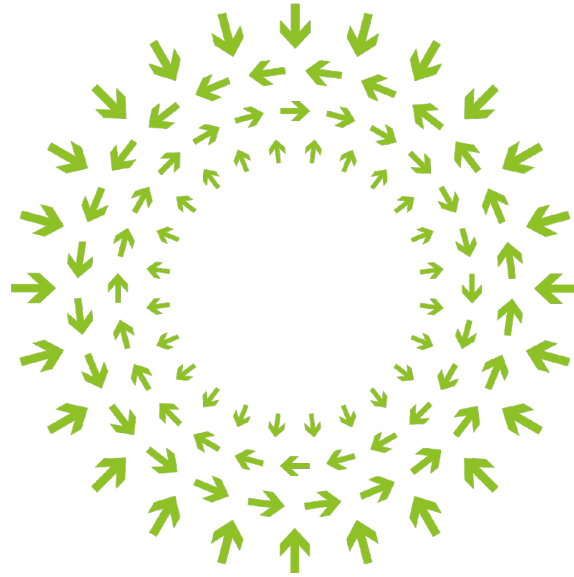




OPEN

Compute Project

openSIL (Open-Source Silicon Initialization Library)

Firmware Architecture Specification

1.0.0

Author: Albert Rennie

Author: Raj Kapoor

Version History

Date	Version #	Author(s)	Description
09 SEPT 25	1.0.0	Albert Rennie, Raj Kapoor	Initial Release

Table of Contents

Version History.....	2
Table of Contents.....	3
License	5
1. Compliance with OCP Tenets	6
1.1. Openness	6
1.2. Efficiency	6
1.3. Impact	6
1.4. Scale	6
1.5. Sustainability	6
2. Terminology	7
3. Introduction	9
4. Scalability and openSIL.....	10
4.1. Definition of openSIL	12
5. Guiding Principles	13
6. High Level Operation	14
6.1. xSIM Time Points vs xPRF Service Calls.....	14
6.2. xSIM Time Points	14
6.3. Memory Allocation	15
6.4. Module Interaction	17
6.5. IP Block Lists	17
6.6. Boot Flow	18
6.7. Input Data	18
6.8. Control Flow	19
6.9. Scalable and Extensible Data	19
7. API Definitions	20
7.1. API - (Host: xSIM).....	20
7.1.1 Library Initialization Services.....	20
7.1.1.1 Function: xSimQueryMemoryRequirements	20
7.1.1.2 Function: xSimAssignMemoryTp1/Tp2/Tp3.....	21

7.1.1.3	Function: SilFindStructure	23
7.1.2	Silicon Specific Support	23
7.1.2.1	Function: InitializeHostSiliconTp*	23
7.2.	API - (Host : xPRF)	24
7.2.1	Feature: Memory Information	24
7.2.1.1	Function: xPrfGetLowUsableDramAddress	25
7.2.2	Feature: PCIe topology	25
7.2.2.1	Function: xPrfGetPcieTopologyStructure	25
7.2.3	Feature: RAS (Reliability, Availability, Serviceability)	26
7.2.3.1	Function: xPrfMcalpIdInstanceIdInit	26
7.3.	API (Internal) - xUSL	26
7.3.1	Read register	27
7.3.2	Write register	27
7.3.3	CPU Functions	28
7.4.	Debug Macros	28
7.4.1	Parameter/pointer validation	29
7.4.2	Algorithm Tracing	29
8.	Appendix A - kconfig	32
8.1.	Overview	32
9.	Appendix B – Host Requirements	33
9.1.	Overview	33
9.2.	Required Elements	33
9.2.1	Required Data	33
9.2.2	Required Services	34
9.2.3	Required Environment Libraries	34
9.3.	Function Specifications	34
9.3.1	Debug Output Service	34
9.3.1.1	Host Function: SilTracePoint	35
9.3.1.2	Message Formatting	36
9.3.1.3	Include file	36

License

Copyright 2025 Advanced Micro Devices, Inc.

openSIL Firmware Architecture Specification by Advanced Micro Devices, Inc. is licensed under CC BY 4.0. To view a copy of this license, visit <https://creativecommons.org/licenses/by/4.0/>

Creative Commons

OCP encourages participants to share their proposals, specifications and designs with the community. This is to promote openness and encourage continuous and open feedback. It is important to remember that by providing feedback for any such documents, whether in written or verbal form, that the contributor or the contributor's organization grants OCP and its members irrevocable right to use this feedback for any purpose without any further obligation.

It is acknowledged that any such documentation and any ancillary materials that are provided to OCP in connection with this document, including without limitation any white papers, articles, photographs, studies, diagrams, contact information (together, "Materials") are made available under the Creative Commons Attribution 4.0 International License found here: <https://creativecommons.org/licenses/by/4.0/>, or any later version, and without limiting the foregoing, OCP may make the Materials available under such terms.

As a contributor to this document, all members represent that they have the authority to grant the rights and licenses herein. They further represent and warrant that the Materials do not and will not violate the copyrights or misappropriate the trade secret rights of any third party, including without limitation rights in intellectual property. The contributor(s) also represent that, to the extent the Materials include materials protected by copyright or trade secret rights that are owned or created by any third-party, they have obtained permission for its use consistent with the foregoing. They will provide OCP evidence of such permission upon OCP's request. This document and any "Materials" are published on the respective project's wiki page and are open to the public in accordance with OCP's Bylaws and IP Policy. This can be found at <http://www.opencompute.org/participate/legal-documents/>. If you have any questions, please contact OCP.

1. Compliance with OCP Tenets

1.1. Openness

Open-sourcing openSIL firmware fosters collaboration and open interface adoption, promoting reuse across silicon vendors and host firmware implementations.

1.2. Efficiency

openSIL enhances efficiency through a common code model that identifies shared elements across different IPs and SoCs, reducing redundancy and development time. Its modular architecture divides code into reusable common components and independently developed program-specific elements, facilitating maintenance and updates. openSIL's host firmware independence and internal API abstraction layers enable broader adoption, easier integration, and streamlined communication between different silicon components.

1.3. Impact

The modularization and common code approach of openSIL are designed to streamline the development process and improve time-to-market for new generation SoCs. This strategy focuses on maximizing code re-use, reducing redundancy, and minimizing incremental validation, which are all crucial to enhancing efficiency and reducing development time.

1.4. Scale

openSIL APIs create extensible deployments for maximum reusability.

1.5. Sustainability

openSIL embraces sustainability by reducing development costs, promoting circular economy and improving the residual value of computing platforms for all business segments.

2. Terminology

- **AGESA™** - AMD Generic Encapsulated Software Architecture, the x86 Silicon Initialization FW limited to UEFI. AMD openSIL is expected to replace AGESA™ in the long term, as of this writing.
- **AGESA-v10** – The successor to AGESA-v9, AGESA-v10 is code that was imported from AGESA-v9. It consists of code that has not yet been rearchitected to AMD openSIL and has not been migrated into TianoCore. AGESA-v10 contains AGESA drivers and code that may not be intended for openSIL (CBS, CPM, etc.). openSIL + AGESA-v10 pair to replace AGESA-v9. AGESA-v10 code will eventually be migrated to either AMD openSIL or AMD EDKII Platform Code (tianocore/edk2*). AGESA-v10 is synonymous with (and sometimes referred to as) *AGCL (AGESA™ Compatibility Layer)*.
- **AMD openSIL** – The openSIL framework integrated into AMD firmware (not to confused with openSIL).
- **API** - Application Programming Interface. A set of function calls provided to the host to accomplish a desired task.
- **APOB** - Communication buffer passed from eSIDs to AGESA™.
- **ASP** - AMD Secure Processor, an embedded µController responsible for the security posture of the AMD silicon. Also referred to as MP0 or MPASP
- **C2R** – Common code to revision specific code APIs that are used to abstract program specific implementations from common code.
- **eSID** - An embedded silicon initialization driver.
- **Host FW** - Firmware that executes on the main platform application processor (In this case, x86 and hence x86 Host FW) including UEFI, coreboot, or Forti BIOS firmware. Host FW ensures that silicon and platform services are enabled prior to passing control to the operating system. In many cases, the host firmware also offers run-time services to the operating system. The Host FW integrates openSIL libraries by linking them statically into its programming environment (or Host Firmware Framework).
- **Host FW Framework** - Programming framework under which the openSIL libraries are located to produce Host FW that is further integrated with other platform silicon firmware to produce the final IFWI image.
- **I2I** – IP to IP APIs that are used to abstract IP blocks from one another.
- **IFWI** - Integrated Firmware Image, a monolithic image of stacked signed/unsigned firmware binaries, hosted by platform boot storage media such as SPI NOR Flash or eMMC NAND Flash devices, necessary to boot platform silicon components.

- **IP Block** – Commonly referred to as sub-modules of openSIL. These contain the silicon initialization for supported hardware sections.
- **openSIL** – Open Silicon Initialization Libraries - an open-source framework of silicon initialization firmware that integrates with x86 host firmware.
- **Platform** - a physical board or system, containing one or more supported processors (a.k.a. 'motherboard' or 'mainboard')
- **SoC** - System on Chip; a supported processor.
- **TP** - Time Point. The xSIM Library allows Host FW to perform SoC initialization in three phases during the boot process. Time Points 1, 2, and 3 are described in subsequent sections of this document.
- **xPRF** - Platform Reference Firmware is one of the three openSIL libraries. It aids in initializing the SoC Si and implements functions that are used by the Host FW to initialize the silicon based on platform configuration.
- **xSIM**: Silicon Initialization Module is one of the three openSIL libraries. It initializes the SoC Silicon. It does not take part in any platform level initialization and is thus not only agnostic to the Host x86 FWs but to the platform layout as well.
- **xUSL** - Utilities Support Layer is one of the openSIL libraries. It offers utility functions required by the two libraries xSIM and xPRF. The xUSL is not exposed to the Host FW.

3. Introduction

openSIL is part of the firmware re-architecture efforts to resolve several identified limitations from both business and technology points of view. Several customers have expressed the need to use a wide variety of host firmware implementations and AMD has had difficulty facilitating that broad support.

openSIL, is a block of x86 Firmware (FW hereafter) that is mainly used to initialize the system CPU and prepare the silicon for booting to a broad spectrum of x86 host firmwares, e.g., proprietary solutions like FortiBIOS, open-source solutions like coreboot, and Independent Firmware Vendor (IFV) UEFI Implementations like Aptio, H2O etc. openSIL is firmware agnostic and does not mandate any one firmware solution. It is one of several firmware re-architecture efforts underway and is focused on the x86 execution elements. Other components in the boot process, such as the ASP, are outside the scope of this specification.

The AMD openSIL implementation splits the AMD Generic Encapsulated Software Architecture (AGESA™) software into two major layers:

- The silicon layer - used for basic silicon operations. This is the openSIL code set described herein.
- The UEFI layer - used to integrate into a UEFI environment. The traditional AGESA™ SW API is retained for compatibility with existing UEFI implementations.

The openSIL architecture will use a loosely coupled design that is scalable and extensible for supporting multiple families of AMD SoC in one data driven set of libraries.

4. Scalability and openSIL

Silicon initialization FW designed to exist within a UEFI FW stack, proves challenging to scale to other nimbler host firmware solutions that foster a more robust security posture due to lower attack surface. The following figure shows the existing stack that caters to the UEFI host firmware requirements.

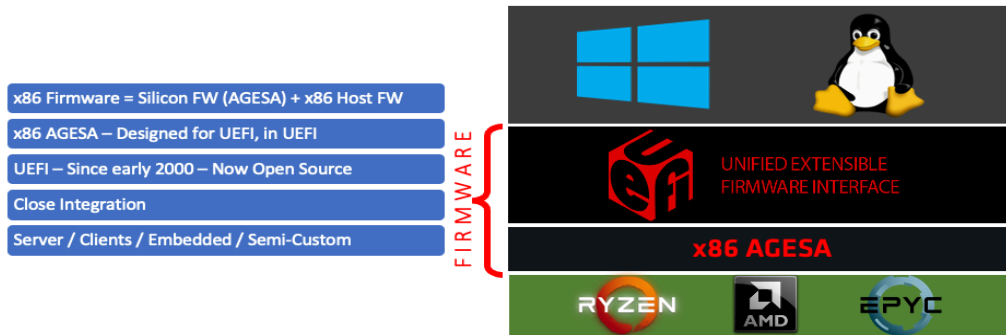


Figure 1: The AGESA™ FW Stack

One way to attain an improved security posture is to open silicon initialization firmware architecture, development, and validation to the open-source communities. AMD is committed to open-source software and is now expanding into various firmware domains with the re-architecture of its x86 AGESA™ firmware stack. AGESA™ was originally designed with UEFI as the host firmware, restricting its scalability for other host bootstrap firmware solutions, including but not limited to coreboot, oreboot, FortiBIOS, and Project µ. A newer, open architecture called openSIL (Open-Source Silicon Initialization Library), that could allow for reduced attack surface and perceivably infinite scalability is now available as a proof-of-concept within the open-source community for evaluation.

openSIL adheres to simple goals of a Host FW agnostic set of libraries written in an industry-standard language that can be statically linked to the host firmware without having to adhere to any host firmware protocols. It is designed to be scalable, simple to integrate, light weight, transparent, and with minimal caller-callee interaction, potentially allowing for an improved security posture.

- ✓ **Agnostic 3 Static Library** solution written in C-17
 - ✓ **Silicon, Platform & Utilities**
- ✓ **Simple & Scalable** integration with any x86 Host FW
- ✓ **Flexible** Platform library scalable to customer and x86 host FW needs
- ✓ **Lightweight & Low chirp** density for increased Security
- ✓ **Open Source** – right from the get-go!

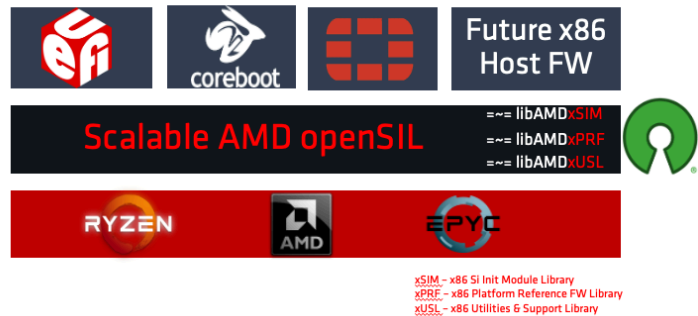


Figure 2: Tenets of openSIL

openSIL is a set of three stateless & static C libraries:

- **xSIM** (x86 Silicon Initialization Module Library)
- **xPRF** (x86 Platform Reference Firmware Library)
- **xUSL** (x86 Utilities & Services Library)

They can be statically linked to any host firmware during compile/link time. Below is a comparison firmware stack diagram that exhibits the scalability between two disparate

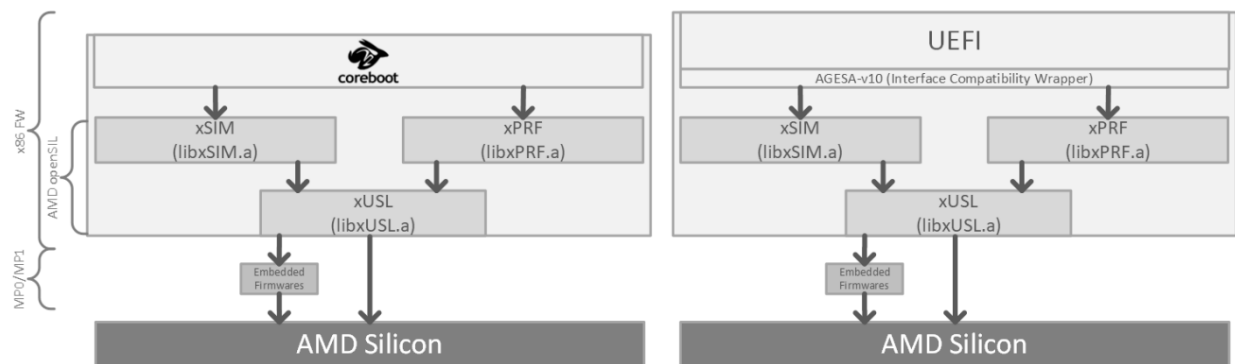


Figure 3: FW Stack diagram of openSIL integrated in coreboot & UEFI Host FW

xSIM (x86 Silicon Initialization Modules) Library provides a minimal set of API services that initialize the platform host silicon. Much of the silicon initialization on AMD-based platforms is performed by embedded μ Controllers prior to x86 reset de-assertion. This includes Memory Interface training & signal conditioning, DRAM JEDEC initialization, and Host Memory Controller Initialization, collectively forming the bulk of Memory Initialization. As a result, system memory is already available by the time the x86 reset is de-asserted. Thus, the services exposed to the Host FW by xSIM library are minimal, yet complete. This library is independent of the platform netlist

and does not perform any silicon initialization tied to platform design (for example Host SoC GPIO programming). Platform designers should not have to make source code modifications to this library.

xPRF (x86 Platform Reference Firmware) Library services are used to initialize platform design-specific silicon initialization as required by the platform firmware. An example of this would be GPIO programming. This may also include SMM handlers to handle events such as RAS etc. and is subject to the board design. Both xSIM and AMD CRB PRF libraries are expected to depart from any industry standard (UEFI, coreboot etc.). They leverage open-source tools (such as GCC) that are expected to be the development tools of choice (although AMD does not preclude one from using proprietary tool set). Users may, if they choose, modify the xPRF to address platform design deviation from the Silicon Vendor's CRB reference design. The library is designed to provide silicon configuration based on platform requirements. However, there are no requirements that limit modifications per system design requirements.

xUSL (x86 Utilities Support Library) Library, as the name suggests offers a set of helper functions to aid the xSIM and the xPRF libraries. This library is not exposed to the host firmware but is instead designed to be exclusively used by the xSIM and xPRF libraries.

Note: A quick note on **AGESA-v10**: Until openSIL is completely adopted into the EDK2 and UEFI firmware packages, the legacy AGESA™ interface will be maintained. This legacy interface is also known as **AGCL**.

4.1. Definition of openSIL

openSIL contains code for host-firmware agnostic code or configs for features that must be initialized at boot-time.

- Included in openSIL:
 - Silicon initialization code.
 - AMD specific firmware features.
 - APIs for silicon specific accesses.
 - Platform-feature configuration for silicon init or silicon features.
 - Boot-time features.
 - Run-time features that need to be initialized at boot-time.
- Not included in openSIL:
 - Any Host FW specific code. (Belongs in the host fw)
 - Industry standard table generation. (Belongs in the host fw.)
 - OS-specific features.
 - Binaries

5. Guiding Principles

- A simple set of static and stateless libraries, written the in C language.
- A minimal set of exposed function calls, allowing for better security posture. At the same time, simplicity should be favored over greatly increased complexity when trying to minimize the exposed functions.
- Isolate silicon initialization from platform design to allow for ease of maintainability and extensibility between ‘thin & light weight’ and ‘feature rich’ silicon initialization firmware catering to the varied needs of the system’s Host Firmware, scaling between the two ends of the spectrum catering to security on one end and feature rich requirements on the other.
- openSIL libraries must specify to the host firmware up-front how much memory is needed for its operation and must be stateless, as such any need for state maintenance must be routed through the Host Firmware, thereby maintaining simplicity in architecture and design.
- All low-level primitives such as access to silicon registers must be implemented within the xUSL to allow for hardware abstraction from the xSIM and xPRF libraries.
- Allow for minimal modification to the existing UEFI ecosystem with enough abstraction to isolate the architectural differences between the native openSIL and legacy AGESA™ interfaces. As AMD openSIL goes through program intercepts, it is expected that this layer will deplete and UEFI Host FW will have fully adopted the native openSIL interfaces.
- Open-Source openSIL, allowing for increased industry vetting, trust from customers, quick functional and security issue resolution turnaround times.
- Must not preclude scaling to Host CPUs/SOCs other than AMD x86 cores allowing mutual customers to seamlessly scale from one silicon vendor to another.
- Changes to data structures must not force a dependency on prior versions of the Host FW
- Provide an interface so that any functionality provided to UEFI & EDK2 by the old AGESA™ can be matched by other host firmware. The host firmware may need to translate information to use that functionality, but it should be supported.

6. High Level Operation

6.1. xSIM Time Points vs xPRF Service Calls

The **xSIM TPs** are required calls from the host firmware to xSIM at a specified time during the boot flow. TP calls provide time-sensitive data to silicon initialization processes, allowing for proper initialization of silicon IPs.

The **xPRF service calls** may be used on demand by the Host (pursuant to any restrictions listed in the function definition). These library functions can be used to support areas like:

- Platform level silicon configuration (e.g., RAS)
- SMM specific init/services
- Multi-Processor configurations

6.2. xSIM Time Points

There are three points in the x86 firmware timeline at which openSIL is designed to be invoked. These are:

- TP1: Early execution. This early time point is responsible for performing basic silicon initialization for fabric, core, and PCIe. openSIL does not place any timing requirement on this time point. The host firmware must ensure the environment is capable of supporting openSIL (memory). Actions include but are not limited to:
 - Apply the PCIe platform configuration to the SOC's PCIe topology and train the PCIe links.
 - Wake up and configure Application Processors (APs).
 - Configuration of peripheral IO devices. For example:
 - USB host controller(s)
 - SATA controller(s)
 - SD/MMC controller(s)
 - GPU frame buffer
- TP2: Post-PCI enumeration. At this timepoint, the host firmware is expected to have finished allocating resources to all PCI devices. Any silicon initialization that requires PCI resource allocation (e.g., IO/MMIO) to be completed should be handled at this time point.
 - Enabling PCIe Hotplug
 - Enablement of some RAS event triggers
- TP3: Pre-OS. Prior to OS handoff, this timepoint provides a final opportunity for silicon initialization/configuration. Functions that might be performed at this TP are:

- Lock registers (NBIO, SMM)
- Prep all IPs for boot to OS

6.3. Memory Allocation

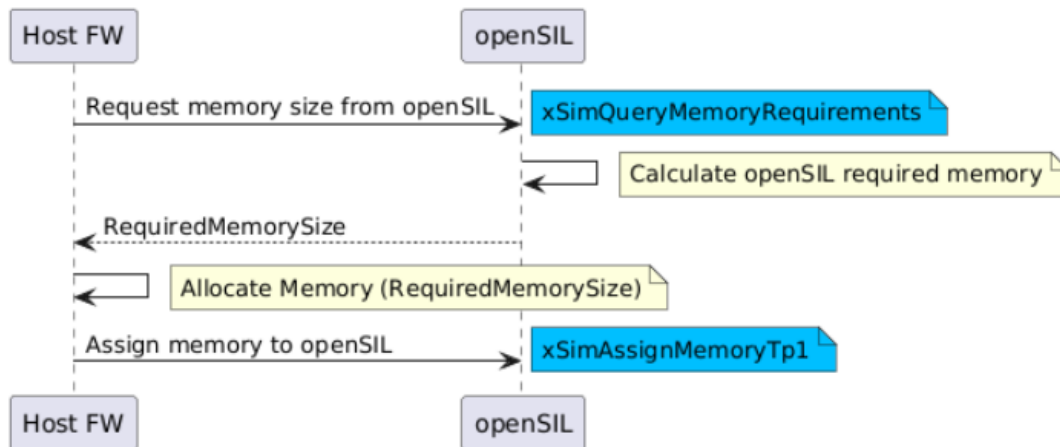


Figure 4: xSIM Memory Allocation Sequence Diagram

Prior to openSIL silicon initialization, the host firmware must query openSIL for the memory required to perform the initialization (Figure 4). Based on the requirements, host firmware must allocate a single block of contiguous memory. The host must pass the base address of the memory block to each time point entry point (refer to *xSIM Time Points vs xPRF Service Calls*).

The size of the block is determined by a call to the xSIM API `xSimQueryMemoryRequirements`. This API will provide the minimum memory requirements for openSIL to execute. It is the responsibility of the host firmware to ensure sufficient memory is allocated. If the allocated memory is insufficient, openSIL will return an out-of-resources error during initialization.

Prior to each timepoint execution, the host firmware must provide the base address of the allocated memory to openSIL via the xSIM API `xSimAssignMemoryTp*` (* = 1, 2, 3).

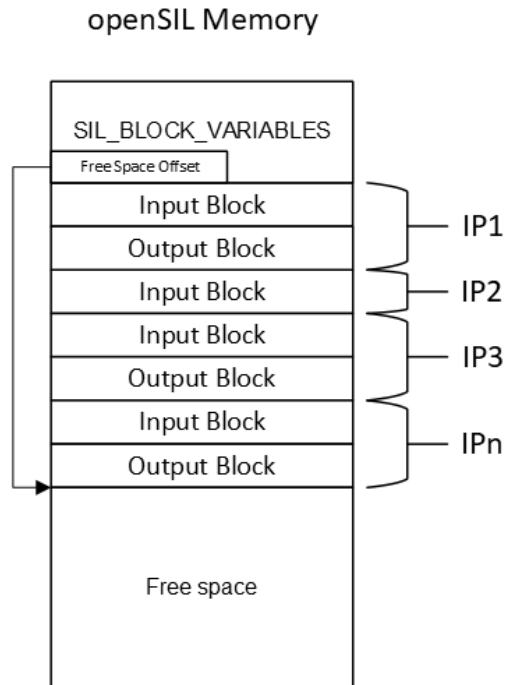


Figure 5: openSIL Memory Contents Example for xSimAssignMemoryTp1

Once the host firmware has provided the base address of the allocated memory to openSIL, each silicon IP can/will reserve chunks of this memory to support its IP data block needs (input, output, or other data) (Figure 5). IP input blocks, specifically, are populated with openSIL defined defaults.

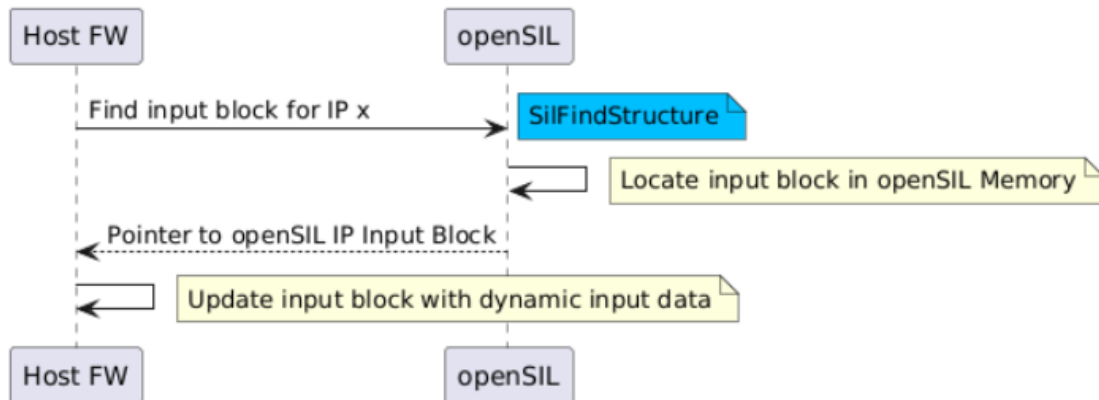


Figure 6: Host Firmware Dynamic Update of Input Block Data

Host firmware has the option to update the default input block values to host defined values (Figure 6). To do this, pointers to IP data blocks can be retrieved using the xSIM API `SilFindStructure`.

Other than return status, the IP input/output blocks are the primary mechanism used to exchange data between host firmware and openSIL.

Notes: 1. [at present] there is no plan for Operating System (OS) run-time xSIM usage or activity. So, there is the potential to reclaim memory that has been allocated to xSIM.

Retention time for the allocated block is decided by the Host. After the initial xSIM call, the block will contain results and configuration details which must be retained until all openSIL calls are completed. After that point, the Host can choose to retain the block or transcribe the details for which it is interested to another storage location (e.g. SMRAM for retention across S3).

6.4. Module Interaction

The xSIM and xPRF APIs are exposed to, and may be called by, the Host Firmware. The xUSL should only be used by the xSIM & xPRF libraries.

xSIM APIs are declared and documented in the host exposed header file xSIM-api.h. Similarly, all xPRF APIs are declared and documented in the host exposed header file xPRF-api.h.

6.5. IP Block Lists

openSIL silicon IP initialization follows table driven execution as defined by a set of IP block lists that are unique for each program/SoC supported in openSIL. An IP block list is an initialized instance of the SOC_IP_TABLE structure.

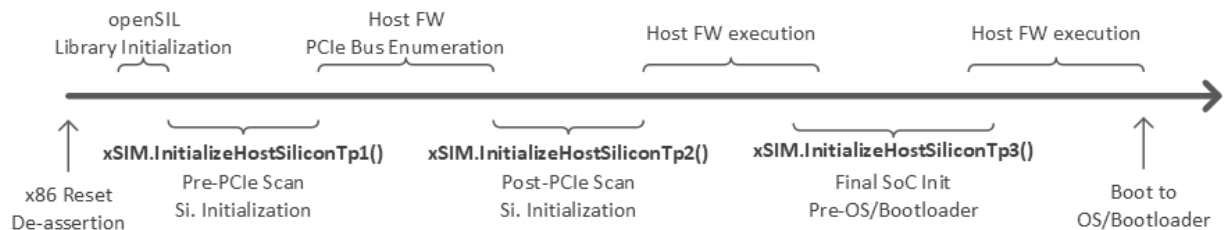
Each IP block list is a collection IP_RECORD structures:

```
typedef struct {
    SIL_DATA_BLOCK_ID IpID
    uint32_t          BlkRequestSize;
    FCN_SET_INPUT     SetInput;
    FCN_IP_INIT       Initialize;
    SIL_API_INIT      ApiInit;
} IP_RECORD;
```

IpID	A unique identifier for the IP record.
BlkRequestSize	The memory requirements in bytes for this IpID
SetInput	An initialize function pointer responsible for reserving and initializing the IP data block(s).
Initialize	The entry point for the IP silicon initialization.
Apilnit	A pointer to a function responsible for initializing this IPs APIs (C2R, I2I)

Each SoC will define a separate block list for each TP (e.g. SocIpTblF**M**Tp1, SocIpTblF**M**Tp2, SocIpTblF**M**Tp3). The naming convention includes the SoC family number, model number, and TP to uniquely identify the table.

6.6. Boot Flow



**Figure 7: x86 Firmware Example Execution Flow.
(Top: Host Firmware Execution, Bottom: openSIL Execution)**

Initialization of the silicon may need to occur in separated time phases, requiring actions by the Host between the phases. This is dependent upon each silicon IP block and the state it requires for execution. To manage these variations the xSIM library defines three time points (TP(s)) during the firmware execution sequence where the xSIM is guaranteed to be given control. Any silicon IP initialization that requires execution at these time points will be launched. Figure 7 includes example host firmware execution sequences. It should be noted that these sequences are optional for host firmware.

- openSIL Library Initialization

Host firmware is responsible for setting up the openSIL execution environment for each library instance.

- PCIe Bus Enumeration

Host firmware is responsible for enumeration of the PCI bus prior to launching openSIL TP2. The timing of this sequence is Host FW dependent but must occur between TP1 and TP2.

- Host FW Execution

Optional sequences that represent phases of host firmware execution. These are host firmware defined and not quantified by this FAS.

6.7. Input Data

Prior to each TP entry, the host firmware is responsible for providing the base address of the openSIL data block.

At each TP, the associated xSIM entry point is called to set the input data for that timepoint. All host firmware configurations/options are passed to the openSIL libraries via a collection of IP defined input blocks. openSIL IPs will initialize their input block data with a set of predefined defaults. The host firmware can update the input block data with dynamic runtime values to override build time defaults (refer to section “[Memory Allocation](#)”).

6.8. Control Flow

The xSIM or xPRF functions will not have ‘private memory’. openSIL cannot dynamically allocate memory resources from the system. It can only reserve chunks of the Host FW provided memory (described in section “[Memory Allocation](#)”) for IP data blocks (input, output, or other). The libraries only have a stack when they are called from the host firmware. openSIL libraries are not drivers. Thus, they cannot initiate control flows. All flows must be initiated by a call from the Host FW.

6.9. Scalable and Extensible Data

The stability of an architecture rests on the foundational elements such as data structures. This means that over time, changes to data structures must not force a dependency on the prior versions of Host FW, requiring it to be modified and re-compiled with the new sets of data structures. When a new feature is introduced, it must be optional to previous revisions of the Host FW, maintaining backward compatibility. To this end, there are some rules about modifying data structures:

1. All data structures must have a version number field. This field must be updated for any changes affecting the openSIL APIs used by the Host FW. This number may have major and minor components with only a change in the major indicating an API upgrade being required at the Host.
2. For minor upgrades, any new structure elements must be added to the end of the structure. The updated structure must be backward compatible with previous implementations. To support this type of change, all openSIL FW and Host FW should declare enums to represent element values. These enums should be used in comparison/conditionals rather than numerical constants (e.g. if (“option” > 0){}).
3. New input block data should be configured with default values that will allow the Host FW flow to remain unchanged.

By extending the backward compatibility of data structures, openSIL can ensure new features and data have no impact on legacy programs, eliminating the need for costly validation cycles, and allowing teams to focus precious resources on critical components.

7. API Definitions

The below definitions are design-level and intended to convey the design concept. As re-architecture efforts continue, these definitions may diverge from implementation. Implementations will conform to the concepts and principles presented in this specification.

Unless otherwise noted in the API documentation, all APIs require initialization of the openSIL environment described in section “[Memory Allocation](#)”.

7.1. API - (Host: xSIM)

The xSIM module serves the following purposes, which provide the basic initialization of silicon.

1. Provide the framework required to prepare the execution environment for openSIL and execute silicon initialization. This includes, but is not limited to, creating and finding data blocks, defining SoC IP block list instances and definitions, and dynamic selection of IP block list(s).
2. Provide xSIM API services that are exposed to host firmware to perform non-platform specific silicon initialization.
3. Control the execution flow of silicon IP initialization through the SoC IP Block list ([IP Block Lists](#))

7.1.1 Library Initialization Services

xSIM provides framework services that are used to initialize the environment required for openSIL IPs to perform the x86 silicon initialization. This framework represents both internal services and externally visible API services. The framework provides a mechanism for host firmware to provide necessary resources (e.g. memory) and for openSIL IP modules to consume those resources (e.g. creating IP data blocks).

7.1.1.1 Function: xSimQueryMemoryRequirements

This API is used by host firmware to query the minimum memory requirements for openSIL to initialize the SoC silicon.

Prototype:

```
size_t xSimQueryMemoryRequirements ();
```

Description:

This xSIM API will collect information from the active SoC IP initialization modules to calculate the aggregate size of memory required for openSIL execution to complete. The aggregated total will be returned to the host firmware.

Returns and Outputs:

The return value is the minimum size in bytes required by all openSIL IP components to successfully initialize their respective silicon.

7.1.1.2 Function: xSimAssignMemoryTp1/Tp2/Tp3

After the Host FW has allocated the requested amount of memory, it uses this function to inform xSIM about the location and size of the allocated block. There are unique memory assignment functions for each time point that reference unique SoC IP block lists for that time point. This provides the flexibility for host firmware to link executable code necessary only for the TimePoint in question.

Prototypes:

```
SIL_STATUS xSimAssignMemoryTp1 (SilContext, MemorySize);  
SIL_STATUS xSimAssignMemoryTp2 (SilContext, MemorySize);  
SIL_STATUS xSimAssignMemoryTp3 (SilContext, MemorySize);
```

Arguments:

`SilContext` - A context structure through which host firmware defined data can be passed to openSIL. The Host FW is responsible for initializing the `SIL_CONTEXT` structure.

`MemorySize` - The size, in bytes, that the Host FW has allocated for use by openSIL.

Description:

This API will:

1. Retrieve the IP block list of the active SoC.
2. Initialize IP APIs (I2I, C2R) according to the requirements in the IP block list.
3. Initialize IP data block (input/output) according to the requirements in the IP block list. If insufficient space is available, this API will return error `SilOutOfResources`.
4. Copy IP block defaults to the newly assigned IP block(s).

Note: IP input/output blocks are currently only assigned during TP1. These blocks should be maintained by the Host FW and provided during subsequent calls into openSIL.

Moving the Memory Block - Some Host FW environments may have the need to move the memory block allocated for openSIL use. The Host FW is responsible for providing the base address via the `SilContext` structure.

Returns and Outputs:

Completion status -

1. `SIL_STATUS`:
 - `SilPass` - function completed successfully.
 - `SilOutOfResources` - the block provided was insufficient for the quantity requested by the sub-modules.

<memory block> - filled with IP sub-modules structures.

Related Definitions

```
typedef struct {
    uint32_t      HostBlockSize;
    uint32_t      FreeSpaceOffset;
    uint32_t      FreeSpaceLeft;
    ACTIVE_SOC_DATA ActiveSoC;
    PLATFORM_DESC PlatformData;
    uint64_t      Ip2IpApi[SilId_ListEnd];
    uint64_t      Common2RevXferTable[SilId_ListEnd];
} SIL_BLOCK_VARIABLES;
```

HostBlockSize	Size of block allocated by the Host
FreeSpaceOffset	Offset to next area of free space
FreeSpaceLeft	Value tracking remaining free space
ActiveSoC	Descriptors for SoC in the socket
PlatformData	Descriptors for the platform
Ip2IpApi	IP to IP API table. This table provides abstraction between IPs.
Common2RevXferTable	Common to Rev specific transfer (Xfer) table. This table provides abstraction IP specific code from common code.

7.1.1.3 Function: SilFindStructure

Once the allocated memory block has been assigned to the IP blocks and filled with structure data, the Host FW will use this function to locate a structure within the memory block based on the IpID and an instance number.

Prototype:

```
void* SilFindStructure (StructureID, InstanceNum);
```

Arguments:

StructureID – A value from SIL_DATA_BLOCK_ID enum defined in xSIM-api.h which identifies the specific IP block to be targeted.

InstanceNum - Always starting with 0, this value indicates which one of many possible instances of an IP block is to be located.

Description:

The Host FW can use this function to retrieve openSIL IP data block(s). Any external data block InstanceNum should be defined in the IP *-api.h.

Returns and Outputs:

void* - This is the address of the block requested. A returned value of NULL indicates the block was not found.

7.1.2 Silicon Specific Support

The following routines provide support code for the specific silicon present in the system. This code may contain support for multiple versions of the SoC. The proper support routine to use is determined by the ID of the SoC present in the socket. There are discrete IP initialize functions for each time point that utilize discrete SoC IP blocks for that time point. This allows host firmware to link executable code only necessary for that timepoint.

7.1.2.1 Function: InitializeHostSiliconTp*

This is the main API implemented by the openSIL FW in the xSIM library to initialize SoC silicon. It is called by the Host FW after `xSimQueryMemoryRequirements` (currently only before TP1) and `xSimAssignMemoryTp*`. This call starts the IP block initializations for the specified time point.

Prototypes:

```
SIL_STATUS InitializeHostSiliconTp1 ();  
SIL_STATUS InitializeHostSiliconTp2 ();  
SIL_STATUS InitializeHostSiliconTp3 ();
```

Description:

This function proceeds through each IP block of the SoC to initialize its related silicon.

Returns and Outputs:

`SIL_STATUS` - indicates the general outcome of the process:

1. `SilPass` - the process completed successfully.
2. `SilDeviceError` - one or more of the sub-modules reported an error in their initialization process. Inspect the IP block output blocks for information.

<IP output block> - each output block will provide extended result information.

7.2. API - (Host : xPRF)

While the xSIM library focuses on the core silicon initialization, the xPRF focuses more on the platform centric features. xPRF services are subject to their usage requirements as defined in the API. These services serve as a means for openSIL to provide on demand information or platform related configuration to Host FW. Take for example, some RAS policy requesting to enable specific error signaling mechanisms. And xPRF service may provide access to the silicon to initialize said mechanisms. The services may also be used to provide silicon information necessary for host firmware to build industry standard tables such as ACPI and SMBIOS.

SMM and Multi-Processor environments are not natively supported in openSIL. Those environments are owned and controlled by Host FW. The xPRF services, however, provide a means for Host FW to perform some silicon configuration in SMM or Multi-processor space on demand.

7.2.1 Feature: Memory Information

This is a special case function that may be used by Host FW prior to openSIL environment setup.

Some host firmware may require memory information provided by openSIL to initialize the system memory map. Therefore, this service is implemented in a way such that Host FW will not be required to provide full openSIL memory requirements.

7.2.1.1 Function: xPrfGetLowUsableDramAddress

Prototype:

```
uint32_t xPrfGetLowUsableDramAddress (SIL_CONTEXT *SilContext);
```

Arguments:

`SilContext` - Input buffer for the Sil Context structure. The Host is responsible for initializing the data in the SIL_CONTEXT structure.

Description:

Get top of low (<4GB) usable DRAM accounting for pre-assigned buffers for system use. This may be used by the host firmware to locate low usable DRAM top. The ApobBaseAddress is optional if set in Kconfig properly.

Returns and Outputs:

Top address of available DRAM region below 4GB. 0 if failure.

Related Definitions:

SIL_CONTEXT

7.2.2 Feature: PCIe topology

7.2.2.1 Function: xPrfGetPcieTopologyStructure

Prototype:

```
void * xPrfGetPcieTopologyStructure (
    SIL_CONTEXT *SilContext,
    uint32_t *InfoBlockDataSize
);
```

Arguments:

`SilContext` - Input buffer for the Sil Context structure. The Host is responsible for initializing the data in the SIL_CONTEXT structure.

`InfoBlockDataSize` - Output block size

Description:

Returns the base address of PCIe Topology Structure and its size.

Returns and Outputs:

A pointer to the openSIL NBIO topology structure.

Related Definitions:

SIL_CONTEXT

NORTH_BRIDGE_PCIE_SIB

7.2.3 Feature: RAS (Reliability, Availability, Serviceability)

7.2.3.1 Function: xPrfMcalpIdInstanceIdInit

Prototype:

```
SIL_STATUS xPrfMcaIpIdInstanceIdInit ( SIL_CPU_INFO *RasCpuInfo );
```

Arguments:

RasCpuInfo - The CPU info structure for the core to program (RasClass-api.h).

Description:

Program the Core MCA_IPID MSR Instance ID values for the CPU specified in RasCpuInfo.

Returns and Outputs:

SilPass - If the function completed normally

SilNotFound - If RAS I2I was not found in the I2I list

Related Definitions:

SIL_CPU_INFO

7.3. API (Internal) - xUSL

These are the low-level access functions available to the xSIM and xPRF libraries. These functions may be instrumented for various needs. It is expected that all IP modules will use these functions for their related purpose. The IP module may implement ‘helper’ functions to provide base address or other context useful for the IP module; however, those helper functions must ultimately call these functions to garner the benefits of the instrumentation.

These functions are **NOT** intended to be available to the Host firmware. The Host firmware should focus on features and not the register level implementations.

7.3.1 Read register

These functions are provided to perform a read access to a silicon IO register. There are different access styles.

Prototypes:

```
uint8_t xUSLIoRead8 ( uint16_t Port);
uint16_t xUSLIoRead16 ( uint16_t Port);
uint32_t xUSLIoRead32 ( uint16_t Port);
```

Arguments:

Port - IO Port identifier for the register to be read.

Description:

These functions will attempt to read the target register and place the value read into the return value.

Returns and Outputs:

(response) - This is the value read from the register.

7.3.2 Write register

Prototypes:

```
void xUSLIoWrite8 (uint16_t Port, uint8_t Value);
void xUSLIoWrite16 (uint16_t Port, uint16_t Value);
void xUSLIoWrite32 (uint16_t Port, uint32_t Value);
```

Arguments:

Value - pointer to the location where the value to be written to the IO register is located.

Port - IO Port identifier for the register to be read.

Description:

These functions will attempt to write the target register with the Value in param list.

Returns and Outputs:

None.

7.3.3 CPU Functions

These functions perform CPU specific operations. Functions in this category are architectural. This means they are described in the *AMD64 Architecture Programmer's Manual*, volumes 1-5. These elements do not change from CPU family to family and are not considered 'silicon specific'. Examples are provided below.

Prototype:

```
bool SocCheck (uint32_t SocFamilyID, uint32_t Attributes);
```

Arguments:

SocFamilyID – SoC base and extended family to compare against active SoC.

Attributes – Base and extended model, package type, and stepping to compare against active SoC.

Description:

This function checks SOC Family & Ext Model & Base Model & Stepping & Package Type identification ID against the active SoC.

Returns and Outputs:

bool – The check status:

- true - The check matched the active SoC.
- false - The check did not match the active SoC.

<Attributes> - The attribute contains optional information about the base and extended model, stepping, and package type that is being checked.

- To check for the extended Model, set Attribute to FIXED_EXT_MODEL
- To not check for the extended Model, set Attribute to ANY_EXT_MODEL
- To check for the base Model, set Attribute to FIXED_BASE_MODEL
- To not check for the base Model, set Attribute to ANY_BASE_MODEL
- To not check for the stepping, set Attribute to ANY_STEPPING
- To not check for the package type, set Attribute to ANY_PKG_TYPE
- To check the package type, set package type to Attribute bit 0~3

7.4. Debug Macros

These macros are defined in the SilCommon.h include file and used throughout the openSIL libraries. Using macros to enable the debug system allows this code to be dropped from production systems to save space and improve performance.

The following Macros are used in the openSIL code and are portals to the implementation by the Host. Any functions supporting the actual output of the debug data to a destination device are supplied by the Host.

7.4.1 Parameter/pointer validation

Macro:

```
assert (condition);
```

This macro is only available when SIL_DEBUG is enabled. The primary use is for validating pointers to check for null - a common development phase error. This macro should be used with care as it is not an acceptable for non-debug error handling.

Arguments:

`condition` - a logical statement. Evaluation of this statement must be true to pass and continue execution. A false value will cause execution to halt - for debug of the reason/cause.

Processing:

This macro invokes a similar macro/function in the Host environment, which is responsible for the condition evaluation and subsequent halt of execution.

7.4.2 Algorithm Tracing

Macros:

```
#define XSIM_TRACEPOINT(MsgLevel, Message, ...) \
do { \
    if (SIL_DEBUG_ENABLE) { \
        ((HOST_DEBUG_SERVICE)SilTracePoint) (MsgLevel, "SIL:xSIM:", \
        Message, __FUNCTION__, __LINE__, ##__VA_ARGS__); \
    } \
} while (false)
#define XUSL_TRACEPOINT(MsgLevel, Message, ...) \
do { \
    if (SIL_DEBUG_ENABLE) { \
        ((HOST_DEBUG_SERVICE)SilTracePoint) (MsgLevel, "SIL:xUSL:", \
        Message, __FUNCTION__, __LINE__, ##__VA_ARGS__); \
    } \
} while (false)
#define XPRF_TRACEPOINT(MsgLevel, Message, ...) \
do { \
    if (SIL_DEBUG_ENABLE) { \
        ((HOST_DEBUG_SERVICE)SilTracePoint) (MsgLevel, "SIL:xPRF:", \
        Message, __FUNCTION__, __LINE__, ##__VA_ARGS__); \
    } \
} while (false)
```

```

    } \
} while (false)

```

Arguments:

MsgLevel - Value from the defined enum that sets the message level of importance. This parameter indicates to which level this message belongs. The Host may filter messages by using level filtering; If the message level meets the threshold of the Host filter value, then the Host will send this message to the log output.

1. **SIL_TRACE_ERROR** - Unexpected/unwanted (problematic) conditions
2. **SIL_TRACE_WARNING** - Result is unexpected but not necessarily a problem
3. **SIL_TRACE_ENTRY** - Messages showing entry flow
4. **SIL_TRACE_EXIT** - Messages showing exit flow
5. **SIL_TRACE_INFO** - Discovery/Configuration information that is useful for debugging
6. **SIL_TRACE_VERBOSE** - Verbose information that may be suppressed in most cases
7. **SIL_TRACE_RAW** - Used to print RAW Message without the openSIL formatting.

Message - String containing the message to log.

... - VarArgs list of values to be processed into the output string.

Description:

Compiler pre-processor will pull the function name and line number to pass to the output. This is to tag the output to indicate the origin.

Processing:

This macro will call the `SilTracePoint` (see below) which will format the output string with SIL arguments extracted from the compiler (function name and line number) in addition to the macro arguments. The format of the SIL message output is as follows:

```
"SIL:"<SILMODULE>": "<FUNCTION>": "<LINENUM>": "<MsgLevel>": "<string
Msg>"\n"
```

Example: "SIL:xSim:InitializeHostSiliconTp1():268:Entry Execute
xSIM IP modules Init"

Macro Implementations:

```

do { \
    if (SIL_DEBUG_ENABLE) { \
        ((HOST_DEBUG_SERVICE)SilTracePoint) (MsgLevel, "SIL:xSIM:", \
        Message, __FUNCTION__, __LINE__, ##__VA_ARGS__); \
    }
}

```

```
    } \
} while (false)
```

NOTE: The **do-while** loop is used to prevent a **double** end statement when using the `TRACEPOINT` macros in c code. The macro calls are suffixed with a semicolon to make it similar to other lines of c code. Double statement endings can have unintended consequences. For example, a **if-else** statement without `{ }` encapsulating the code.

8. Appendix A - kconfig

In addition to support Host FW customization of the IP input blocks, openSIL provides kconfig configuration support to customize default arguments based on platform requirements without requiring Host FW interaction.

8.1. Overview

openSIL will support a selection-based configuration tool (kconfig) that will produce includable files that can be used to customize the platform selections.

9. Appendix B – Host Requirements

9.1. Overview

While openSIL is SoC centric, platform firmware also plays a major role in the initialization of silicon. openSIL IP input blocks contain configuration arguments that are the main mechanism for Host FW to provide platform level configuration input to openSIL. Each sub-module provides default configuration data for its input blocks. Host FW has the opportunity, if required, to update those defaults to some customized configuration.

9.2. Required Elements

openSIL has a set of requirements that must be satisfied by host firmware or platform firmware developers. These requirements are broken into two categories.

9.2.1 Required Data

openSIL input blocks are the mechanisms for Host FW to customize the silicon initialization. This customization is supported through the IP input block data elements.

- ***_SilCfg.h** - This is the output file from the Kconfig Tool that reflects the choices made by the user for how this platform should be configured. The Kconfig tool should be run at least once for the platform to establish the platform config file *_SilCfg.h

The Host FW is not required to generate these configuration headers. openSIL includes a default set of configurations for all support programs/platforms. The header(s) should be re-generated by the FW designer using the kconfig configuration tool if customizations are required. The configuration header(s) can be used within openSIL to provide customized build time defaults for IP input block. Additionally, some configurations may be available to specify other Host FW platform requirements that may be outside the scope of IP initialization (for example, specification of the APOB base address)

- **IPxxxDefaults.c(.o)** - Each IP block will define an input block and related option selections for the Kconfig tool. The Kconfig tool's output file will be used to set the values of the input block's 'defaults' declaration of the IP's input block. The original .C file(s) are located in the xSIM/IPxxx directories, and the host specific object file(s) will be located in the platform's build output directory.

9.2.2 Required Services

The silicon initialization algorithms define some restrictive timing requirements for certain operations. These requirements are satisfied by using object linked routines provided by the Host FW which will be called from the openSIL silicon initialization algorithm at the appropriate time. As such, these services must be present and linked with the openSIL libraries.

1. **Debug Output Service** - The SIL modules will be instrumented to facilitate problem debugging and quick resolution. Debug support routines will be used to provide Trace Output via the Host services. This function will present a message string to the Host selected output destination.

9.2.3 Required Environment Libraries

The C language compilers use intrinsic functions and definitions when compiling the C source code. The Host environment (e.g. UEFI or standard C libraries) must provide the following elements for the openSIL source files to compile:

1. `stdbool.h`
2. `stdint.h`
3. `stddef.h`
4. `assert.h` - This includes a translation for the openSIL `assert` macro to the environment's `assert`, for example:

```
#define assert(Statement) ASSERT(Statement)
```
5. `MemOpsX32.nasm` and `MemOpsX64.nasm` - provides the `_memset()` and `_memcpy()` functions in 32b and 64b x86 modes.
6. `MathU64.nasm` - intrinsic functions for handling 64b math when in 32b x86 mode.

9.3. Function Specifications

This section provides details for implementing the Host FW required elements.

9.3.1 Debug Output Service

This service is to be provided by the Host to facilitate debugging. The SIL modules will be instrumented with macros that will send debug string messages through this Host service. If the SIL debug macros are enabled, then this service is required to be present; see the `SIL_DEBUG_ENABLE` description below.

9.3.1.1 Host Function: SilTracePoint

This function is implemented by the Host environment and is called by the openSIL libraries' debug instrumentation when generating debug output information. The Host FW is responsible for determining the appropriate output destination based on its environmental configuration options.

Prototype

```
void SilTracePoint (
    size_t      SilMsgLevel,
    const char  *SilPrefix,
    const char  *Message,
    const char  *Function,
    size_t      Line,
    ...
);
```

Arguments:

SilMsgLevel - This indicates the level of detail being provided by this message. The Host environment may use this value to filter the amount of detail that is sent to the output log. The value comes from the SIL enum defined for level filtering.

SilPrefix - String containing the SIL prefix message with placeholders for function name, line number, and message level to print.

Message - String containing the message to log. See notes on the format below. This string may contain VarArg style references similar to the standard C `printf()` functions.

Function - String containing the name of the calling function.

Line - The line number in the calling file.

... - This is the list of VarArgs to be applied into the message string. The Host environment is responsible for the processing of the VarArgs.

Description

This function will process the Message, SilPrefix, and VarArgs arguments then output the resulting string to the debug destination. The Host is responsible for any formatting required by the destination device. This function will be called by the SIL layer via the `XSIM_TRACEPOINT`, `XUSL_TRACEPOINT`, and `XPRF_TRACEPOINT` macros whenever the Host has enabled the SIL debug service by setting the master enable switch.

Returns and outputs:

None

9.3.1.2 Message Formatting

The SIL debug macros will accept or create several arguments that identify the source of the message. The string will be formatted to include these identifiers in a form suitable for filtering by a script or editor.

The list of SIL arguments is as follows:

name	purpose/definition
SilPrefix	This is a string "SIL: [xSIM xPRF xUSL]:". This signature is used to differentiate openSIL message from other Host FW messages.
Function	This value is extracted from the compiler and automatically included into the output string. It contains the C function name from which the call is performed.
Line	This value is extracted from the compiler and automatically included into the output string. It contains the file line number (file relative, not function relative) where the call is performed.
MsgLevel	This indicates a type of message being sent. This parameter will be passed to the Host routine for potential filtering. The below values are defined in an order of most important (low value) to least important (higher value) to facilitate level filtering. Defined values include: 1. Error - an unrecoverable error that is the cause of an unstable system or asserts. 2. Warning - error condition leading to a return code other than SilPass. 3. Entry - This is the entry point of the specified function. 4. Exit - This is the end/exit point of the specified function. 5. Info - information purposes, such as specific code path (if () {} ...), announcing normal progress/flow.
Message	A formatted string null terminate string.
Variable	A collection of arguments that can be inserted into the formatted Message Args

9.3.1.3 Include file

An include file (SilCommon.h) is provided to define the debug macros and the master build time enable switch.

```
#ifndef SIL_DEBUG_ENABLE
#define SIL_DEBUG_ENABLE true
#endif
#if !SIL_DEBUG_ENABLE
#define NDEBUG
#endif
```

SIL_DEBUG_ENABLE

1. true - When set, the SIL debug macros will be expanded in the compile and will output messages for debugging.
2. false - When unset, the SIL debug macros will be excluded from the compile and NO output messages will be present.

The Host FW should produce SIL_DEBUG_ENABLE based on the value of their IDE/build environment 'build type' (typically 'Release' or 'Debug').