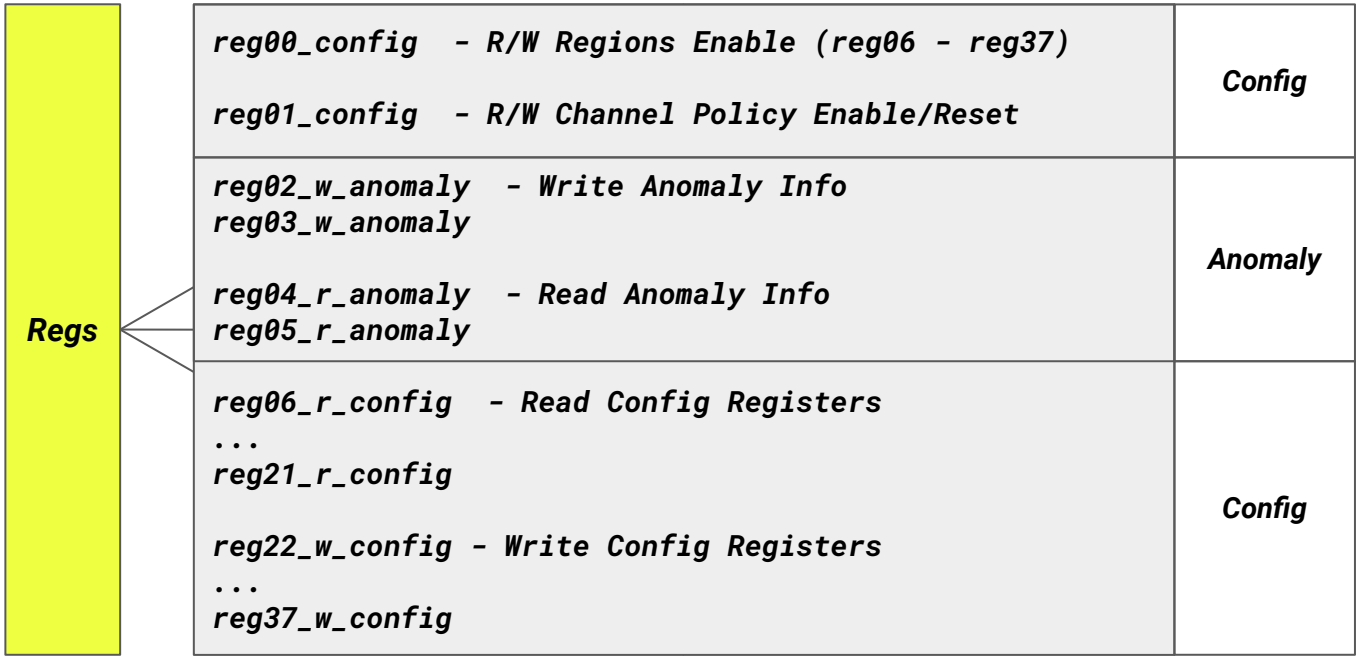
# 5. Identify Assets *(Group A Signals)*

The AXI signal pairs managed by the ACW that transmit data between P and C



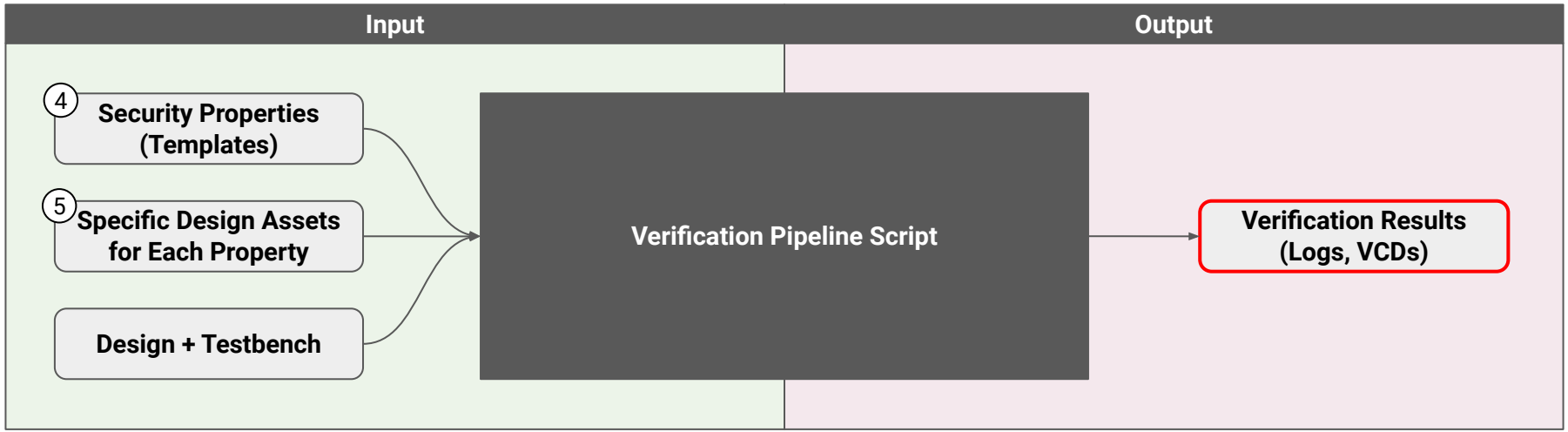
# 5. Identify Assets *(Group B Signals)*

The configuration and information registers utilized by the ACW



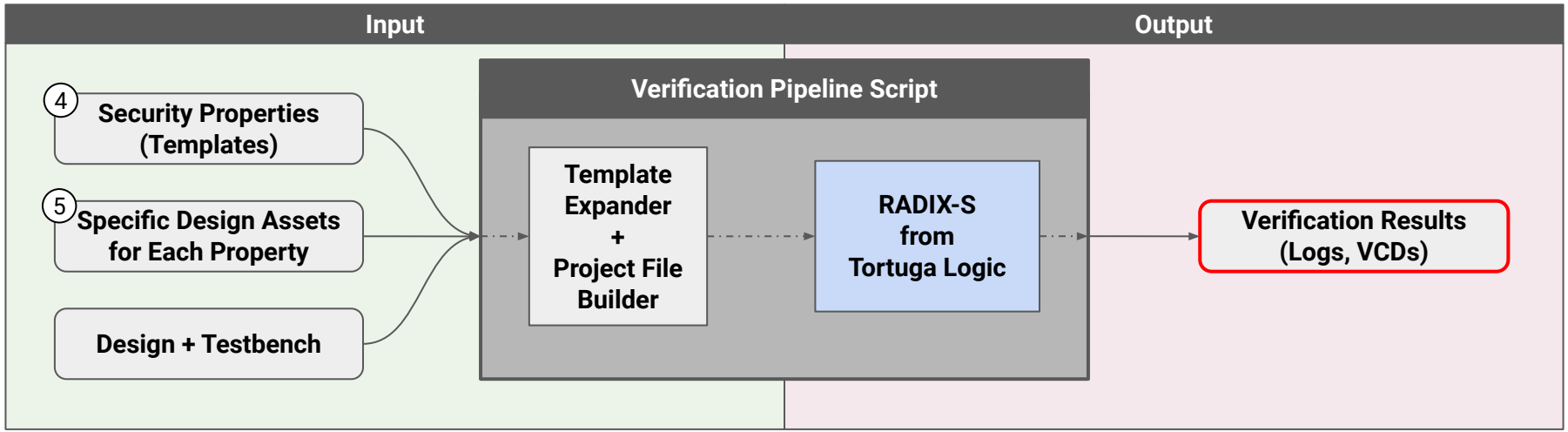
# 6. Verify Security Properties

- The final step generates specific security properties and then verifies them via simulation.
- AKER's property generation framework automatically generates these specific properties given security property templates and a list of target design signals for each template.
- Tortuga Logic Radix-S is used to generate a security model from the properties and the design. When simulated, this model will report how many times each individual property fails along with the time at which each failure occurs.



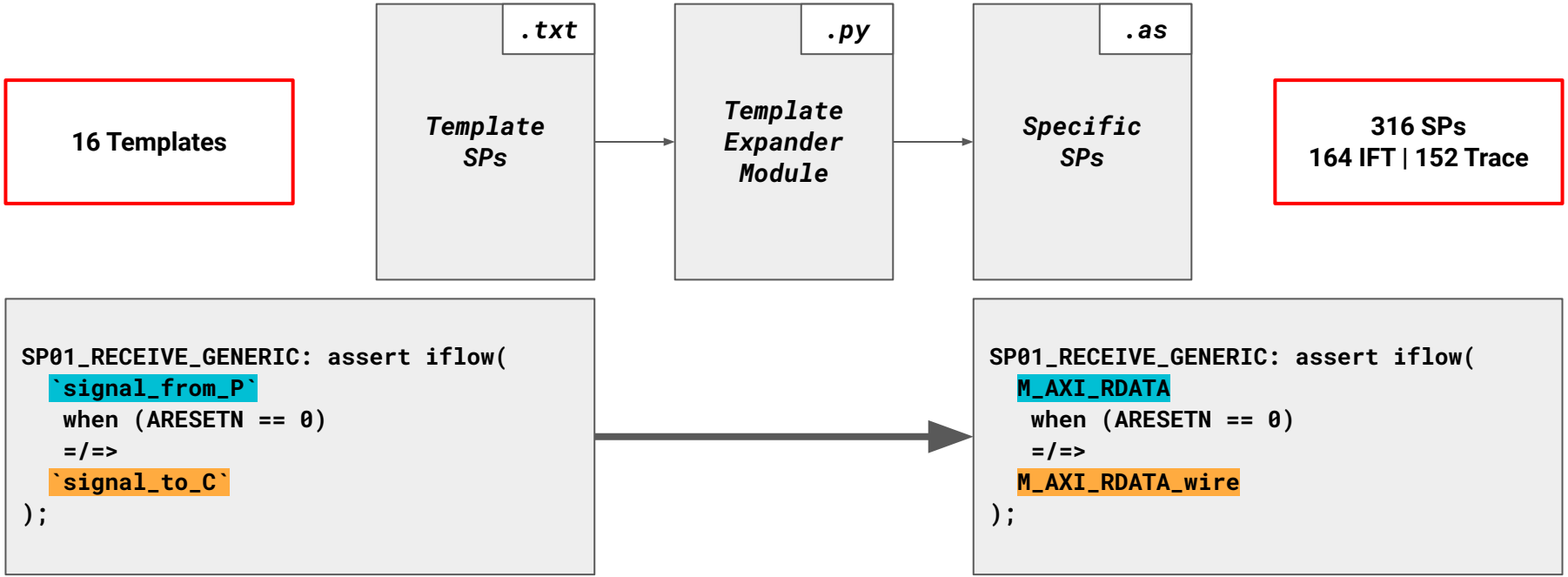
# 6. Verify Security Properties

- The final step generates specific security properties and then verifies them via simulation.
- AKER's property generation framework automatically generates these specific properties given security property templates and a list of target design signals for each template.
- Tortuga Logic Radix-S is used to generate a security model from the properties and the design. When simulated, this model will report how many times each individual property fails along with the time at which each failure occurs.



# 6. Verify Security Properties *(Template Expansion)*

Generic signals/values in the security properties must be replaced before verification/simulation.



# 6. Verify Security Properties *(Radix-S™)*

Security properties are verified via simulation using Tortuga Logic's Radix-S tool and Questa Sim

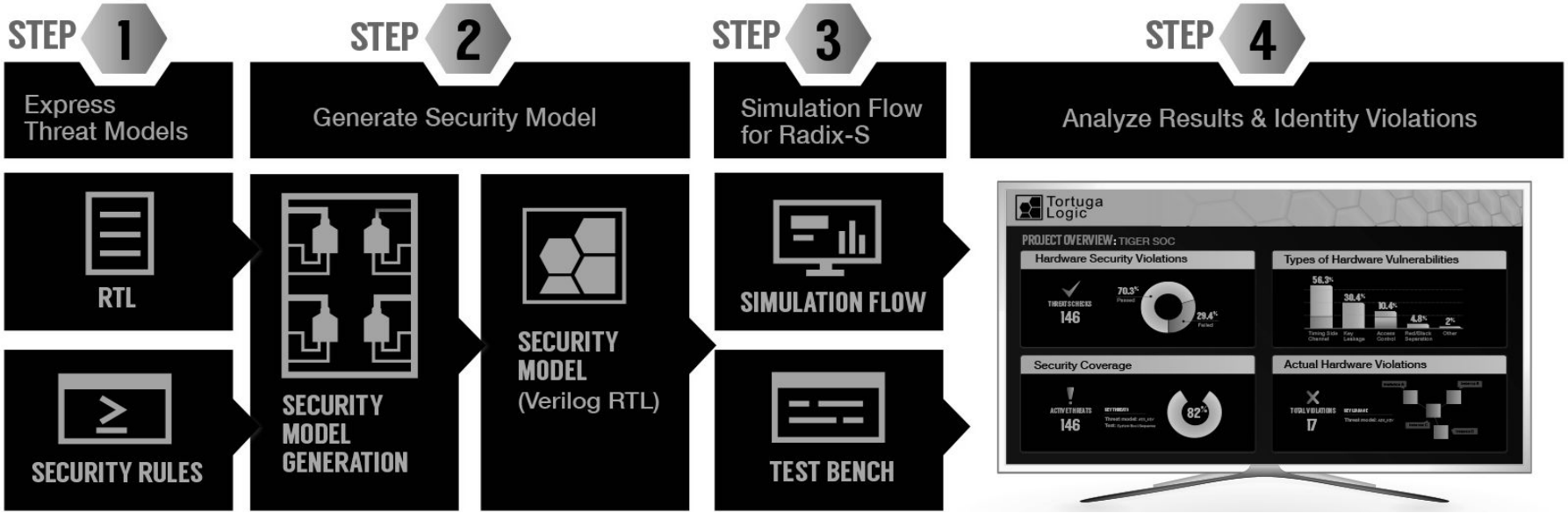



**Image Source: Tortuga Logic* Radix-S™***

# ACW

Firmware Level Security Verification

*Verifying the security of a single dynamically-configured access control wrapper.*

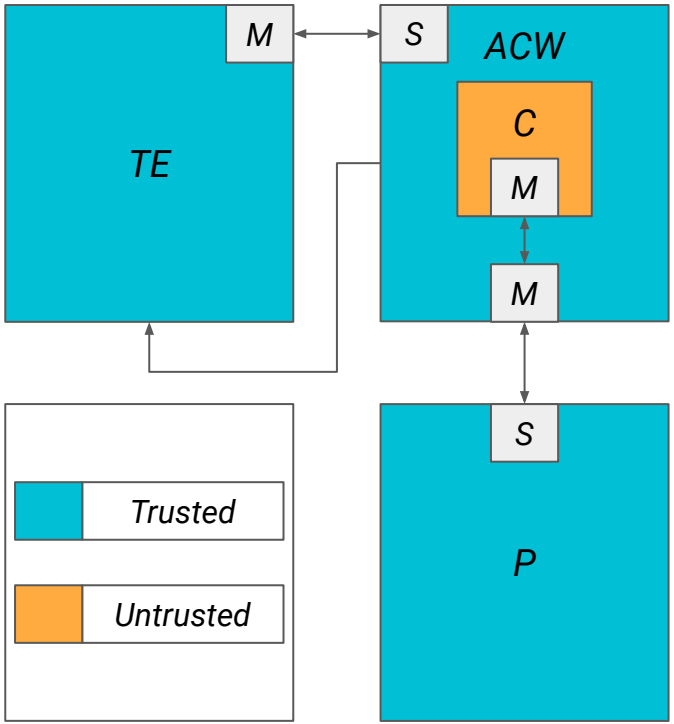
# 1. Create Threat Model

At the firmware level, there are four entities that we are concerned with:

- the controller (C)
- the access control wrapper (ACW)
- the peripheral (P)
- the hardware root of trust (TE)

We consider an integrity scenario where C is untrusted and the ACW, P, and TE are trusted.

Therefore, the threat model considers C's ability to communicate with P via the ACW in a manner which does not adhere to the dynamically-configured LACP.



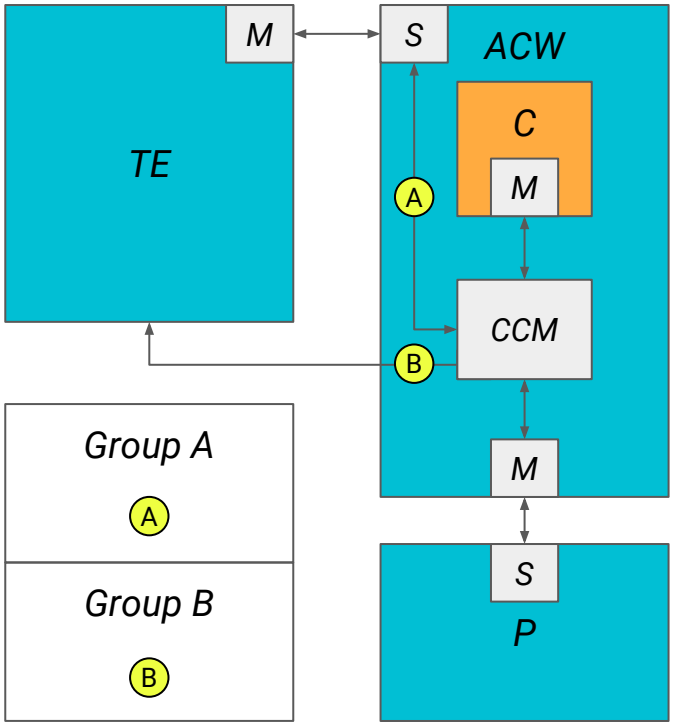
# 2. Identify Potential Weaknesses

We identified the following 7 relevant CWEs:

276, 1191, 1193, 1262, 1283, 1290, 1292

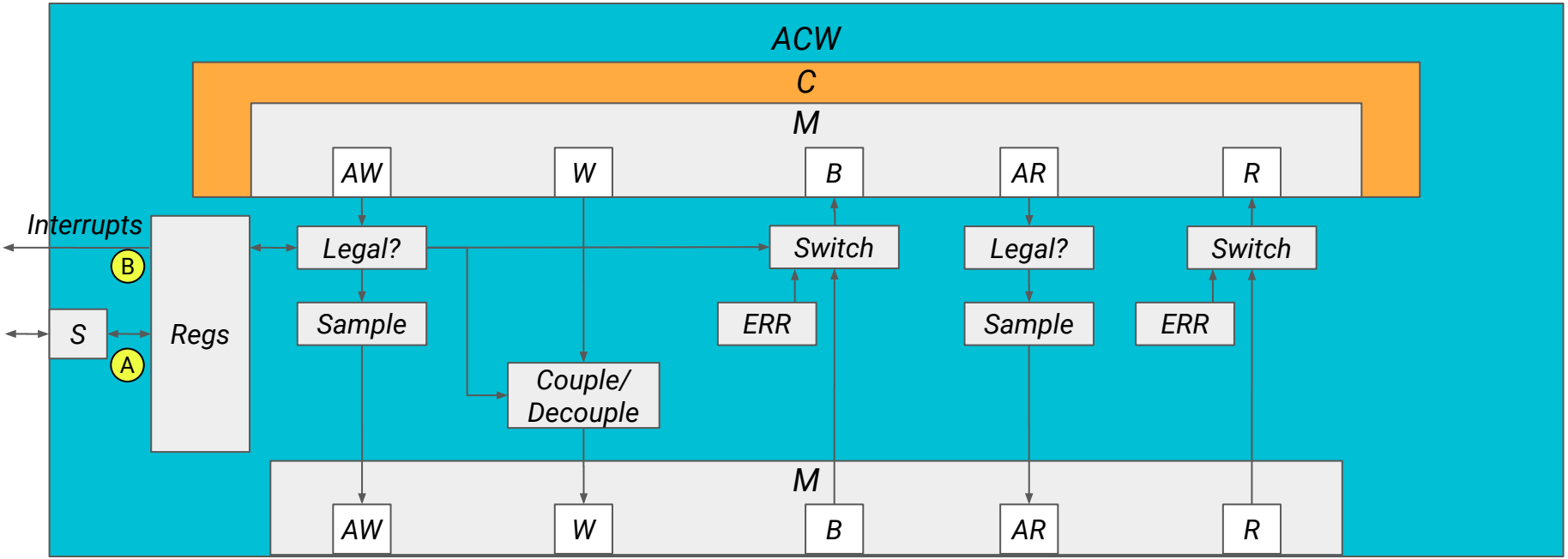
These aided in identifying two groups of weaknesses:

- Group A: TE <-> ACW AXI ports
- Group B: Interrupts line from ACW to TE

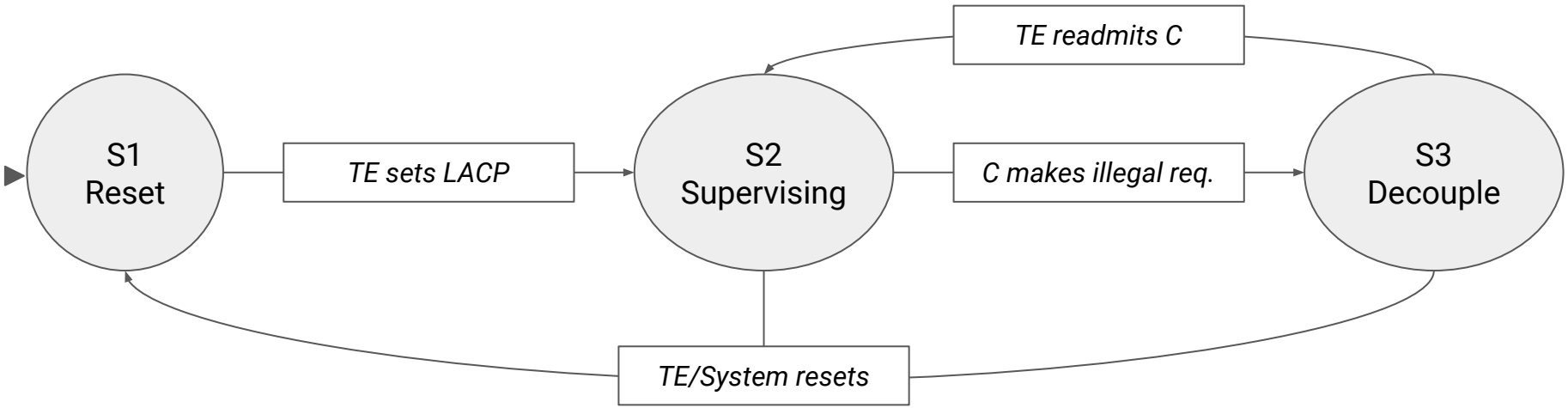


# 2. Identify Potential Weaknesses

The TE <-> ACW AXI ports and the interrupt lines from ACW to TE are shown below.



# 3. Define Security Requirements *(ACW Modes)*




*LACP = local access control policy*

# 3. Define Security Requirements *(ACW Modes)*

Below is an overview of the intended behavior of the ACW design along with its modes

| Mode | Description | Security Implication for C |
|---|---|---|
| Active Reset (AR) | The ACW is being reset to its default state and is not serving requests. | The ACW should block controller C from sending or receiving information |
| Reset (R) | The ACW does not have a valid configuration and is not serving requests. | The ACW should block controller C from sending or receiving information. |
| Supervising (S) | The ACW does have a valid configuration set and is serving/monitoring requests. | The ACW should allow controller C to send/receive information according to the set configuration. |
| Decouple (D) | The ACW does have a valid configuration set and is not serving (new) requests because an illegal request has been detected. | The ACW should block controller C from sending or receiving information. |

# 3. Define Security Requirements

Requirements related to the firmware used to dynamically configure the ACW.

| SP No. | Related CWEs | SP Requirement | ACW Mode | Weakness Group |
|---|---|---|---|---|
| 5 | 1269, 1272, 1280 | The TE can read from but not write to anomaly registers. | All | A |
| 9 | 1221, 1258, 1259, 1267, 1269, 1271, 1272, 1274, 1280, 1282 | The configuration/anomaly registers contain the default values until they are modified by the TE (config.) and/or ACW (illegal req. metadata tracking). | All | A |
| 15 | 1221, 1272, 1280 | An interrupt to TE is generated after the ACW detects an illegal request. | D | B |

# 4. Specify Security Properties *(SP Types)*

Security properties fall into one of two categories: information flow* and trace properties.

**Sample Information Flow Property**

Information from C originating during reset mode will never flow to P.

```
‘sig_from_C‘     (source)
when (RST == 0) (tainting condition)
=/=>            (no-flow operator)
‘sig_to_P‘       (destination)
```

**Sample Trace Property**

C receives the baseline AXI signals while the ACW is in reset mode.

```
‘sig_to_C‘ == ‘val‘ (value check)
|| (RST != 1)       (condition)
```

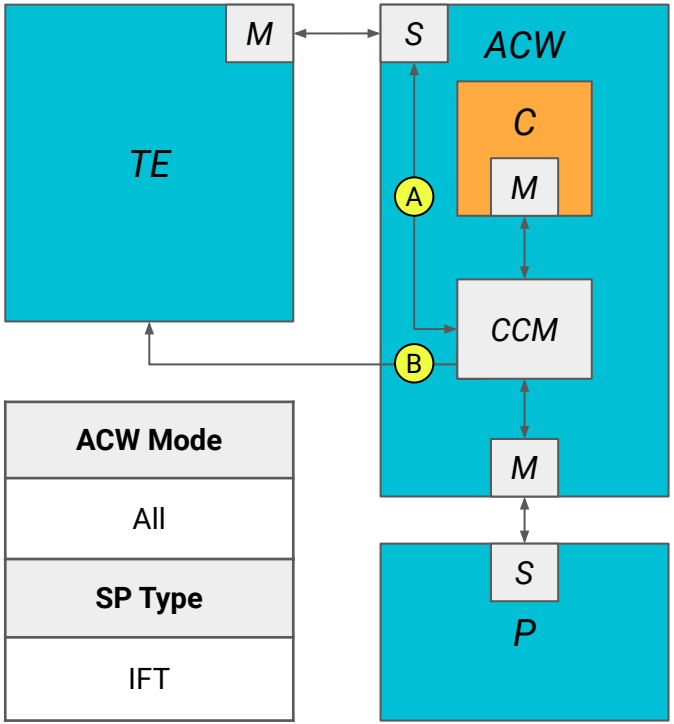
** Requires information flow tools**

# 4. Specify Security Properties *(SP 5)*

**Security Property - 5:**

The TE can read from but not write to anomaly registers.

```
//# of Generic Signals to Replace = 3
SP05_RONLY_GENERIC: assert iflow(
  `signal_from_TE`
   when (S_AXI_CTRL_AWADDR == `reg_addr`)
   =/=>
  `anomaly_reg`
);
```

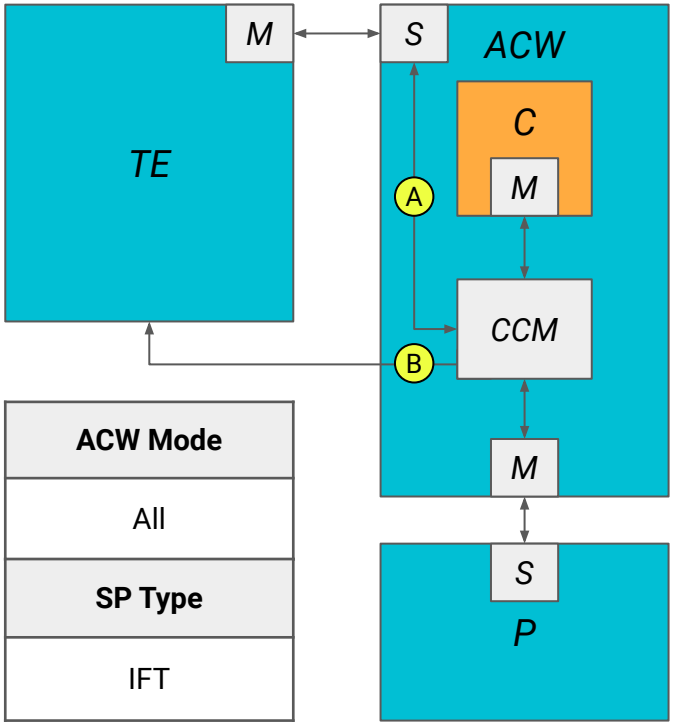


# 4. Specify Security Properties *(SP 9)*

**Security Property - 9:**

The configuration/anomaly registers contain the default values until they are modified by the TE (config.) and/or ACW (illegal req. metadata tracking).

```
//# of Generic Signals to Replace = 3
SP09_DEFAULT_GENERIC: assert iflow(
   `unauthorized_signal`
    when (`reg` == `default_value`)
    =/=>
   `reg`
    unless (`reg` == `default_value`)
);
```

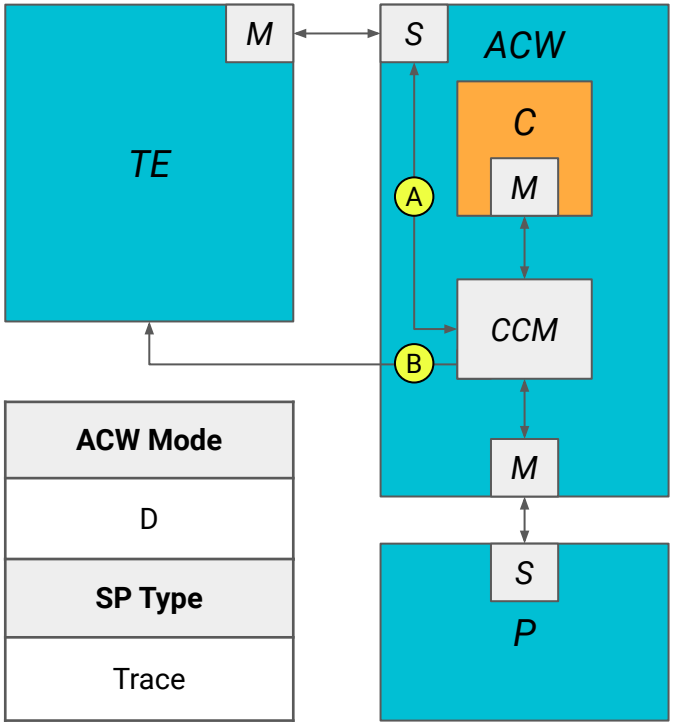


# 4. Specify Security Properties *(SP 15)*

**Security Property - 15:**

An interrupt to TE is generated after the ACW detects an illegal request.

```
//# of Generic Signals to Replace = 2
SP15_R_INTERRUPT: assert iflow(
  `INTR_LINE_W/R` == 1
   unless (`acw_w/r_state` != 2'b10)
);
```

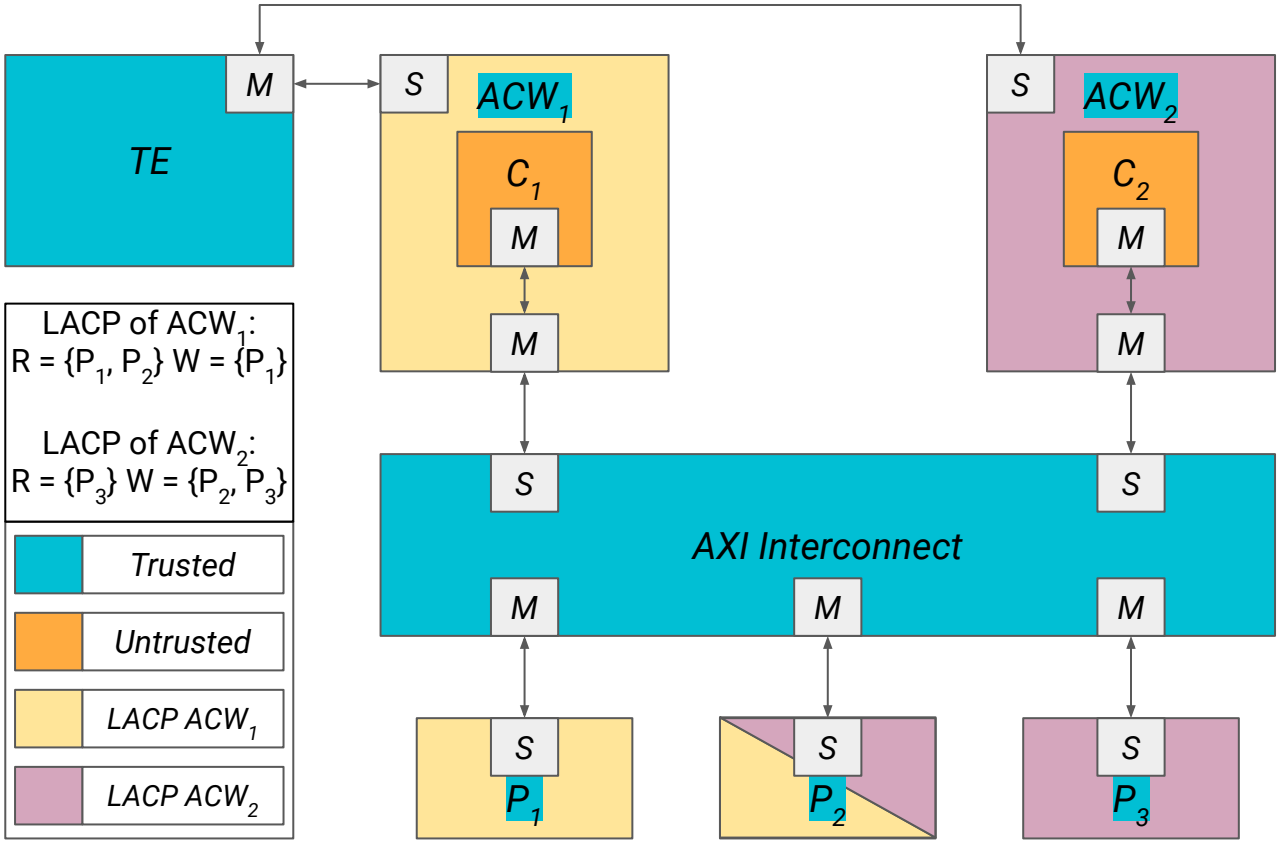


# ACW

System Level Security Verification

*Verifying the security of a simple SoC with multiple dynamically-configured access control wrappers.*

# Multi-ACW SoC for System Level Verification



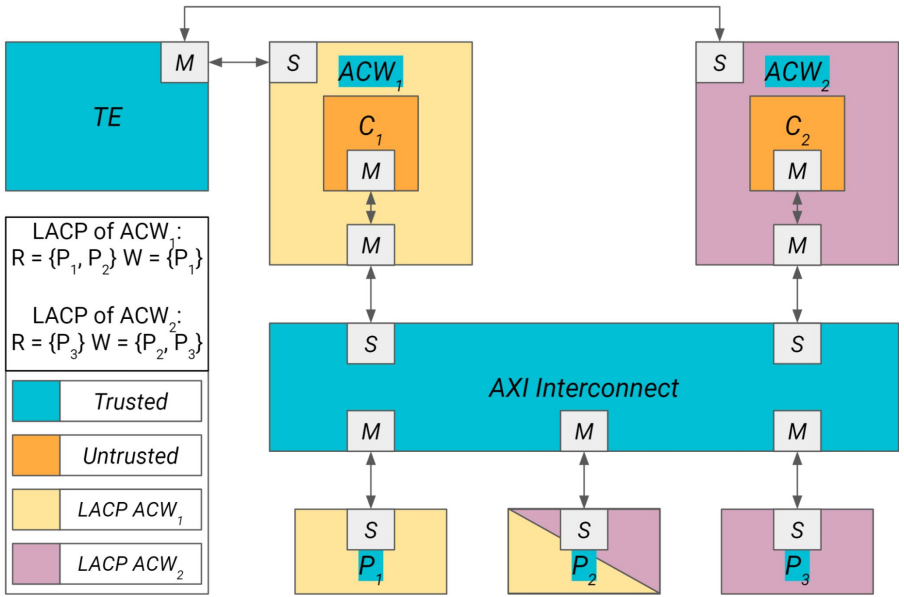
# 1. Create Threat Model

At the firmware level, there are four entities that we are concerned with:

- the controllers (C)
- the access control wrapper (ACW)
- the peripherals (P)
- the hardware root of trust (TE)

We consider an integrity scenario where Cs is untrusted and the ACW, Ps, and TE are trusted.

Therefore, the threat model considers C's ability to communicate with any P via the ACW in a manner which does not adhere to its dynamically-configured LACP.

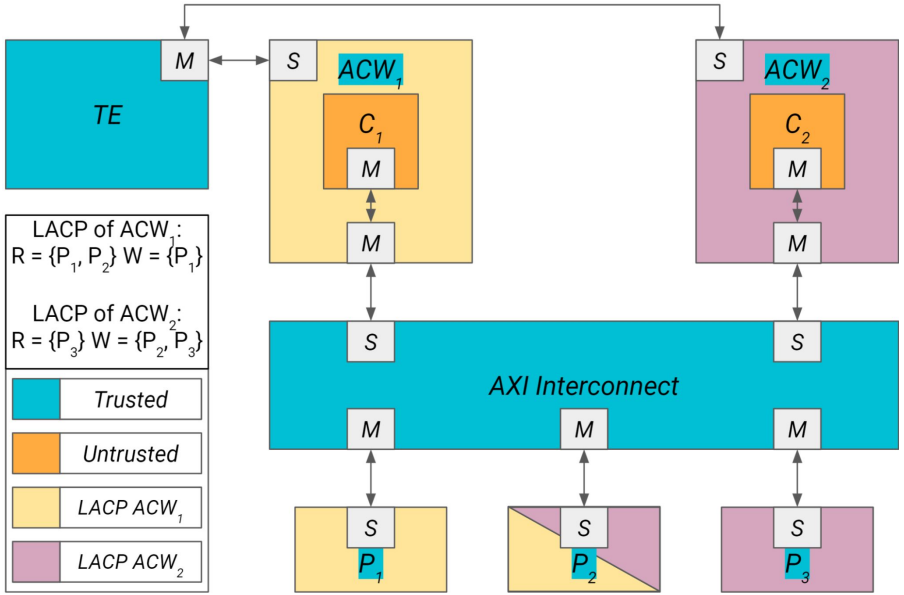


# 2. Identify Potential Weaknesses

We identified the following 3 relevant CWEs:

441, 1189, 1260

The identified potential weaknesses include the manner in which the LACP for $ACW_i$ is set as it relates the generic $C_i$ sharing resources with the generic $C_j$ .



# 3. Define Security Requirements

Requirements related to the firmware used to dynamically configure the ACW.

| SP No. | Related CWEs | SP Requirement | ACW Mode |
|---|---|---|---|
| 16 | 441, 1189, 1260 | Any C cannot receive/send data from/to any region not contained within its ACW's LACP. | All |

# 4. Specify Security Properties *(SP Types)*

Security properties fall into one of two categories: information flow* and trace properties.

**Sample Information Flow Property**

Information from C originating during reset mode will never flow to P.

```
‘sig_from_C‘     (source)
when (RST == 0) (tainting condition)
=/=>            (no-flow operator)
‘sig_to_P‘       (destination)
```

**Sample Trace Property**

C receives the baseline AXI signals while the ACW is in reset mode.

```
‘sig_to_C‘ == ‘val‘ (value check)
|| (RST != 1)       (condition)
```

** Requires information flow tools**

# 4. Specify Security Properties *(SP 16)*

**Security Property - 16:**

Any C cannot receive/send data from/to any region not contained within its ACW's LACP.

```
//# of Generic Signals to Replace = 2
SP16_RECEIVE_GENERIC: assert iflow(
`unauthorized`
 =/=>
`sig_from_C`
);

//# of Generic Signals to Replace = 2
SP16_SEND_GENERIC: assert iflow(
`sig_from_C`
 =/=>
`unauthorized`
);
```