Design Approaches and Architectures of RISC-V SoCs

Author: P R Sivakumar, Founder and CEO, Maven Silicon

We design different kinds of System-on-Chips (SoCs/Chips) tailored for different electronic products. Let's explore how we approach designing various electronic products like embedded microcontrollers, smartphones, Linux servers, and cloud servers.

Usually, we prefer the 32-bit embedded compressed version of RISC-V Base ISA, called RV32EC, for embedded microcontrollers that we use in coffee machines, fitness bands, and other similar applications. We might prefer bare-metal or RTOS (Real-Time Operating System) for such simple microcontrollers. Some of them could be real-time systems like automotive chips that demand real-time processing without any delay. Obviously, we prefer bare-metal coding (physical addressing) for such real-time software applications instead of using an operating system (OS – virtual addressing) like Linux. A bare-metal software application directly runs on the processor following physical addressing with RAM. As the OS follows virtual addressing with the RAM for virtual memory management, it's time-consuming. But we prefer OS for executing and managing multiple applications (hundreds or thousands or millions of threads) in parallel - for products like desktops and servers.

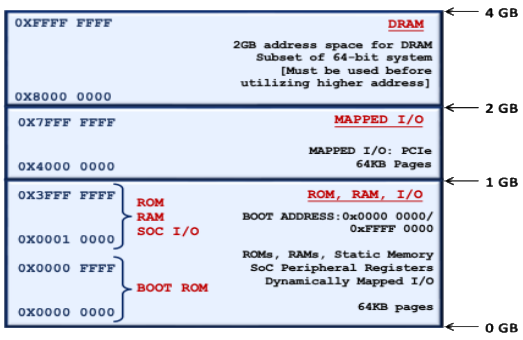
Similarly, we prefer a high-performance multicore RISC-V processor like RISC-V RV64G for a smartphone that demands a complex 64-bit SoC - runs a 64-bit OS like iOS/Android. RV64G is based on the 64-bit Base ISA, called RV64I, which supports 64-bit OSes for smartphones and desktops. G means General Purpose. It includes all the standard extensions: IMFD&A. You can refer to my **RISC-V ISA Overview YouTube Video** to explore the RISC-V ISA, unprivileged and privileged architectures.

The complexity of the chip and software varies based on the complexity of the product. RISC-V ISA offers various base ISAs - RV32I, RV32E, RV64I, and RV128I to support 32/64/128-bit Oses, and extensions IMFDA, etc., to design different kinds of SoCs for various products like embedded microcontrollers, smartphones, Linux servers, and cloud servers.

Let me explain how we approach creating bare-metal and OS-based applications with two good examples – RISC-V 32-bit SoC and RISC-V 64-bit SoC.

RISC-V 32-bit SoC – Embedded Microcontroller

As shown in the figure, we follow a 32-bit memory map to design 32-bit SoCs.



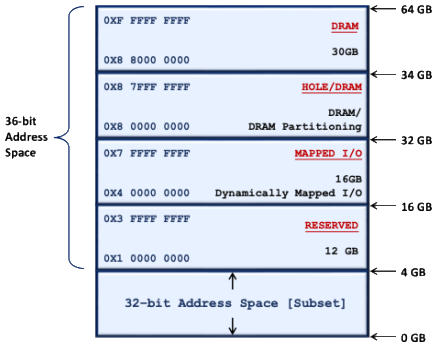
At present, RISC-V does not define or offer any standard memory map framework to design SoCs, but you can expect it from RISC-V International in the future. As there is no standard framework, different vendors follow different memory maps for their chips.

Let's understand the 32-bit memory map shown above. We design a chip with 4GB address space for the bare-metal coding. Within 4GB, we map everything: ROM, I/Os, SRAM, DRAM, external memories, etc., as shown in the figure. In this case, you can have only limited DRAM (2 to 3 GB), as we need a minimum of 1 GB for other components like ROM and I/O interfaces. The bare-metal software application will follow physical addressing with RAM. Usually, we follow this framework and approach for designing 32-bit embedded microcontrollers.
RISC-V ISA also offers Sv32 address translation to realize 32-bit systems with a 32-bit OS. The Sv32 address translation scheme generates a 32-bit virtual address with 4GB of virtual address space and a 34 bits physical address with 16GB of physical address space. Now, with this 16GB address space, we can comfortably increase RAM, ROM, and I/Os. This explains how we usually scale up a hardware system even with 32-bit RISC-V chips. Still, 4GB of virtual address space is limited for desktop/server OS and applications.

Also, general purpose 32-bit operating systems struggle to manage bigger DRAM chips. A 32-bit OS kernel can comfortably manage a DRAM chip of maximum size 32GB; hence, we prefer 64-bit operating systems on 64-bit SoCs. So, let's explore the 64-bit memory map.

RISC-V 64-bit SoC – Desktop and Server Chips

As shown in the figure, we follow a 36-bit memory map to design 64-bit SoCs.

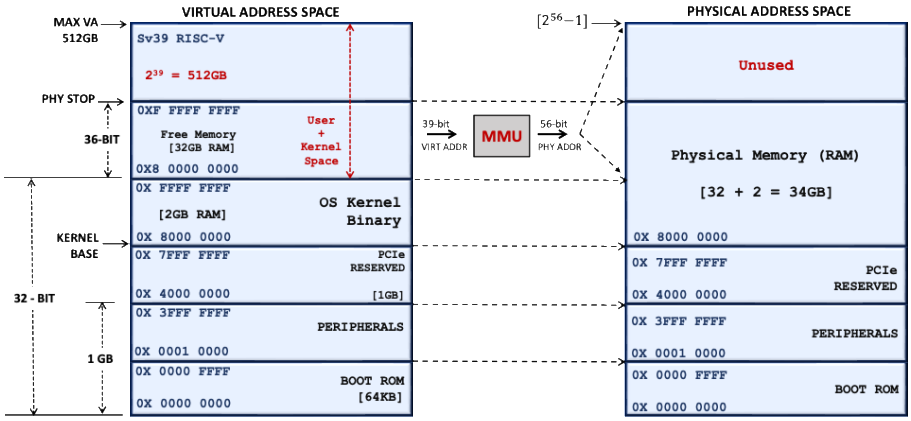


Using 36 bits (address bus), we can design a 64-bit SoC comfortably with all I/O interfaces, controllers, and memories within a 64GB address space. This allows us to expand the DRAM more than 30 to 40 GB and realize any kind of bare-metal complex embedded and OS-based desktop applications.

Usually, we prefer a 36/40-bit memory map to design 64-bit RISC-V SoCs, rather than simply using a 64-bit address space (address bus). Also, RISC-V offers various address translation schemes - Sv39, Sv48, Sv57, and Sv64 - to increase the virtual and physical address space for 64-bit SoCs, as shown below.

Scheme	Virtual Address Width	Physical Address Width	Page Table Levels	Page Sizes	Max Virtual Address Space	Max Physical Address Space	Target Use Case
Sv32	32 bits	34–40 bits	2 levels	4 KiB, 2 MiB	4 GiB	Up to 1 TiB	Embedded & 32-bit platforms
Sv39	39 bits	Up to 56 bits	3 levels	4 KiB, 2 MiB, 1 GiB	512 GiB	Up to 64 PiB	Standard 64-bit OSes
Sv48	48 bits	Up to 56 bits	4 levels	4 KiB, 2 MiB, 1 GiB	256 TiB	Up to 64 PiB	Servers, virtualization
Sv57	57 bits	Up to 56 bits	5 levels	4 KiB, 2 MiB, 1 GiB	128 PiB	Up to 64 PiB	Large-scale compute/data centers
Sv64	64 bits	Up to 56 bits	6 levels	4 KiB, 2 MiB, 1 GiB	16 EiB	Up to 64 PiB	Reserved for future extreme scaling

Let’s explore the RISC-V Sv39 address translation scheme on a 64-bit RISC-V SoC.



The Sv39 address translation scheme generates a 39-bit virtual address with 512GB of virtual address space for software applications and a 56-bit physical address with 64PB of physical address space, as shown above. The 32-bit memory map is always a subset of the 36/40-bit memory map for backward compatibility—porting 32-bit software (OSes, bootloaders & drivers) to 64-bit systems. Generally, the firmware, bootloaders and test codes are 32-bit software even for 64-bit SoCs. It executes without MMUs; hence all peripherals must be implemented within the physical address space - following physical addressing, as shown in the above figure.

Innovating high-performance electronic products requires more than just creativity - it demands a deep understanding of design approaches like 32-bit and 64-bit SoCs, along with modern system-level design strategies. System level designers must know the nuts and bolts of processor technologies like **RISC-V ISA**, and **SoC** and **Software** design methodologies as well.

Our **RISC-V Powered Executive MTech VLSI Design** course empowers system designers with essential knowledge of RISC-V ISA, SoC architecture, and software design methodologies - building the foundation to create the next generation of smart, scalable systems.