# [RISC-V] [HW-Designer] Which "optional" debug IP features are needed by TRACE32?

2024-11-20 - Comments (0) - RISC-V

Some parts of the RISC-V debug specification are very generic and leave room for optional features and optional implementations. Several debug registers of the Debug Module (DM), bitfields of these registers or whole features are declared as 'optional'. The following list tries to summarize all optional features, debug registers as well as all debug registers that contain optional bitfields, and describes whether they are currently needed by our RISC-V debugger.

Please note that this list can change at any time in the future, due to new versions of our RISC-V debugger with extended set of features.

If a register or bitfield is not in the below list, then the RISC-V debug specification sees it as non-optional, and consequently it is always required by our RISC-V debugger. Or we forgot to add it (in which case please contact us).

- **Abstract Data 0 - 11 (data0 - data11):**
  - The required amount of dataX registers depends on several factors, such as base ISA (RV32, RV64, ...) of all cores that are connected to the RISC-V debug module. Also, it depends on the implemented set of features in the RISC-V Debug Module, in particular the implemented abstract commands. Please refer to the RISC-V debug specification for details.
    In most standard RISC-V chips the following applies: if the debug module only addresses RV32 cores then at least data0 is needed, however if the debug module addresses at least one RV64 core then at least data0 and data1 are needed.
  - **Scratch memory**:
    - In certain situations, the debugger needs to use the 'data' debug registers as "scratch memory". See below point about 'hartinfo' for details.
    - If this is necessary, then the minimum required amount of data registers may be bigger than what is mentioned above (depending on

the actual scenario in which the scratch memory is used). Our current recommendation in this case would be at least 4 data registers (no matter if RV32 or RV64), but the actual required size might be bigger depending on the scenario.

- **Debug Module Control (dmcontrol):**
  - **'hasel' bitfield:** Recommended for multicore debugging, as this allows a synchronized start and stop of multiple RISC-V cores via hardware.
    If there are multiple RISC-V cores connected to a debug module and 'hasel' is not supported, then the debugger can not start/stop all cores simultaneously, but only consecutively.
    If there is only one RISC-V core connected to a debug module, then this bit is not needed.

- **Debug Module Status (dmstatus):**
  - Mandatory / always required as defined by RISC-V debug standard.

- **Hart Info (hartinfo):**
  - **'data' debug registers as scratch memory**:
    - In certain situations, the debugger needs to use "scratch memory" (i.e. memory that can be used for debug operations). One example is accessing double-precision floating point registers (FLEN=64bit) of a RV32 core (XLEN=32bit). If the 'data' debug registers are shadowed in the hart's memory map, then they can be used as such scratch memory.
    - If it is required that the 'data' debug registers are used as scratch memory, then it is necessary that the 'hartinfo' debug register exists, and their 'dataaccess', 'datasize' and 'dataaddr' fields are set accordingly.
    - An alternative to using the 'data' debug registers as scratch memory is using the program buffer as scratch memory. See below points regarding program buffer for details.
  - In addition to the above-mentioned use-cases, there may of course be other use-cases added to our RISC-V debugger in the future, which may make it necessary that the hartinfo register is implemented.
- **Halt Summary (haltsumN)**
  - Mandatory / always required as defined by RISC-V debug standard. Which haltsum register is required does depend on the number of harts connected to the DM.
- **Hart Array Window Select (hawindowsel) + Hart Array Window (hawindow)**
  - Mandatory / always required as defined by RISC-V debug standard, if the dmcontrol.hasel bitfield is supported (see above).
- **Abstract Control and Status (abstractcs)**

    - Mandatory / always required as defined by RISC-V debug standard.
- **Abstract Command (command)**
    - Mandatory / always required as defined by RISC-V debug standard.
    - For the required abstract command types, please see "Abstract Command Based Approach" below.

- **Abstract Command Autoexec (abstractauto):**
    - Although this register is declared as optional, we highly recommend to implement it. The 'autoexecdata[0]' feature bit allows significant(!) performance boosts when executing multiple consecutive abstract commands. This is e.g. extensively used during memory read/write via program buffer or abstract command.
      Our debug driver does however also support designs that do not support this optional feature.

- **Configuration Structure Pointer (confstrptrN)**
    - For rules regarding the existence of these registers, see RISC-V debug standard and dmstatus.confstrptrvalid.
- **Next Debug Module (nextdm)**
    - For rules regarding the existence of this register, see RISC-V debug standard
- **Custom Features (custom)**
    - Currently not supported by our debugger
- **Program Buffer 0 - 15 (progbuf0 - progbuf15):** The debug specification allows program buffer sizes between 0 and 16. A size of 0 means that there is no program buffer at all. In general there are two main ways of implementing a valid RISC-V debug IP (both of which are described in the RISC-V debug specification in more detail):

    - **Abstract Command Based Approach:** All main operations and interactions of the debugger are done via abstract commands. In this case, no program buffer is needed, so program buffer size can be 0.
        - However, the current ratified debug specifications v0.13.x/0.14.x/1.0.x do not provide a proper mechanism to detect/discover if a certain abstract command type is supported, and which individual options of an abstract command are supported. In particular, this affects the debugger system discovery for the **'access memory' abstract command** , which is why Lauterbach can currently only provide limited support for this abstract command. See the discussions of the official RISC-V debug workgroup for more details. Due to these discrepancies regarding system discovery of abstract commands, **we currently recommend to use the "Execution Based" approach** instead until the issues with system discovery are resolved.

That is why the 'access memory' abstract command can currently only be supported by our debugger, if all possible options and combinations of the 'aamsize', 'aampostincrement' and 'write' bitfields of that command are supported by the hardware.
To tell the debugger to use this abstract command as the default method for memory access, configure **SYStem.MemAccessStop AAM**.

- **Flexibility** when abstract command based:
  - If the debugger does not have a program buffer, then it has significantly less flexibility with regard to the operations that it can execute (see execution based approach below for comparison).
- **Cache coherency** when abstract command based: the 'access memory' abstract command and the RISC-V debug specification in general do not provide a standardized mechanism to operate on instruction- or data caches of the CPU. If there is no program buffer, then the debugger can not ensure cache coherency by itself, so the hardware needs to automatically ensure cache coherency on its own (e.g. execute cache flushes when the debugger initiates a respective access memory abstract command - if needed). See FAQ question "How are instruction- and data caches handled by the Lauterbach RISC-V debugger?" for details.

- **Execution Based Approach:** The main operations and interactions of the debugger do partially rely on the **program buffer** execution.

  - In this case, the Lauterbach debugger currently uses the program buffer in a variety of scenarios, and there might be added additional scenarios in the future, or existing scenarios and behaviors change due to bug fixes or improvements.

    **Program buffer size:**
    Our Lauterbach RISC-V debugger does automatically adapt its behavior to the program buffer size of the target.
    It is difficult to recommend a certain (small) program buffer size without knowing whether this size might be sufficient for all features in the future. Also, the required program buffer size depends on whether the 'implicit ebreak' feature is implemented (dmstatus.impebreak).

    The RISC-V debug standard allows a **program buffer size of 1** (with implicit ebreak) or 2 (without implicit ebreak). This minimum size is also supported by our Lauterbach debugger with software versions from 2024-07-16 or later, to cover most debug features. However, we

**strongly advise against** using such a small program buffer. The reason is that with such a small program buffer many of the performance optimization features that are defined within the RISC-V debug standard can not be used. In certain scenarios, such as for example a large memory access via program buffer, this can lead to a performance decrease of factor 4x -- 5x, compared to a hardware design with program buffer size of at least 2 (+ abstractauto.autoexecdata[0] feature).
A **program buffer size of 2** (with implicit ebreak) or 3 (without implicit ebreak) is the minimum size that we can recommend in order to take advantage of certain performance optimization features (in combination with the abstractauto.autoexecdata[0] feature).
One exception to the above recommendations may be using the program buffer as scratch memory, in which case larger program buffers are needed (see below).
Furthermore, it is not unlikely that an advanced and improved future RISC-V debugger might require (or at least could make good use of) a larger program buffer.

In summary, in case that a hardware designer wants to stay on the safe side for future updates of the Lauterbach RISC-V debugger, and if chip area is not quite that important, then we suggest a full program buffer size of 16. However, the minimum size mentioned above is also sufficient for certain basic debug operations. Please note the significant performance difference between program buffer sizes 1 and 2 (with implicit ebreak, respectively). Also, in case that the driver will add functions that require a size larger than the minimum size mentioned above, we will always try to leave a "fallback" option with possibly reduced functionality for hardware with smaller program buffer sizes.

- **Scratch memory**:
  - In certain situations, the debugger needs to use "scratch memory" (i.e. memory that can be used for debug operations). One example is accessing double-precision floating point registers (FLEN=64bit) of a RV32 core (XLEN=32bit). If the program buffer data debug registers (progbuf0, progbuf1, ...) are shadowed in the hart's memory map, and is writable by the hart, then the program buffer can be used as scratch memory.
  - In that case, the debugger will execute the 'auipc' instruction in the program buffer, in order to determine the program buffer's address from perspective of the hart.
  - If it is necessary to use the program buffer as scratch memory, then the minimum required program buffer size may be bigger

than what is stated above. Our current recommendation in this case would be at least 4 program buffer registers, but the actual required size might be bigger depending on the scenario.
    - An alternative to using the program buffer as scratch memory is using the 'data' debug registers as scratch memory. See above points regarding 'Abstract Data' registers and the 'hartinfo' register for details.
  - **Flexibility** when execution based:
    - Having access to a program buffer grants the debugger significantly more flexibility, as the operations that it can execute are not limited to the specific operations of the abstract commands.
  - **Cache coherency** when execution based: the RISC-V debug specification in general does not provide a standardized mechanism to operate on instruction- or data caches of the CPU. However, the existence of a program buffer allows the debugger to execute "FENCE" and/or "FENCE.I" instructions. This allows the debugger to ensure cache coherency, by executing these instructions before/after certain debugger operations. See FAQ question "How are instruction- and data caches handled by the Lauterbach RISC-V debugger?" for details.

- **Recommendation**: from debugger point of view, we recommend to implement the **execution based approach** with a program buffer size of at least 2 (with implicit ebreak), because it grants significantly more flexibility to the debugger than the abstract command based approach.

- **Authentication Data (authdata)**
  - For rules regarding the existence of this register, see RISC-V debug standard and dmstatus.authenticated/authbusy.

- **Debug Module Control and Status 2 (dmcs2)**
  - For rules regarding the existence of this register, see RISC-V debug standard
- **System Bus Address (sbaddress0/1/2) + System Bus Data (sbdata0/1/2/3):**
  - These optional registers are only needed if you want to access memory via the system bus (see 'SB:' access class of the RISC-V debugger). The debugger can also work without system bus access. Whether you need the system bus access or not depends on the individual needs of the user and on the target architecture.

Tags

Hardware Designer

RISC-V