Security Annotation for Electronic Design Integration Standard

July 2021
Rev 1.0

Abstract: The standard is collateral-centric with a focus on security concerns; it applies to electrical designs that are integrated into other circuits. The standard defines a methodology that (1) identifies elements, such as input or output ports, that can influence the behavior of a critical section within the design and (2) associates known security weaknesses based on the type of design and/or critical section. The methodology uses data objects, which are both human and machine readable, to capture security relevant information through the architectural and design phase of the electrical design to be consumed by an Integrator for their product lifecycle. The standard is independent of existing standards and is not part of the electrical design itself.

Keywords: Security, RTL, attack surface, threat modeling, security weakness, mitigation, hardware, circuit design, integrated circuit, SoC, ASIC, IP

Notices

Participants

At the time this draft standard was completed, the Accellera IPSA Working Group had the following membership:

Brent Sherman, Intel Corp, *Chair*
Mike Borza, Synopsys, *Vice Chair*

Sohrab Aftabjahani, Intel Corp.
Adam Cron, Synopsys
Monica Farkash, AMD
Nicole Fern, Tortuga Logic
Dave Graubart, Allied Member
John Hallman, OneSpin Solutions
Kathy Hayashi, Qualcomm, Inc.
Nathan Mandelke, Cadence Design Systems, Inc.
Jean-Philippe Martin, Intel Corp.
Steven McNeil, Xilinx, Inc.
Michael Munsey, Methodics, Inc.
Anders Nordstrom, Tortuga Logic
James Pangburn, Cadence Design Systems, Inc.
Ambar Sarkar, NVIDIA Corp.
Yaron Schiller, Cadence Design Systems, Inc.
Adam Sherer, Cadence Design Systems, Inc.
Ireneusz Sobanski, Intel Corp.
Badhri Uppiliappan, Analog Devices, Inc.
Jesse Wyant, Intel Corp.

Introduction

A System on Chip (SoC) or Application Specific Integrated Circuit (ASIC) is comprised of multiple components referred to as Intellectual Property (IP) blocks or just IP. These blocks come from multiple sources such as internal development teams, IP suppliers, tools to generate IP, etc. Typically, the SoC/ASIC owner integrates multiple IPs from multiple sources, which raises concerns about security risk. This standard addresses these concerns by introducing a methodology and formalized data objects that identify security risks an Integrator might inherit. These concerns may be addressed by an Integrator to make an informed decision at the time of IP integration. The options may be to select another IP with less risk, implement mitigations to address the concerns, or simply decide the risks are out of scope for the product.

The methodology uses two approaches to identify security concerns. One is to identify attack points that can be used to compromise the security of the IP block. These attack points are what an adversary would use to perform a malicious act on the design. By presenting this information, the Integrator can decide how to manage the associated risks. The other approach is to identify and associate known security concerns to an IP block. These concerns have been discovered, classified and published by fellow travelers in the industry, academia, or security researchers. Anyone researching security may be able to contribute to a knowledge base.

The standard is primarily directed towards IP developers and integrators. It is design, product, and tool independent. Users of this standard will be able to provide consistent security collateral in a uniform format.

Contents

Security Annotation for Electronic Design Integration Standard

1. Overview

The SA-EDI standard defines a specification that documents security concerns for hardware IP and its associated components when integrated into an Integrated Circuit. With the standard, IP Providers can identify security concerns to either: 1) mitigate in their IP; or 2) disclose to the Integrator to address at their level.

The standard is structured as follows: Section 4 introduces a background on the existing IP development process; Section 5 describes the proposed methodology to support the SA-EDI standard; Section 6 introduces the concept of a security weakness knowledge base which is comprised of known security concerns; Section 7 outlines the data objects in the standard; Section 8 is the threat model which is the end result; and Section 9 provides guidelines for compliance to the standard. The Annex sections provide additional information to help use the standard: Annex A outlines the JSON schema for the data objects; Annex B provides an example application to an IP; and Annex C contains the source code of the example IP.

The standard is completely contained in this document and any references to whitepapers such as [B1] are for background information only and are not considered part of the standard.

1.1 Scope

The standard defines data objects to identify critical elements in a digital hardware IP design and their associated security concerns. It defines the format and relationship of data objects that may be generated by tools during the hardware development process. Since the standard is external to the IP design, it can be applied to existing designs even if the hardware source (e.g., RTL) is encrypted.

The standard assumes the relationship between the IP Provider and Integrator is trusted. The standard does not address issues such as supplier credentialing; it simply provides a mechanism for an IP Provider to identify security concerns to an Integrator. Secure integration requires (among other things) that IP suppliers act in good faith by providing complete collateral.

1.2 Purpose

The intent of the standard is to identify known security concerns, documented in a knowledge base, associated with an asset and/or family type during IP integration. The IP Provider uses the standard to identify assets in the design that require a security objective (e.g., Confidentiality, Integrity, Availability) and elements (e.g., ports, parameters, etc.) that can compromise the objective. The Integrator uses the collateral to create a threat model with identified mitigations to support the security objective. The methodology provides the means to validate security assurance collateral to an IP design.

1.3 Word usage

The word *shall* indicates mandatory requirements strictly to be followed in order to conform to the standard and from which no deviation is permitted (shall equals is required to).

The word *should* indicates that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required (should equals is recommended that).

The word *may* is used to indicate a course of action permissible within the limits of the standard (may equals is permitted to).

The word *can* is used for statements of possibility and capability, whether material, physical, or causal (can equals is able to).

2. Normative references

The following referenced documents, if any are listed in this section, are indispensable for the application of this document (i.e., they must be understood and used, so each referenced document is cited in text and its relationship to this document is explained). For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

3. Definitions, acronyms, and abbreviations

3.1 Definitions

For the purposes of this document, the following terms and definitions apply.

Adversary: A malicious entity that prevents security objectives from being achieved

Asset: Anything of value or importance that is used, produced, or protected within the IP

Attack Point: An access location or means through which a threat can be realized against an asset

Attack Surface: A set of attack points (can be applied to multiple assets).

Concern (Consequence): The potential harm that a threat poses to an asset

Fully Qualified Name: In Verilog, a design element with its module name. Format: <*module_name*>.<*asset_name*>. In VHDL, a design element with its component name. Format: <*component_name*>.<*asset_name*>. Other languages may have corresponding notations.

Integrated Circuit: An electronic design (e.g. SoC, ASIC, etc.) that consists of multiple IPs

Integrator: The entity that integrates IP into an electronic design

IP: Intellectual Property – The RTL or other design representation that is the subject of this standard

IP Bundle: The collateral that is supplied by the IP Provider which contains everything an Integrator needs to incorporate the IP

IP Provider: The entity that supplies an IP

Mitigation: A solution that reduces the risk or consequence of an attack

RTL: Register-Transfer Level – A design abstraction that models a digital circuit

Security Objective: A measurable way to achieve a security goal. For example, a security goal may be "protect an asset". A security objective would be "Confidentiality" on that asset as a means of protection. This standard identifies Confidentiality, Integrity, and Availability [B3] as security objectives.

Threat (Attack): Anything that can potentially adversely affect an asset

Threat Model: A collection of threats that are in scope for an electronic design

Vulnerability: A weakness in the IP that could be exploited

Weakness: A way in which an IP fails to protect an asset

3.2 Acronyms and abbreviations

ADC	Analog-Digital Converter
AES	Advanced Encryption Standard
API	Application Programming Interface
APIC	Advanced Programmable Interrupt Controller
APSO	Attack Points Security Objective
ASIC	Application Specific Integrated Circuit
BIST	Built-In Self-Test
CDMA/GSM	Code Division Multiple Access / Global System for Mobile
CIA	Confidentiality, Integrity, Availability
CISC	Complex Instruction Set Computer
CPU	Central Processing Unit
CWE	Common Weakness Enumeration

DAC	Digital-Analog Converter
DDR	Double Data Rate
DRAM/SRAM	Dynamic Random-Access Memory / Static Random-Access Memory
DSP	Digital Signal Processor
EDA	Electronic Design Automation
EEPROM	Electrically Erasable Programmable ROM
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GPIO	General Purpose Input/Output
GPS	Global Positioning System
GPU	General Processing Unit
HDL	Hardware Description Language
HMAC	Keyed-Hash Message Authentication Code
I2C	Inter-IC bus
IC	Integrated Circuit
IP	Intellectual Property
IPSA	IP Security Assurance (Workgroup)
JSON	JavaScript Object Notation
JTAG	Joint Test Action Group
LSB	Least Significant Bit
MMC	Memory Management Controller
MSB	Most Significant Bit
NoC	Network on Chip
NVRAM	Non-Volatile RAM
OTP	One-Time Programmable
PCIe	Peripheral Component Interconnect Express

PHY	Physical layer
RISC	Reduced Instruction Set Computer
RNG	Random Number Generator
ROM	Read-Only Memory
RSA	Rivest, Shamir, and Adelman
RTL	Register-Transfer Level
SA-EDI	Security Annotation for Electronic Design Integration
SHA	Secure Hash Algorithm
SoC	System on Chip
SWKB	Security Weakness Knowledge Base
TPU	Tensor Processing Unit
URI	Uniform Resource Identifier
USB	Universal Serial Bus
VHDL	Very High Speed Integrated Circuit Hardware Description Language

4. Background

In today's IP development and delivery process, there's no standard guidance in security assurance. At the basic level, an IP is defined based on standards such as Verilog, VHDL, etc. that are compiled and synthesized using EDA tools to produce outputs such as netlists, place & route databases, etc. As it pertains to the standard in this document, the focus of discussion is the IP design (i.e. RTL), gate-level netlist, and any testbench that's produced. At a high level, the workflow is shown in Figure 1.

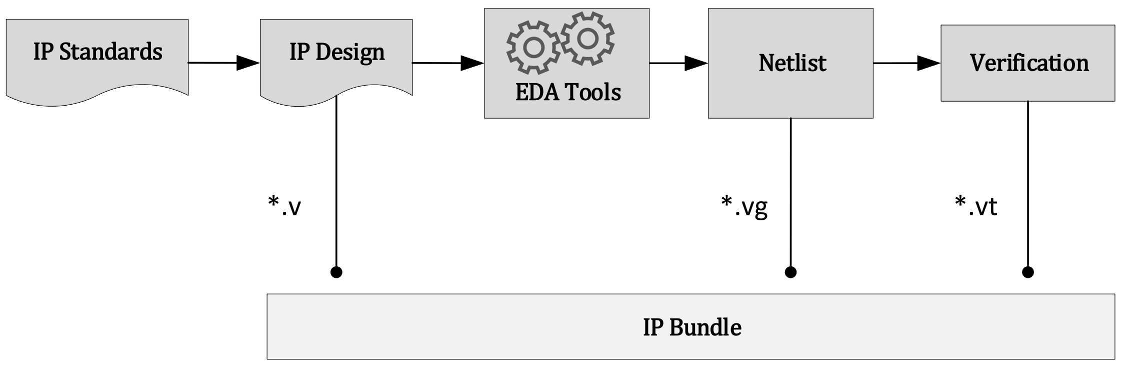

Figure 1, Existing IP Workflow

The file extensions shown are examples only to establish the context of what is contained in an IP Bundle. Additionally, the "bundle" may contain files such as documentation, executables, configuration files, etc. This bundle is what is being offered as an IP for the Integrator to incorporate into their product (e.g. SoC), as shown in Figure 2.

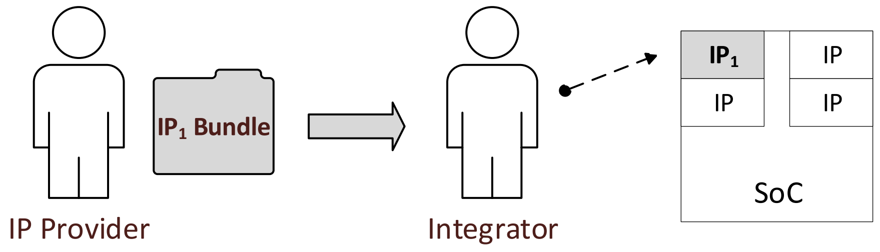

Figure 2, IP Delivery

The Integrator will unpack the IP Bundle to extract its contents and execute simulation tests to prove it is functionally sound. After performing initial sanity checks or functional verification, the Integrator incorporates the IP into the SoC. Once integrated, additional tests are performed to verify the IP is behaving properly with other IPs in the product. The process is repeated for each IP that is integrated into the SoC.

There is a notable gap in the IP development and delivery workflow, which does not include a statement of security concerns that an Integrator inherits or introduces when accepting an IP from a provider. This standard will provide guidance as to what IP security assurance collateral is needed and how the collateral should be consumed to close this gap.

5. SA-EDI Methodology

The standard introduces new collateral into the IP bundle which is shown in Figure 3. This collateral includes data objects that represent assets, database, elements, and attack points and security objectives. These objects are discussed in detail in section 7. As additions, they can be added to an existing workflow without modification to the IP design.

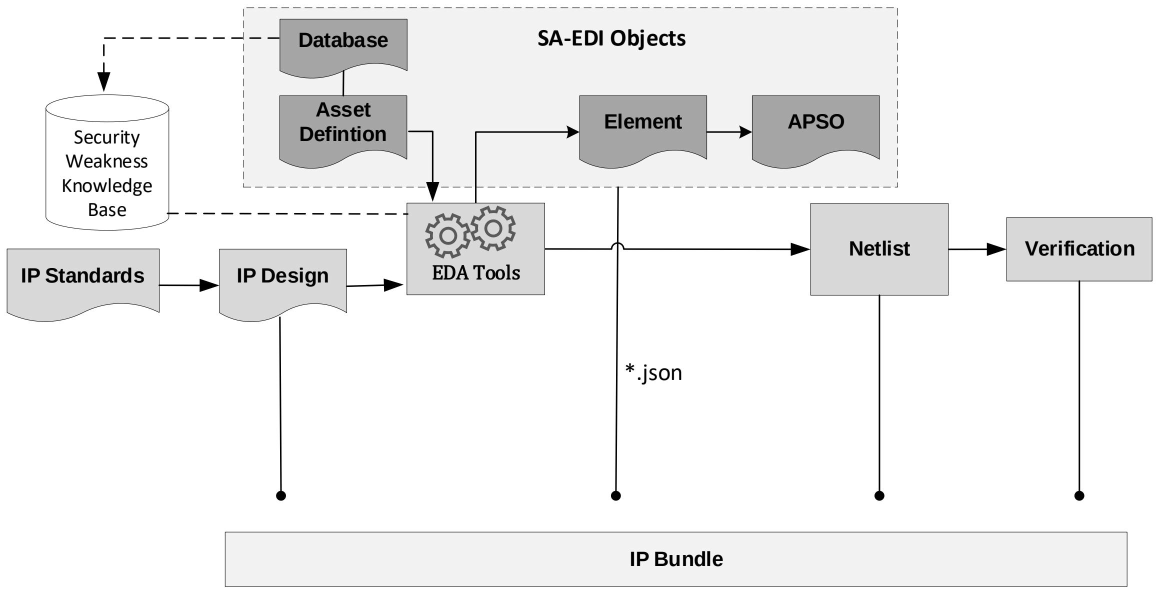

Figure 3, SA-EDI IP Bundle

The Asset Definition data object identifies critical or valued material within the IP. It also contains a Database data object that provides information about a Security Weakness Knowledge Base. These data objects are inputs into EDA tools that create Element data objects. The Element data objects contain ports and parameters that influence the behavior of the asset and include potential security weaknesses that are associated with the asset and/or IP type. The Element data objects are used to create Attack Points Security Objective (APSO) data objects. APSO data objects associate a security objective such as confidentiality, integrity, availability, to a list of ports or parameters thus creating an attack surface. These APSO data objects eventually build the threat model for the IP Integrator to use. The details of these objects are in later sections of this document.

The data objects in Figure 3 are shown as JSON format. This is detailed later in section 7.1.

It is also worth noting that the Element objects can be created manually without an EDA tool.

The database labeled “Security Weakness Knowledge Base” contains security weaknesses that are known due to industry experience and/or security researchers. The standard allows the use of multiple databases from multiple sources. The details of how such a knowledge base can be utilized are listed in a later section.

5.1 IP Bundle

Upon receiving the IP Bundle, the Integrator will use the new data objects to create a threat model that is specific to the integrated circuit with respect to the IP. This is shown in Figure 4. The Integrator will repeat some of the steps that the IP Provider performed in order to verify that the SA-EDI data objects were indeed derived from the IP definition. Performing this verification is optional, however highly recommended by the standard.

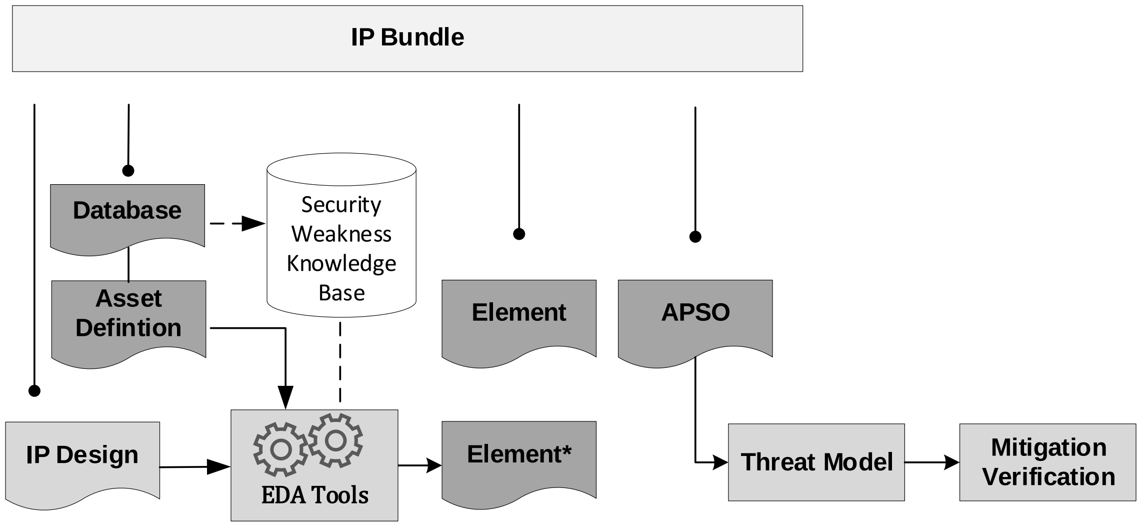

Figure 4, SA-EDI Integrator

The Integrator, using the Asset Definition and Database in the bundle and the same Security Weakness Knowledge Base, generates an Element data object. This object is labeled "Element*" in Figure 4 and should correspond to the same assets that were defined in the IP bundle. Once generated, the Integrator compares the contents of the Element* object to the contents of the Element object in the bundle. This comparison can be done by visual inspection or by using a tool. If the contents are the same, then the Integrator knows the security assurance collateral corresponds to the RTL and can proceed with integration. If the contents are not the same, then the Integrator and the IP Provider need to resolve the differences before integrating the IP into the IC. Mismatches may be caused by RTL changes after the Element object was generated or that there was an error in the generation of the Element object itself.

The Integrator then reviews the Attack Point Security Objective (APSO) data objects in the bundle to determine which ones are in scope for the IC. This will become the threat model which contains mitigations to be verified. The Integrator may also create additional APSO objects that are product specific for this IP. These additions become part of the IC's threat model for verification.

6. Security Weakness Knowledge Base

The Security Weakness Knowledge Base (SWKB) is a database or repository that contains security concerns that are associated with hardware IP and its integration. The term SWKB is generic and does not represent a specific database. It is instead used to reference existing databases such as the Common Weakness Enumeration (CWE) [B2] or even a proprietary database. The standard allows for the use of multiple databases in multiple locations. Additionally, the SWKB should support an API that allows for software queries in order to aid in automation.

The standard requires that a SWKB support searchability on IP and asset categories or types. This requirement makes it possible for security weaknesses to be associated with a specific IP or asset. The IP family types are listed in Table 1. Along with the types are definitions and examples to help provide more clarification about the IP family. This table has the capability to support additional IP family types, which can then be shared with Integrators to preserve the associations and methodology workflow.

The family types defined in Table 1 are intended to be high-level, generic classifications. They should not be used as detailed descriptions of the IP itself. Therefore, an IP may be classified by several family types and the standard does not prohibit assigning multiple family types to a single IP.

Table 1, IP Family Types

#	Name	Definition	Examples
1	Accelerator	IP dedicated to offload a specific workload to enhance performance	DSP, TPU, packet processing, mathematical, compression
2	Analog & Mixed-Signal	IP that controls or senses the electricals for communication, which receives or transmits signals conditioned outside of a system's digital domain	PHY, ADC, DAC
3	Audio/Video	IP designed to manipulate audio/video data	Coders/Decoders, speech recognition, format converters
4	Bus/Interface	IP implementing an interconnect among elements in and/or within a computing system	I2C, PCIe, DDR, MMC, USB, GPIO, AXI
5	Communications	IP designed to transmit/receive information	Modulator/Demodulator, 802.11, Bluetooth, CDMA/GSM
6	Controllers	A circuit hard-wired (e.g. Finite State Machine) to react in a closed-loop control system or other limited context, to control another entity	Arbiter, APIC, USB, Peripheral, Memory, Storage
7	Counter/Timer	IP reflecting the passage of time in oscillations or human units	Real Time Clock, Watchdog, Monotonic Counter
8	Memories	Volatile (transient) data storage	DRAM, SRAM
9	Microcontroller	A specialized processor acting as a programmable controller	8051, Nios
10	Power Management	IP which controls and/or monitors the power state of a system	Voltage regulators, power controllers or monitors
11	Processors	A programmable computing engine	CPU, GPU, TPU
12	Security	IP designed to protect assets	Cryptography, authorization, tamper detection, access controls, RNG
13	Storage	non-volatile (permanent) data storage	EEPROM, eFuse, flash, ROM, OTP, NVRAM
14	Test/Debug	IP designed to verify functionality and identify root cause of defects	JTAG, BIST, boundary scan, pattern generator
15	Transducers	IP which converts energy from one form to another, such as physical to electrical	sensors, actuators
16	<User Defined>	This type is used to accommodate families that have not been defined in this table (e.g. proprietary IP). To add a family, the value should have the prefix "UD:".	UD: *CustomIP*

Table 2 shows the classification types for assets. These types provide more information about the asset (e.g., what makes it an asset) and can be used to associate additional security weaknesses to the IP itself. Similar to the IP family types, the asset types are intended to be high-level, generic classifications in which several may be used to describe a single asset.

Table 2, Asset Type

#	Name	Definition	Examples
1	Critical	Material that is critical for proper functionality. Without this asset, the IP would not be able to function.	Timers/Counters, clock generators
2	Secret	Material that requires confidentiality and should not be accessible outside the IP	Password, cryptographic keys
3	Sensitive	Material that requires integrity but not necessarily confidentiality.	Root of Trust (e.g. Asymmetric public key), fuse/OTP
4	Control	Material used to alter and/or control the state of the IP. This material can also setup or configure the IP.	FSM, control register
5	Cryptographic	Material that is part of a cryptographic operation	AES, RSA, SHA, HMAC, RNG
6	Code/Data	Material that contains information which can alter the behavior of the IP	Storage (Volatile/Non-volatile)
7	Compute	Material that is part of an execution engine that operates on opcodes or instructions	CISC, RISC, CPU, GPU
8	<User Defined>	This type is used to accommodate asset types that have not been defined in this table (e.g. proprietary IP). To add an asset type, the value shall have the prefix "UD:".	UD: *CustomIP*

6.1 Format

To support the standard, a SWKB shall provide the following attributes for each entry:

a) **Title**: A brief and high-level description about the weakness, normally a single sentence or phrase.

b) **Reference number**: A unique identifier within the knowledge base. It will be used in the Element data object (Section 7.4) to reference a specific entry.

c) **Description**: A detailed description of why the weakness is a problem or concern. It may include possible unwanted behaviors, affected resources, etc.

d) **Consequence**: A classification of the risk(s) due to the weakness as confidentiality, integrity, and/or availability. The impact of the consequence should be captured as well.

e) **Applicability**: A list of IP families (Table 1) and/or asset types (Table 2) that may be impacted by the security weakness.

f) **Modes of Introduction**: A list of lifecycle phases in which the weakness could have been introduced. Some examples are architecture, design, implementation, integration, manufacturing or provisioning, etc.

g) **Mitigations**: This attribute lists techniques that are intended to minimize the severity of the weakness. The attribute should include relevant lifecycle phases in which mitigations can be introduced. See reference f) above.

Table 1 and Table 2 are used alone or in combination to associate security weaknesses to an Asset Definition data object. In addition, the standard allows the use of any keywords or text in the data fields of an entry.

6.2 Specifications

The rules are as follows:

a) The SWKB database should reference IP Family types as shown in Table 1, column "Name" into the appropriate entries by string value.

b) The SWKB database should reference Asset Functionality types as shown in Table 2, column "Name" into the appropriate entries by string value.

7. Data Objects

The data objects (Asset Definition, Database, Element, and Attack Points Security Objective) and SWKB are linked via attributes as shown in Figure 5. Please note that the associated attributes between the data objects are shown and not the complete list of attributes in each object. The variable *n* in the diagram represents one or more objects and not equivalence or a specific value. The Asset Definition object uses the attribute *Database_ID* to reference a SWKB(s). This reference is linked to the attribute *ID* of the Database object. The Database object defines the properties of a SWKB. The Asset Definition object uses *Family* and *Type* attributes to identify entries in the SWKB that match the values in (e) in section 6.1. The Element object uses *Security Weakness Reference* attribute to link to those entries in the SWKB. This attribute matches the values in (b) in section 6.1. The Asset Definition and Element objects are linked by the *Name* attribute as defined in the Asset Definition object.

The Element object is used to create the Attack Points Security Objective (APSO) data object and the attributes *Asset Name*, *Ports*, *Parameters*, and *Security Weakness Reference* are the associations between them. The value in the *Asset Name* attribute matches the *Name* attribute of the Asset Definition object. The value(s) in the *Ports* attribute correlates (may not be 1:1) to the value(s) in the *Attack Points* attribute of the APSO object.

There may be circumstances in which an Element object is not created, however there is still a need to create an APSO object to identify a security objective to an asset. In this case, the APSO object may be created from the Asset Definition object and the attribute *Asset Name* will be associated to *Name*, respectively.

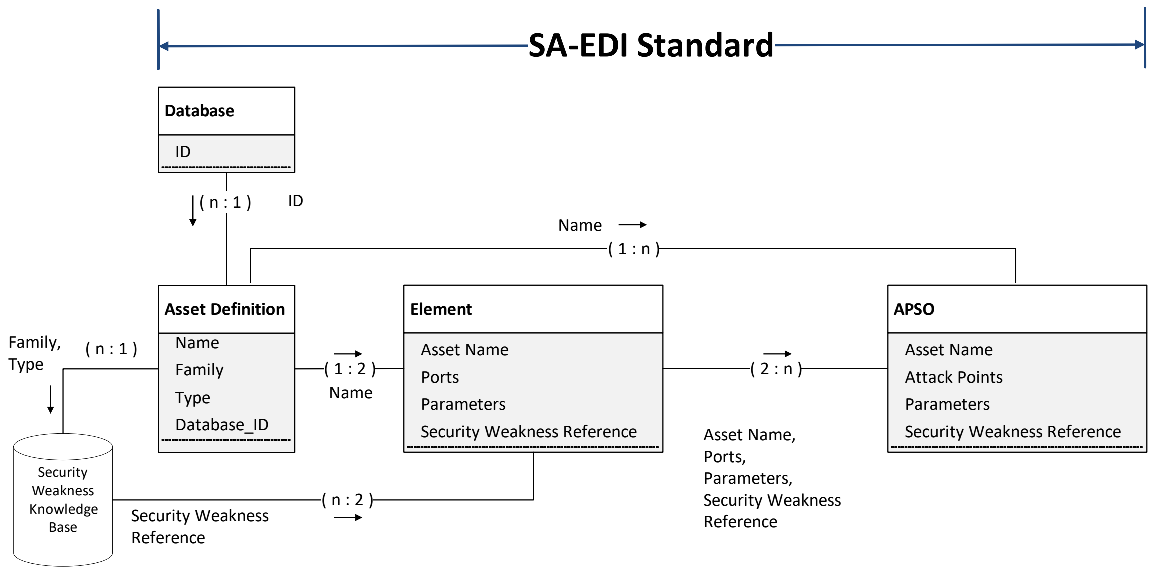

Figure 5, SA-EDI Associations

7.1 Data Object Language

Data objects shall be machine readable and should be human readable. The standard uses JavaScript Object Notation (JSON)[B4] as its data modeling language. JSON was chosen due to its adaptability and small footprint for easier documentation. The examples use JSON 2019-09. However, any version greater than or equal to Draft 4 can be used since required field capabilities were introduced in Draft 4, which is needed to support attributes that are required by the standard.

The JSON schema for each of the data objects are defined in section Annex A. The schema may be extended to support future attributes and/or specific use-cases. For simplicity, data objects and objects are equivalent throughout the standard.

7.2 Asset Definition

The Asset Definition data object is the critical dependency in the standard. All other data objects are derived from this object. Therefore, defining assets correctly is crucial to completing a proper threat model.

The Asset Definition object is used to identify assets within the IP. An asset is anything of value or importance that is critical to proper behavior which require security objective protections. An asset can be identified as a port, module, register, or another object in the design. The paper [B1] provides more information, along with examples, about how to possibly identify assets within an IP. In addition, there’s a use-case example in Annex B that highlights the complete methodology.

Once an asset is identified, its definition is comprised of the attributes defined in Table 3. The attribute *Name* is used to reference the asset in RTL and shall match its corresponding text in the source. Each asset will have its own Asset Definition data object. The attributes are provided by the IP Developer and will be used later to create the Element data object.

Table 3, Asset Definition Data Object

Attribute	Required	Type	Definition
Name	Yes	String (case-sensitive[1])	Full hierarchical path name of the asset as defined in the RTL source
Description	No	String	Brief description about the asset (e.g., what makes it an asset, its purpose, etc.). This is not a required field however it is strongly recommended since it provides useful information to the IP Integrator.
Family	Yes	Array of Strings	Describes the IP type or family. The values are listed in Table 1. The value may be the numeric string or string name. There may be more than one type that is applicable.
Type	Yes	Array of Strings	Describes the asset type. The values are listed in Table 2. The value may be the numeric string or string name. There may be more than one type that is applicable.
Database_ID	No	Array of Strings	Reference to a SWKB. The string should match the attribute value of *ID* in Table 4. This is an array to support multiple databases.

7.2.1 Specifications

The rules are as follows:

a) An IP may have multiple assets.

b) An Asset Definition object shall reference a single asset.

c) If the asset is an array, it is assumed the entire array is the asset unless a specified range is included in the *Name* attribute.

d) If an asset is in several ranges of an array, then each range shall have its own Asset Definition object.

7.3 Database

The Database data object is used to provide details about a security weaknesses database that is to be used in the methodology flow. A Database object is associated to an Asset Definition object via the *ID* attribute in Table 4. Since the methodology supports the use of multiple databases, there may be many Database objects associated to an Asset Definition object.

The Database object is not required if a security weaknesses database is not used.

[1] Case-sensitivity may be dependent on the language of the RTL source.

Table 4, Database Data Object

Attribute	**Required**	**Type**	**Definition**
ID	Yes	String	A unique identifier that is associated to a SWKB. This may be the name of the database. This attribute is referenced in the Asset Definition object in the *Database_ID* attribute (Table 3)
Description	No	String	Brief description about the database (e.g. how to use it, types of weaknesses, etc.)
URI	Yes	String	URI locator of the security weaknesses database
Version	Yes	String	Version identifier of the security weaknesses database

7.3.1 Specifications

The rules are as follows:

a) Every SWKB version shall have at least one Database object associated with it.

b) A Database object may be associated with one or more Asset Definition objects.

7.4 Element

The Element data object is created when Asset Definition object(s) are defined. An Asset Definition object provides enough information for a tool (e.g., EDA) to generate Element objects. An Element object defines the top module influencers (i.e., elements) of the IP that can affect and/or observe the behavior of the asset. These elements may include input/output ports and/or configuration parameters in the RTL. These are access points that either: 1) an adversary can use to affect the asset's behavior, or 2) an Integrator needs to take into consideration to ensure proper protections are in place.

An Element object is associated with an Asset Definition object via the Asset Identifier, which is defined in section 7.2. Every Asset Definition object shall have at least one associated Element object. Element objects are categorized by the attribute *Direction* shown in Table 5. This attribute represents the direction of influence for the *Ports* attribute. Therefore, the signals listed in *Ports* shall all be in one direction. If a port is bidirectional, it may be listed in both the "Input" and "Output" Element objects.

Table 5, Element Data Object

Attribute	Required	Type	Definition
Asset Name	Yes	String	Reference to the attribute *Name* as defined in Table 3
Direction	Yes	Enumeration	Defines the direction of the *Ports* attribute: 1. Input 2. Output
Security Weakness Reference	No	Array of Strings	Security weakness reference(s) from the SWKB. The format is dependent on the format of the entries in the database.
Ports	Yes	Array of Strings (case-sensitive[2])	Ports exposed at the integration level that influence or observe the behavior of an asset
Parameters	No	Array of Strings (case-sensitive[2])	Configuration parameters in the RTL that are associated with the asset. Since parameters are language dependent, the text should match the syntax of the language.

7.4.1 Specifications

The rules are as follows:

a) An Element object shall reference only one Asset Definition object.

b) No more than one “Input” Element object shall reference the same Asset Definition object.

c) No more than one “Output” Element object shall reference the same Asset Definition object.

d) The *Asset Name* attribute must match the text, including case, in the attribute *Name* in the Asset Definition object.

e) If multiple Database objects are defined in the Asset Definition object then each entry in the attribute *Security Weakness Reference* shall include the value of *ID* in Table 4.

7.5 Attack Points Security Objective (APSO)

The Attack Points Security Objective (APSO) data object is the starting point for the Integrator to understand the inherited security concerns and objectives. The intent of the APSO object is to assign a security objective to an attack surface of an asset and any conditions that may violate that objective. It may be derived directly from Element objects or an Asset Definition object if there are side-channel concerns to address. The supported security objectives are Confidentiality, Integrity, and Availability which are aligned with the definitions in the NIST SP 800-100 handbook[B3]. The APSO object may include applicable security weakness references identified in the Element object(s).

An APSO object may be created without an association to an Element object. An asset may lack a fan-in and/or fan-out that reaches the IP boundary. In this case, there will be no need for an Element object, but there may be security objectives pertaining to the asset, for which an APSO object is required.

An example could be the entropy source of a random number generator (RNG) integrated in the IP. This entropy source might not be exposed to the integration layer via a port, so there will be no Element object associated to the Asset Definition object. However, the asset may still require a security objective (e.g., Integrity), therefore an APSO object may be created with the *Attack Points* attribute empty. These types of

[2] Case-sensitivity may be dependent on the language of the RTL source.

APSO objects are used to identify implicit security concerns such as side-channel or injection attack points associated with a particular asset.

An IP Provider may create APSO objects that address security objectives external to the IP. These objects are intended to provide additional integration guidance. For example, an asset's port may have requirements to support a security objective, such as Availability. In this case, the *Description* attribute could recommend that "this port should be directly connected to the IC's reset logic and not gated by any logic". This object provides additional guidance on how the IP should be integrated.

Table 6, APSO Data Object

Attribute	Required	Type	Definition
Name	Yes	String	Unique identifier for the security objective that is associated with this *Asset Name*. The *Name* need not be unique across multiple assets.
Asset Name	Yes	String	Reference to the *Name* attribute in Table 3
Security Objective	Yes	Enumeration	Describes the security objective required for the asset. There should only be one security objective identified per APSO object. 1. Confidentiality 2. Integrity 3. Availability
Description	No	String	Additional information about the security objective
Condition	No	SVA expression	Condition under which the security objective is violated, expressed in SystemVerilog Assertion (SVA) syntax. An example may be a lock bit, which protects the integrity of a register, not being enabled. All RTL signals used in the expression should be qualified such that it can be evaluated at the IP top level.
Security Weakness Reference	No	Array of Strings	Reference to *Security Weaknesses Reference* attribute identified in Table 5.
Additional Security Weaknesses	No	Array of Strings	Additional weaknesses that were not identified in attribute *Security Weakness Reference*. These can be newly discovered or use-case/customer specific weaknesses.
Attack Points	No	Array of Strings	Ports listed in Table 5 that are associated with this security objective
Parameters	No	Array of Strings	Configuration parameters listed in Table 5 that are associated with this security objective

7.5.1 Specifications

The rules are as follows:

a) The combination of *Name* and *Asset Name* shall be unique.

b) An APSO object shall have exactly one Security Objective defined.

c) An APSO object shall apply to exactly one Asset Definition object.

d) An APSO object may have no associated Element objects, in which case the attribute *Attack Points* shall be empty.

8. Threat Model

The next step in the methodology is the creation of a threat model for the IC. This is performed by the Integrator and may be created from applicable APSO objects. The APSO objects may come from both the IP Provider and the Integrator.

APSO objects are based on the architecture and design of the IP as a standalone component. Additionally, the IP Provider may have created APSO objects based on potential use-cases of the IP in an integrated circuit such as an SoC. When the Integrator examines the APSO objects, some may not be relevant to the IC. For example, there may be an APSO object that addresses confidentiality concerns on the counter of a watchdog timer, the counter being the asset. The concern is that by leaking the count value, an attacker could gain an advantage (e.g. the length of time remaining to launch an exploit). This may not be relevant to the security of the IC. If the watchdog is being used for boot ROM execution and gets disabled when this execution is finished, confidentiality is probably not an objective due to the limited agents that are out of reset at the time. Therefore, this APSO object would not apply to the use-case of the IC.

Once the Integrator has evaluated which IP level APSO objects are in scope for the IC, the next step is to identify which IC level APSO objects are relevant to the integration of the IP. Using the watchdog example, the Integrator may add an APSO object that pertains to the availability of the watchdog's reset assertion due to a timeout. The object would have the security objective Availability to ensure that there is no gating logic on the watchdog reset.

When the IC level APSO objects have been created, the integration Threat Model is complete for the IP. The standard does not define the format of a threat model beyond the APSO data object definition. This allows the flexibility of converting APSO objects into other formats that align with an industry or company-specific verification process.

9. Workflow Compliance

The intent of this section is to state the responsibilities of both the IP Provider and Integrator in the workflow. A compliant IP Bundle includes the applicable Asset Definition, Database, Element, and APSO data objects. See section 7.5 for cases where an Element object is not required. See section 7.3 for cases where a Database object is not required. Table 7 shows the steps of the workflow. Steps #1-5 can be followed by an IP Provider to create a compliant IP Bundle. Steps #6-11 can be followed by an Integrator to integrate and verify a compliant IP Bundle.

Table 7, Workflow

Step#	Owner	Details	Output
1	IP Provider	Identify a database of known weaknesses and create a Database object(s)	Table 4, Database Data Object
2	IP Provider	Identify asset(s) in the IP and create an Asset Definition object(s) for each asset	Table 3, Asset Definition Data Object
3	IP Provider	Using the output of step #1 and #2, generate the Element object(s) using an EDA tool. This step may also be done manually.	Table 5, Element Data Object
4	IP Provider	Using the output of step #3, create Attack Points Security Objective object(s)	Table 6, APSO Data Object
5	IP Provider	Bundle all the data objects created in steps #1-4 in the IP delivery package	IP Bundle
6	Integrator	Using the output of step #5, repeat step #3 to regenerate the Element object(s) in a file for comparison. This requires that the Integrator has access to the RTL source. If the Integrator is using an EDA tool, it should be functionally equivalent to the tool used in step #3. The output of this step will be used to verify the accuracy of the Element objects with respect to the RTL source.	Table 5, Element Data Object
7	Integrator	Using the output of steps #5-6, the Integrator compares the locally generated Element objects to those from the IP Bundle. This compare can be done by visual inspection or by a tool. If the contents of objects are the same, then report SUCCESS. This means the Element objects are consistent with those in the IP Bundle. Otherwise, report FAILURE and stop the workflow.	SUCCESS or FAILURE[3]
8	Integrator	Using the output of step #5, determine which APSO objects are in scope for the IC.	Threat Model
9	Integrator	Using the output of step #5, create any additional APSO objects from the Element objects that have security objectives at the IC level as it pertains to the IP. Additional security weaknesses may be identified also.	Table 6, APSO Data Object
10	Integrator	If there are any APSO objects created in step #9, add them to the IC threat model	Threat Model
11	Integrator	Using the output of step #10, verify the security objectives are met and the security weaknesses are properly addressed during integration of the IP into the IC.	Verification

[3] If FAILURE, the Integrator can either choose an equivalent IP from a different supplier or have a discussion with the IP Provider to address the discrepancies.

Annex A : Data Object JSON Schema

This section contains JSON schemas for the data objects defined in this standard, validated against the referenced JSON schema standard.

A.1 Asset Definition

```
{
  "$schema": "http://json-schema.org/draft/2019-09/schema",
  "title": "Asset Definition",
  "description": "An asset is something that's critical for proper IP operation",
  "type": "object",
  "properties": {
     "Name" : { "type" : "string" },
     "Description" : { "type" : "string" },
     "Family" : { "type" : "array", "items" : {"type" : "string"} },
     "Type" : { "type" : "array", "items" : {"type" : "string"} },
     "Database_ID" : { "type" : "array", "items" : {"type" : "string"} }
     },
  "required" : ["Name", "Family", "Type"]
}
```

A.2 Database

```
{
  "$schema": "http://json-schema.org/draft/2019-09/schema",
  "title": "Database",
  "description": "Information that defines a security weaknesses database",
  "type": "object",
  "properties": {
     "ID" :  { "type" : "string" },
     "Description" : { "type" : "string" },
     "URI" : { "type" : "string" },
     "Version" : { "type": "string" }
     },
  "required" : ["ID", "URI", "Version"]
}
```

A.3 Element

```
{
  "$schema": "http://json-schema.org/draft/2019-09/schema",
  "title": "Element",
  "description": "An element is a relationship to the asset, directly or indirectly",
  "type": "object",
  "properties": {
     "Asset Name" :  { "type" : "string" },
     "Direction" : { "type" : "string", "enum": ["Input", "Output", "None"] },
     "Security Weakness Reference" : { "type" : "array", "items" : {"type" : "string"} },
     "Ports" : { "type" : "array", "items" : {"type" : "string"} },
     "Parameters" : { "type" : "array", "items" : {"type" : "string"} }
     },
  "required" : ["Asset Name", "Direction", "Ports"]
}
```

A.4 Attack Points Security Objective

```
{
  "$schema": "http://json-schema.org/draft/2019-09/schema",
  "title": "Attack Points Security Objective",
  "description": "Attack points with associated security objective",
  "type": "object",
  "properties": {
     "Name" : { "type" : "string" },
     "Asset Name" : { "type" : "string" },
     "Security Objective" : { "type" : "string",
                "enum": [
                    "Confidentiality",
                    "Integrity",
                    "Availability"
                ] },
     "Description" : { "type" : "string" },
     "Condition" : { "type" : "string" },
     "Security Weakness Reference" :  {"type" : "array", "items" : {"type" : "string"} },
     "Additional Security Weaknesses": {"type" : "array", "items" : {"type" : "string"} },
     "Attack Points" :  {"type" : "array", "items" : {"type" : "string"} },
     "Parameters" : { "type" : "array", "items" : {"type" : "string"} }
     },
"required" : ["Name", "Asset Name", "Security Objective"]
}
```

A.5 SA-EDI Data Object

The SA-EDI data object may be used to collect the standard's data objects into a single JSON file. This is optional, however if provided in the IP Bundle this schema shall be used.

```
{
  "$schema": "http://json-schema.org/draft/2019-09/schema",
  "definitions" : {
    "ASSET" : {
                "title": "Asset Definition",
                "description": "An asset is something of importance",
                "type": "object",
                "properties": {
                   "Name" : { "type" : "string" },
                   "Description" : { "type" : "string" },
                   "Family" : { "type" : "array", "items" : {"type" : "string"} },
                   "Type" : { "type" : "array", "items" : {"type" : "string"} },
                   "Database_ID" : { "type" : "array", "items" : {"type" : "string"} }
                },
                "required" : ["Name", "Family", "Type"]
               },
     "DATABASE" : {
                "title": "Database",
                "description": "Information that defines a security weaknesses database",
                "type": "object",
                "properties": {
                   "ID" :  { "type" : "string" },
                   "Description" : { "type" : "string" },
                   "URI" : { "type" : "string" },
                   "Version" : { "type": "string" }
                },
                "required" : ["ID", "URI", "Version"]
                },
     "ELEMENT" : {
                "title": "Element",
                "description": "An element is a relationship to the asset",
                "type": "object",
                "properties": {
                   "Asset Name" : { "type" : "string" },
                   "Direction" :{ "type" : "string", "enum": ["Input", "Output", "None"] },
                   "Security Weakness Reference" : { "type" : "array", "items" : {"type" :
                   "string"} },
                   "Ports" : { "type" : "array", "items" : {"type" : "string"} },
                   "Parameters" : { "type" : "array", "items" : {"type" : "string"} }
                },
                "required" : ["Asset Name", "Direction", "Ports"]
                },
     "APSO" : {
```

```
              "title": "Attack Points Security Objective",
              "description": "Attack points with associated security objective",
              "type": "object",
              "properties": {
                 "Name" : { "type" : "string" },
                 "Asset Name" : { "type" : "string" },
                 "Security Objective" : { "type" : "string",
                             "enum": [
                                     "Confidentiality",
                                     "Integrity",
                                     "Availability"
                                     ] },
                 "Description" : { "type" : "string" },
                 "Condition" : { "type" : "string" },
                 "Security Weakness Reference" :  {"type" : "array", "items" : {"type" :
                 "string"} },
                 "Additional Security Weaknesses": {"type" : "array", "items" : {"type" :
                 "string"} },
                 "Attack Points" :  {"type" : "array", "items" : {"type" : "string"} },
                 "Parameters" : { "type" : "array", "items" : {"type" : "string"} }
              },
              "required" : ["Name", "Asset Name", "Security Objective"]
              }
  },
  "title": "SA-EDI Group Object",
  "description": "Used to save all SA-EDI data objects in a single .json file",
  "type": "object",
  "properties": {
     "Asset Definition" : {"type" : "array", "items" : {"$ref" : "#/definitions/ASSET"} },
     "Database" : {"type" : "array", "items" : {"$ref" : "#/definitions/DATABASE"} },
     "Element" : {"type" : "array", "items" : {"$ref" : "#/definitions/ELEMENT"} },
     "Attack  Points  Security  Objective":{"type"  :  "array",  "items"  :  {"$ref"  :
     "#/definitions/APSO"} }
     },
  "required" : ["Asset Definition", "Attack Points Security Objective"]
}
```

Annex B : Use-case Example

The intention of this section is to demonstrate how the standard can be applied to an example IP. The IP was crafted to be simple and minimalistic for easy comprehension. The IP is not intended to be functionally complete or optimal. The source code can be referenced in section Annex C.

B.1 Watchdog IP

The Watchdog IP (WDIP) is a simple timer that when it expires, will assert an output signal that can be used to put an IC into a known good state. The timer counts down from an initial value that is the concatenation of the REG_COUNT_HIGH and REG_COUNT_LOW registers. The block diagram of the WDIP is shown in Figure 6.

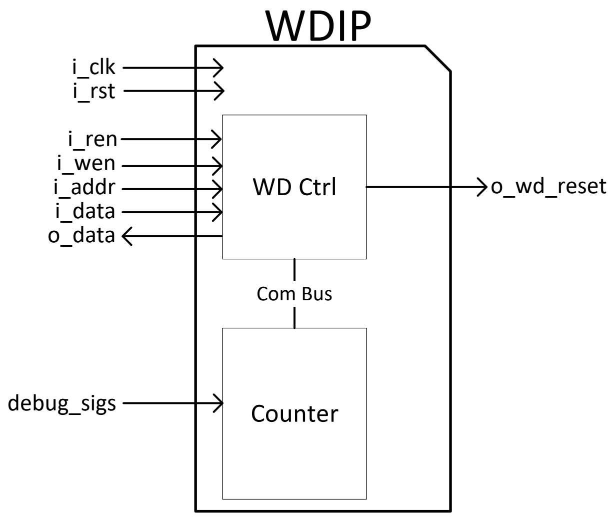

Figure 6, Watchdog Block Diagram

The WDIP consists of two basic blocks:

1. WD Ctrl: This controller is used to configure the watchdog settings, which includes enabling, disabling, and servicing the timer. The source for this block is provided in section C.2. The block supports a parallel bus that is sampled on a single clock cycle for both writes and reads. The bus consists of the following signals:

   a) *i_ren*: When asserted, enables read access to the register space in the WD Ctrl block.
   b) *i_wen*: When asserted, enables write access to the register space in the WD Ctrl block. If asserted during an *i_ren* assertion, a read will take place, (i.e., reads take precedence).
   c) *i_addr*: This is an 8-bit bus [7:0]. Represents the register address space inside the controller.
   d) *i_data*: This is an 8-bit bus [7:0]. It contains data to write to the targeted register on *i_addr* when *i_wen* is asserted.
   e) *o_data*: This is an 8-bit bus [7:0]. It contains the data read from the targeted register on *i_addr* when *i_ren* is asserted.

2. Counter: This block is the actual timer of the watchdog and communicates to the WD Ctrl block through the parallel "Com Bus". The source for this block is provided in section C.3. The "Com Bus" is internal to the watchdog IP and is defined as follows.

    a) count_val: This is a 16-bit input [15:0] that is used as the initial value of the timer.
    b) wd_start: Used to start the timer.
    c) wd_service: Used to service the timer (i.e. reset the count).
    d) wd_pause: Used to pause the timer.
    e) wd_timer: This is a 16-bit output [15:0] that represents the current timer value.
    f) wd_timout: Counter has reached zero.
    g) clk: Clock. It is connected to *i_clk*.

    The Counter block also supports the following input debug signals on the "debug_sigs" interface which are exposed to the top module. During debug mode, the debug signals override the WD Ctrl block signals.

    a) *i_dbg_enable*: Used to put the WDIP into debug mode.
    b) *i_dbg_clk_en*: Once asserted, the debug clock will be used instead of *i_clk*.
    c) *i_dbg_clk*: Debug clock.
    d) *i_dbg_cnt_val*: This is a 16-bit input [15:0] that is used as the initial value of the timer.
    e) *i_dbg_timout*: Asserts the timeout.
    f) *i_dbg_pause*: Used to pause the timer.
    g) *i_dbg_start*: When asserted, the timer is running.
    h) *i_dbg_service*: Services the timer to reset the count.

B.1.1 Registers

The WD Ctrl block supports the following register interface to the top module.

1. REG_CONTROL (Address: 0x1)

Bit #	Access	Description
0	RW	Lock bit. Once set, REG_CONTROL, REG_COUNT_LOW, and REG_COUNT_HIGH can not be altered until either *i_rst* or *o_wd_reset* is asserted. • 0 – unlocked • 1 – locked
1	RW	Start. Once set, the timer will start counting down from the initial value. • 0 – disabled. The timer is cleared. • 1 – starts the timer
2	RW	Pause. Once set, the timer will pause. All state information is preserved. • 0 – continue timer • 1 – pause timer
3-7	-	Reserved

2. REG_SERVICE (Address: 0x2)

Bit #	Access	Description
0	W	Service bit. Once set, the timer will be reloaded from the initial values in REG_COUNT_LOW and REG_COUNT_HIGH registers. • 0 – nothing • 1 – serviced. This will be cleared on the next clock cycle.
1-7	-	Reserved

3. REG_COUNT_LOW (Address: 0x3)

Bit #	Access	Description
0-7	W	The lower byte of the initial value of the timer.

4. REG_COUNT_HIGH (Address: 0x4)

Bit #	Access	Description
0-7	W	The upper byte of the initial value of the timer.

5. REG_TIMER_LOW (Address: 0x5)

Bit #	Access	Description
0-7	R	The lower byte of the timer value

6. REG_TIMER_HIGH (Address: 0x6)

Bit #	Access	Description
0-7	R	The upper byte of the timer value

B.2 Workflow Steps

Using the WDIP as an example, the methodology outlined in Table 7 is as follows:

Step #1. Identify a database of known security weaknesses. In this example, the CWE database is used and the Database data object will be as such:

```
{
  "ID" : "CWE VIEW: Hardware Design",
  "Description" : "A community developed list of hardware weakness types",
  "URI" : "https://cwe.mitre.org/data/definitions/1194.html",
  "Version" : "4.3"
}
```

Step #2. Identify the asset(s). Inside the counter block (*wd_count.v*), the register *wd_timer* holds the timeout value of the watchdog. The watchdog functionality may be used to detect an undesirable condition in the IC. Therefore, an adversary would want to prevent this timeout from happening and may want to modify the value of the counter (e.g., increase or reset its value). This makes the *wd_timer* an asset to the IP. The Asset Definition data object is defined as such:

```
{
  "Name" : " wd_top.count_block.wd_count.wd_timer",
  "Description" : "Timer count status.  Critical for proper operation",
  "Family" : ["Counter/Timer", "Test/Debug"],
  "Type" : ["Control", "Critical"],
  "Database_ID" : ["CWE VIEW: Hardware Design"]
}
```

To clarify, “Test/Debug” was included in the *Family* attribute because the IP supports a debug interface. This value will help associate security concerns around debug from the CWE database.

Once a timeout occurs, it is critical that the indication (i.e., system reset) gets propagated out to the top module without any modification. This timeout is in the counter block and is a register defined as *wd_assert_timeout*. An adversary who gains control of or influence over this register can modify the behavior of the watchdog IP (e.g., block the assertion of output signal *o_wd_reset* or assert constantly to create a denial of service). Therefore, *wd_assert_timeout* is critical for proper operation which makes it an asset. The Asset Definition data object for this asset is as follows:

```
  {
    "Name" : " wd_top.count_block.wd_count.wd_assert_timeout",
    "Description" : "Timeout assertion signal.  Critical for proper operation",
    "Family" : ["Counter/Timer","Test/Debug"],
    "Type" : ["Control", "Critical"],
    "Database_ID" : ["CWE VIEW: Hardware Design"]
  }
```

Step #3. Generate the Element data objects. For the WDIP, there are four objects generated: two are associated with the *wd_timer* asset and two are associated with the *wd_assert_timeout* asset. The Element data objects are as follows.

```
  {
    "Asset Name" : "wd_top.count_block.wd_count.wd_timer",
    "Direction" : "Input",
    "Security Weakness Reference" : ["CWE-1244","CWE-1191","CWE-1234"],
    "Ports" : [
        "wd_top.i_rst",
        "wd_top.i_clk",
        "wd_top.i_ren",
        "wd_top.i_wen",
        "wd_top.i_data",
        "wd_top.i_addr",
        "wd_top.i_dbg_enable",
        "wd_top.i_dbg_clk_en",
        "wd_top.i_dbg_clk",
        "wd_top.i_dbg_pause",
        "wd_top.i_dbg_start",
        "wd_top.i_dbg_service",
        "wd_top.i_dbg_timeout",
        "wd_top.i_dbg_cnt_val" ],
    "Parameters" : ["wd_top.COUNT_SIZE"]
  }

  {
    "Asset Name" : "wd_top.count_block.wd_count.wd_timer",
    "Direction" : "Output",
    "Security Weakness Reference" : ["CWE-1244","CWE-1191","CWE-1234"],
    "Ports" : ["wd_top.o_data"],
    "Parameters" : ["wd_top.COUNT_SIZE"]
  }

  {
    "Asset Name" : "wd_top.count_block.wd_control.wd_assert_timeout",
    "Direction" : "Input",
    "Security Weakness Reference" : ["CWE-1244","CWE-1191","CWE-1234"],
    "Ports" : [
        "wd_top.i_rst",
        "wd_top.i_clk",
        "wd_top.i_ren",
        "wd_top.i_wen",
        "wd_top.i_data",
        "wd_top.i_addr",
        "wd_top.i_dbg_enable",
        "wd_top.i_dbg_clk_en",
        "wd_top.i_dbg_clk",
        "wd_top.i_dbg_pause",
        "wd_top.i_dbg_start",
        "wd_top.i_dbg_service",
        "wd_top.i_dbg_timeout",
        "wd_top.i_dbg_cnt_val"],
    "Parameters" : ["wd_top.COUNT_SIZE"]
  }

  {
    "Asset Name" : "wd_top.count_block.wd_control.wd_assert_timeout",
    "Direction" : "Output",
    "Security Weakness Reference" : ["CWE-1244","CWE-1191","CWE-1234"],
    "Ports" : ["wd_top.o_wd_reset "]
  }
```

Step #4. Create the APSO data objects. For the *wd_timer* asset, the Integrity security objective needs to be upheld since this is what an adversary would want to alter. To protect the integrity of the timer value, the IP provides a locking mechanism. Only when the lock is not asserted, can the timer be manipulated, which is captured in the *Condition* attribute. The APSO data objects for *wd_timer* are as follows:

```
  {
    "Name" : "SO_1",
    "Asset Name" : "wd_top.count_block.wd_count.wd_timer",
    "Security Objective" : "Integrity",
    "Description" : "If the lock bit is not enabled then the counter can be altered",
    "Condition" : "(wd_top.i_wen=1) && (wd_top.i_addr=REG_CONTROL) && (wd_top.i_data[0]=0)",
    "Security Weakness Reference" : ["CWE-1244","CWE-1191","CWE-1234"],
    "Attack Points" : [
        "wd_top.i_wd_rst",
        "wd_top.i_wd_clk",
        "wd_top.i_enb",
        "wd_top.i_wen",
        "wd_top.i_addr",
        "wd_top.i_data"],
    "Parameters" : ["wd_top.COUNT_SIZE"]
  }
```

APSO object “SO_2” requires the *Condition* of debug mode to be enabled to violate the security objective.

```
  {
    "Name" : "SO_2",
    "Asset Name" : "wd_top.count_block.wd_count.wd_timer",
    "Security Objective" : "Integrity",
    "Description" : "Debug signals can alter the counter",
    "Condition" : "wd_top.i_dbg_enable == 1",
    "Security Weakness Reference" : ["CWE-1244","CWE-1191","CWE-1234"],
    "Attack Points" : [
        "wd_top.i_dbg_enable",
        "wd_top.i_dbg_clk_en",
        "wd_top.i_dbg_clk",
        "wd_top.i_dbg_pause",
        "wd_top.i_dbg_start",
        "wd_top.i_dbg_cnt_val"],
    "Parameters" : ["wd_top.COUNT_SIZE"]
  }
```

The *wd_assert_timeout* asset requires the Integrity security objective. If the integrity was compromised, a spurious timeout action will be taken, which may cause unwanted behavior such as extend the timeout or cause a DoS. This can be done when the IP is in debug mode. The APSO data objects for the *wd_assert_timeout* are as follows:

```
  {
    "Name" : "SO_3",
    "Asset Name" : "wd_top.count_block.wd_count.wd_assert_timeout",
    "Security Objective" : "Integrity",
    "Description" : "Debug can assert a timeout at any time",
    "Condition" : "wd_top.i_dbg_enable == 1",
    "Security Weakness Reference" : ["CWE-1244","CWE-1191","CWE-1234"],
    "Attack Points" : [
        "wd_top.i_dbg_enable",
        "wd_top.i_dbg_timeout"],
    "Parameters" : ["wd_top.COUNT_SIZE"]
  }

  {
    "Name" : "SO_4",
    "Asset Name" : "wd_top.count_block.wd_count.wd_assert_timeout",
    "Security Objective" : "Integrity",
    "Description" : "Debug can assert a timeout by setting count value to 0",
    "Condition" : "wd_top.i_dbg_enable == 1",
    "Security Weakness Reference" : ["CWE-1244","CWE-1191","CWE-1234"],
    "Attack Points" : [
        "wd_top.i_dbg_enable",
        "wd_top.i_dbg_cnt_val"],
    "Parameters" : ["wd_top.COUNT_SIZE"]
  }
```

In this example, the *Output* Element object that is associated with the "*wd_timer*" was not used in the creation of the APSO objects. This is because there were no identified security objectives on the asset that are associated with the *Ports* in this object. This does not violate compliance to the standard.

Step #5. Create the IP Bundle. This will include the source code in Annex C, netlist and testbenches, and the SA-EDI data objects produced in Steps #1-4. The SA-EDI data objects may be organized into a JSON object as shown below by using the schema defined in section A.5. The IP Bundle is then delivered to the Integrator.

```
  {
    "Asset Definition" : [
        {
          "Name" : " wd_top.count_block.wd_count.wd_timer",
          "Description" : "Timer count status.  Critical for proper operation",
          "Family" : ["Counter/Timer","Test/Debug"],
          "Type" : ["Control", "Critical"],
          "Database_ID" : ["CWE VIEW: Hardware Design"]
        },
        {
          "Name" : " wd_top.count_block.wd_count.wd_assert_timeout",
          "Description" : "Timeout assertion signal.  Critical for proper operation",
          "Family" : ["Counter/Timer","Test/Debug"],
          "Type" : ["Control", "Critical"],
          "Database_ID" : ["CWE VIEW: Hardware Design"]
        }],

    "Database" : [
        {
          "ID" : "CWE VIEW: Hardware Design",
          "Description" : "A community developed list of hardware weakness types",
          "URI" : "https://cwe.mitre.org/data/definitions/1194.html",
          "Version" : "4.3"
        }],

    "Element" : [
        {
          "Asset Name" : "wd_top.count_block.wd_count.wd_timer",
          "Direction" : "Input",
          "Security Weakness Reference" : ["CWE-1244","CWE-1191","CWE-1234"],
          "Ports" : [
                  "wd_top.i_rst",
                  "wd_top.i_clk",
                  "wd_top.i_ren",
                  "wd_top.i_wen",
                  "wd_top.i_data",
                  "wd_top.i_addr",
                  "wd_top.i_dbg_enable",
                  "wd_top.i_dbg_clk_en",
                  "wd_top.i_dbg_clk",
                  "wd_top.i_dbg_pause",
                  "wd_top.i_dbg_start",
                  "wd_top.i_dbg_service",
                  "wd_top.i_dbg_timeout",
                  "wd_top.i_dbg_cnt_val" ],
          "Parameters" : ["wd_top.COUNT_SIZE"]
        },
        {
```

```
      "Asset Name" : "wd_top.count_block.wd_count.wd_timer",
      "Direction" : "Output",
      "Security Weakness Reference" : ["CWE-1244","CWE-1191","CWE-1234"],
      "Ports" : ["wd_top.o_data"],
      "Parameters" : ["wd_top.COUNT_SIZE"]
    },
    {
      "Asset Name" : "wd_top.count_block.wd_control.wd_assert_timeout",
      "Direction" : "Input",
      "Security Weakness Reference" : ["CWE-1244","CWE-1191","CWE-1234"],
      "Ports" : [
            "wd_top.i_rst",
            "wd_top.i_clk",
            "wd_top.i_ren",
            "wd_top.i_wen",
            "wd_top.i_data",
            "wd_top.i_addr",
            "wd_top.i_dbg_enable",
            "wd_top.i_dbg_clk_en",
            "wd_top.i_dbg_clk",
            "wd_top.i_dbg_pause",
            "wd_top.i_dbg_start",
            "wd_top.i_dbg_service",
            "wd_top.i_dbg_timeout",
            "wd_top.i_dbg_cnt_val"],
      "Parameters" : ["wd_top.COUNT_SIZE"]
    },
    {
      "Asset Name" : "wd_top.count_block.wd_control.wd_assert_timeout",
      "Direction" : "Output",
      "Security Weakness Reference" : ["CWE-1244","CWE-1191","CWE-1234"],
      "Ports" : ["wd_top.o_wd_reset "]
    }],

"Attack Points Security Objective": [
    {
      "Name" : "SO_1",
      "Asset Name" : "wd_top.count_block.wd_count.wd_timer",
      "Security Objective" : "Integrity",
      "Description" : "If the lock bit is not enabled then the counter can be altered",
      "Condition":"(wd_top.i_wen=1)&&(wd_top.i_addr=REG_CONTROL)&&(wd_top.i_data[0]=0)",
      "Security Weakness Reference" : ["CWE-1244","CWE-1191","CWE-1234"],
      "Attack Points" : [
              "wd_top.i_wd_rst",
              "wd_top.i_wd_clk",
              "wd_top.i_enb",
              "wd_top.i_wen",
              "wd_top.i_addr",
              "wd_top.i_data"],
      "Parameters" : ["wd_top.COUNT_SIZE"]
    },
    {
      "Name" : "SO_2",
      "Asset Name" : "wd_top.count_block.wd_count.wd_timer",
      "Security Objective" : "Integrity",
      "Description" : "Debug signals can alter the counter",
      "Condition" : "wd_top.i_dbg_enable == 1",
      "Security Weakness Reference" : ["CWE-1244","CWE-1191","CWE-1234"],
      "Attack Points" : [
              "wd_top.i_dbg_enable",
              "wd_top.i_dbg_clk_en",
              "wd_top.i_dbg_clk",
              "wd_top.i_dbg_pause",
              "wd_top.i_dbg_start",
              "wd_top.i_dbg_cnt_val"],
      "Parameters" : ["wd_top.COUNT_SIZE"]
    },
    {
      "Name" : "SO_3",
      "Asset Name" : "wd_top.count_block.wd_count.wd_assert_timeout",
      "Security Objective" : "Integrity",
      "Description" : "Debug can assert a timeout at any time",
      "Condition" : "wd_top.i_dbg_enable == 1",
      "Security Weakness Reference" : ["CWE-1244","CWE-1191","CWE-1234"],
      "Attack Points" : [
            "wd_top.i_dbg_enable",
            "wd_top.i_dbg_timeout"],
      "Parameters" : ["wd_top.COUNT_SIZE"]
    },
    {
      "Name" : "SO_4",
            "Asset Name" : "wd_top.count_block.wd_count.wd_assert_timeout",
            "Security Objective" : "Integrity",
            "Description" : "Debug can assert a timeout by setting count value to 0",
            "Condition" : "wd_top.i_dbg_enable == 1",
            "Security Weakness Reference" : ["CWE-1244","CWE-1191","CWE-1234"],
            "Attack Points" : [
                  "wd_top.i_dbg_enable",
                  "wd_top.i_dbg_cnt_val"],
            "Parameters" : ["wd_top.COUNT_SIZE"]
          }
      ]
    }
```

Step #6. Regenerate the Element objects. The Integrator extracts the Asset Definition objects from the IP Bundle. Using these objects, the Integrator repeats Step #3 to regenerate the Element objects.

Step #7. Verify the Element objects. The Integrator performs a file compare between the locally generated Element objects and the Element objects from the IP Bundle. If the objects do not match, the process stops with a report of FAILURE. In the case where they match, the Integrator has verified that the SA-EDI collateral in the IP Bundle corresponds with the provided RTL, yielding SUCCESS.

Step #8. Scope the Threat Model. The Integrator reviews the APSO objects that were included in the IP Bundle to see which ones are in scope for the IC. For example, the APSO object labeled "SO_2" may not be a concern if the debug ports are tied off to be disabled in the IC. However, if the debug ports are to be connected, then this object would be in scope.

Step #9. Create the Threat Model. There may be some specific security objectives relevant to integration of the WDIP block in the IC. As an example, the *o_wd_reset* signal should not be gated, and therefore requires the security objective Availability. The Integrator could add the following APSO object to the Threat Model. Notice that some of the optional attributes are not included in the object because their values are not needed.

```
{
  "Name" : "SO_5",
  "Asset Name" : "wd_top.count_block.wd_control.wd_assert_timeout",
  "Security Objective" : "Availability",
  "Description" : "The timeout assertion should never be gated",
  "Attack Points" : ["wd_top.o_wd_reset"]
}
```

Step #10. Complete the Threat Model. Add the five created APSO objects to the threat model for the IC. Since this is just an example, the IC threat model is not shown for simplicity reasons.

Step #11. Verify the Threat Model. The last step is to verify that the security objectives in the threat model are upheld in the architecture and design of the IC. For example, verify "SO_1" is true by trying to prevent a timeout assertion via the WD Ctrl block interface once the lock bit is set. Another example may be to verify that deprivileged agents in the IC do not have access to the debug signals for "SO_2". Other examples may exist, however, the verification process is out of scope of the standard.

B.2.1 WDIP Security Evaluation

The WDIP block functions as expected, meaning there are no identified security vulnerabilities in the module. However, the SA-EDI methodology did identify security concerns that could be potential issues in an IC. The IP implemented a protection mechanism that can be circumvented by the debug interface. The lock bit in the REG_CONTROL register prevents modifications to the counter once set. However, the protection logic does not extend to the debug interface. Therefore, if not addressed in the IC, this concern could lead to multiple vulnerabilities in the IC.

Annex C : WDIP Source Code

This section includes the source code for the watchdog IP architecture detailed in B.1. The source files are written in Verilog and are as follows:

- wd_top.v – top module
- wd_control.v – logic which manages the registers and the counter block. It also controls the assertion of the watchdog timeout signal.
- wd_count.v – logic which manages the timer itself and its debug signals

C.1 wd_top.v

```
module wd_top #(parameter COUNT_SIZE = 16)      // top module
 (
  output              o_wd_reset, //wd timeout, active high
  input               i_rst,   //reset, active low
  input               i_clk,   //sys clk

  input               i_wen,      //write enable
  input               i_ren,      //read enable
  input  [7:0]        i_data,     //input data to register
  output [7:0]        o_data,     //output data from register
  input  [7:0]        i_addr,     //register address

  input               i_dbg_enable,  //debug enable
  input               i_dbg_clk_en,  //debug clk enable
  input               i_dbg_clk,     //debug clk override
  input               i_dbg_timeout, //debug timeout assertion
  input               i_dbg_pause,   //debug temporarily stops timer
  input               i_dbg_start,   //debug starts/stops timer
  input               i_dbg_service, //debug services timer
  input[COUNT_SIZE-1:0] i_dbg_cnt_val //debug sets timer count
  );

  wire                  wd_timeout;   //wd timout
  wire [COUNT_SIZE-1:0] timer_status; //status of timer count
  wire [COUNT_SIZE-1:0] count_val;    //timer count value
  wire                  wd_start;     //starts timer
  wire                  wd_service;   //services timer
  wire                  wd_pause;     //pauses timer

    wd_control #(.COUNT_SIZE(COUNT_SIZE))
    control_block(
    .clk   (i_clk),        //in
    .i_wd_rst (i_rst),     //in

    .reg_address (i_addr), //address
    .reg_data_i (i_data),  //write data
    .reg_data_o (o_data),  //read data
    .reg_wr_enb (i_wen),   //write enable
    .reg_rd_enb (i_ren),   //read enable

    .wd_timeout (wd_timeout),   //in
    .timer_status(timer_status),//in
    .count_val (count_val),     //out
    .wd_start (wd_start),       //out
    .wd_service (wd_service),   //out
    .wd_pause (wd_pause),       //out
    .o_wd_reset (o_wd_reset)    //out
  );

  wd_count #(.COUNT_SIZE(COUNT_SIZE))
    count_block(
    .clk (i_clk),            //in
    .rst_n (i_rst),          //in

    .wd_timer (timer_status),   //out
    .wd_timeout (wd_timeout),   //out
    .count_val (count_val),     //in
    .wd_start (wd_start),       //in
    .wd_service (wd_service),   //in
    .wd_pause (wd_pause),       //in
    .i_dbg_enable (i_dbg_enable),   //in
    .i_dbg_clk_en (i_dbg_clk_en),   //in
    .i_dbg_clk (i_dbg_clk),         //in
    .i_dbg_timeout (i_dbg_timeout), //in
    .i_dbg_pause (i_dbg_pause),     //in
    .i_dbg_start (i_dbg_start),     //in
    .i_dbg_service (i_dbg_service), //in
    .i_dbg_cnt_val (i_dbg_cnt_val)  //in
  );
endmodule
```

C.2 wd_control.v

```
module wd_control #(parameter COUNT_SIZE = 16)
  (
  input            clk,          //clock
  input            i_wd_rst,     //reset

  input      [7:0] reg_address, //address
  input      [7:0] reg_data_i,  //data
  output     [7:0] reg_data_o,
  input            reg_rd_enb,  //read enable
  input            reg_wr_enb,  //write enabl0065

  input      [COUNT_SIZE-1:0] timer_status, //timer cnt status
  output reg [COUNT_SIZE-1:0] count_val,  //timer cnt value
  input             wd_timeout,   //timeout assertion
  output            wd_start,     //starts/stops timer
  output            wd_service,   //services timer (reset)
  output            wd_pause,     //temporarily pauses timer
  output            o_wd_reset    //timeout assertion reset
  );

  parameter REG_CONTROL    = 'd1;
  parameter REG_SERVICE    = 'd2;
  parameter REG_COUNT_LOW  = 'd3;
  parameter REG_COUNT_HIGH = 'd4;
  parameter REG_TIMER_LOW  = 'd5;
  parameter REG_TIMER_HIGH = 'd6;


  reg [7:0] reg_data;
  reg [7:0] reg_control;
  reg [7:0] reg_service;
  reg       reg_pause;

  wire      reg_read;
  wire      reg_write;

  wire      lock_flag;    //lock bit
  wire      start_flag;   //start bit
  wire      pause_flag;   //pause
  wire      service_flag; //service bit

  assign o_wd_reset = wd_timeout; //timeout assertion
  assign lock_flag = reg_control[0];
  assign start_flag = reg_control[1];
  assign pause_flag = reg_control[2];
  assign wd_start = start_flag; //start to cnt blk
  assign wd_pause = reg_pause; //pause to cnt blk
  assign service_flag = reg_service[0];
  assign wd_service = service_flag; //service to cnt blk
  assign reg_data_o = reg_data;
  assign reg_write = reg_wr_enb && ~reg_rd_enb;
  assign reg_read = reg_rd_enb;



  reg [7:0] reg_count_low;
  reg [7:0] reg_count_high;
  reg       reg_count_low_set;    //flag when count[7:0] is set
  reg       reg_count_high_set;   //flag when count[15:8] is set
  reg       reg_timer_done;       //flag when timer status is ready

  //
```

```
always @(posedge clk)
  if (~i_wd_rst)
  begin
      reg_data    <= 8'b0;
      reg_control <= 8'b0;
      reg_service <= 8'b0;
      reg_count_low <= 8'b0;
      reg_count_high <= 8'b0;
      reg_timer_done <= 1'b0;
      reg_count_low_set <= 1'b0;
      reg_count_high_set <= 1'b0;
      reg_pause <= 1'b0;
  end
  else
  begin
    if (reg_write || reg_read)
    begin
      reg_data <= 8'd0;
      case (reg_address)
          REG_CONTROL:    //RW
              if (reg_read)
                  reg_data <= reg_control;
              else if (reg_write && !lock_flag)
                  reg_control <= reg_data_i;

          REG_SERVICE:    //WO
              if (reg_write)
                  reg_service = reg_data_i;

          REG_COUNT_LOW:  //WO
              if (reg_write && !lock_flag) begin
                  reg_count_low <= reg_data_i;
                  reg_count_low_set <= 1'b1;
              end

          REG_COUNT_HIGH: //WO
              if (reg_write && !lock_flag) begin
                  reg_count_high <= reg_data_i;
                  reg_count_high_set <= 1'b1;
              end

          REG_TIMER_LOW: //RO
              if (reg_read) begin
                  reg_pause <= 1'b1;    //pause for 8bit reads
                  reg_data <= timer_status[7:0];
                  if (!reg_timer_done)
                      reg_timer_done <= 1'b1;
                  else begin
                      reg_pause <= 1'b0;   //all 16bits availabe, continue
                      reg_timer_done <= 1'b0;
                  end
              end

          REG_TIMER_HIGH: //RO
              if (reg_read) begin
                  reg_pause <= 1'b1;    //pause for 8bit reads
                  reg_data <= timer_status[15:8];
                  if (!reg_timer_done)
                      reg_timer_done <= 1'b1;
                  else begin
                      reg_pause <= 1'b0; //all 16bits avail, continue
                      reg_timer_done <= 1'b0;
                  end
              end

          default:
              ;   //do nothing
      endcase
    end

    if (reg_count_low_set && reg_count_high_set) begin
        reg_count_low_set <= 1'b0;  //clear the flags
        reg_count_high_set <= 1'b0;
    end

    if (service_flag) begin
        reg_service[0] <= 8'b0;
    end

    if (wd_timeout) begin
        reg_control <= 8'b0; //timeout so clear cntrl settings
      end
    end

    //send timer cnt to counter block
    always @(posedge clk)
      if (~i_wd_rst)
      begin
            count_val <= {COUNT_SIZE{1'b1}};
      end
      else
      begin
        if (reg_count_low_set && reg_count_high_set)
            begin
                count_val[7:0] <= reg_count_low;
                count_val[15:8] <= reg_count_high;
            end
      end

endmodule
```

C.3 wd_count.v

```
module wd_count #(parameter COUNT_SIZE = 16)
  (
  input                       clk,           //clock
  input                       rst_n,         //reset

  output reg [COUNT_SIZE-1:0] wd_timer,      //timer count status
  output                      wd_timeout,    //timeout assertion
  input      [COUNT_SIZE-1:0] count_val,     //timer start count
  input                       wd_start,      //starts/stops timer
  input                       wd_service,    //services timer
  input                       wd_pause,      //temporarily pauses timer
  input                       i_dbg_enable,  //enables debug mode
  input                       i_dbg_clk_en,  //enables debug clk override
  input                       i_dbg_clk,     //debug clk
  input                       i_dbg_timeout, //asserts timeout
  input                       i_dbg_pause,   //temporarily pauses timer
  input                       i_dbg_start,   //starts/stops timer
  input                       i_dbg_service, //services the timer
  input      [COUNT_SIZE-1:0] i_dbg_cnt_val  //timer start value
  );

  reg                   wd_assert_timeout;

  //debug interface insertion
  wire                  wd_clk_w     = (i_dbg_enable && i_dbg_clk_en) ? i_dbg_clk : clk;
  wire                  wd_start_w   = (i_dbg_enable) ? i_dbg_start   : wd_start;
  wire                  wd_service_w = (i_dbg_enable) ? i_dbg_service : wd_service;
  wire                  wd_pause_w   = (i_dbg_enable) ? i_dbg_pause   : wd_pause;
  wire [COUNT_SIZE-1:0] count_val_w  = (i_dbg_enable) ? i_dbg_cnt_val : count_val;

  assign wd_timeout = (i_dbg_enable) ? i_dbg_timeout : wd_assert_timeout;

  //timer/counter
  always @(posedge wd_clk_w)
    if (~rst_n)
    begin
        wd_timer <= 16'hFFFF;
    end
    else
    begin
        //watchdog setup
        case ({wd_start_w, wd_service_w, wd_pause_w})
            3'b100:
                if (wd_timer > 0)
                    wd_timer <= wd_timer - 1'b1;  //timer count
            3'b110:
                wd_timer <= count_val_w;          //reload timer (service)
            default:
                wd_timer <= wd_timer;             //pause
        endcase
    end

  //timeout detection
  always @(posedge wd_clk_w)
    if (~rst_n)
    begin
        wd_assert_timeout <= 1'b0;
    end
    else
    begin
        if (!wd_start_w) begin
            //watchdog is disabled, initialize
            wd_assert_timeout <= 1'b0;
        end else if (wd_timer==0) begin
            //assert timeout, if not reload timer (service)
            wd_assert_timeout <= ~wd_service_w;
        end
    end

endmodule  // wd_count
```

Bibliography

Bibliographical references are resources that provide additional or helpful material but do not need to be understood or used to implement this standard. Reference to these resources is made for informational use only.

[B1] Sherman, B., et al. IP Security Assurance Standard Whitepaper, Accellera, 2019. https://www.accellera.org/images/activities/working-groups/ipsa-wg/Whitepaper_IPSA_Sept_4_2019.pdf

[B2] Common Weakness Enumeration, https://cwe.mitre.org

[B3] FIPS 199: Standards for Security Categorization of Federal Information and Information Systems, NIST, 2004, https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.199.pdf

[B4] JSON Schema. The home of JSON Schema, https://json-schema.org/