Introduce RISC-V debugging and openocd

Xiang W <wxjstz@126.com / wangxiang@nj.iscas.ac.cn>

Outline

- Introduction
- GDB Remote Serial Protocol
- JTAG
- Target
- RISC-V debugging specification

Introduction

- openocd is an open source debugger used to communicate with GDB and control the adapter to communicate with specific debugging targets.
- openocd can also control the target chip through the JTAG interface to program the flash

GDB Remote Serial Protocol

- This protocol is based on the C/S model, with openocd as the server and gdb as the client
- This protocol is mainly used for status query, read and write registers, read and write memory, operation control (step,continue), breakpoint,and multi-thread related control
- Please refer to GDB Remote Serial Protocol

GDB Remote Serial Protocol

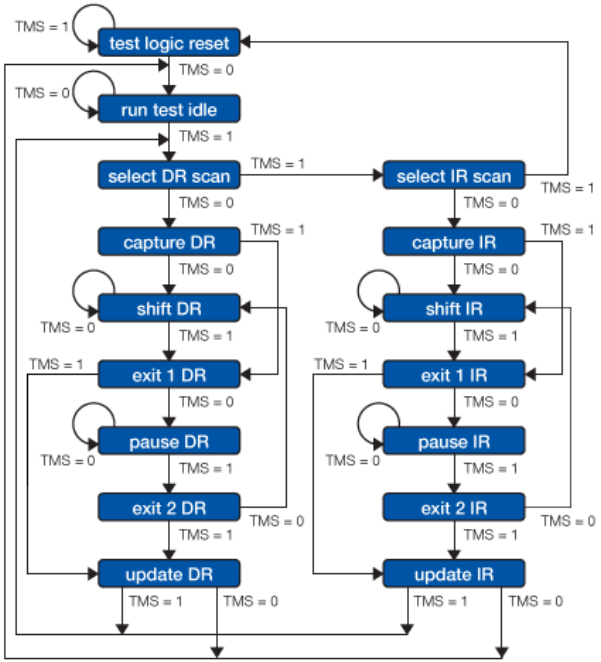
- **The gdb remote Serial protocol related code is mainly located in:**
  - src/server/gdb_server.h
  - src/server/gdb_server.c
- **The specific operations are implemented through the interfaces provided by the target. These interface declarations are located in:**
  - src/target/target.h

JTAG

- **JTAG is the most commonly used debugging interface, which consists of the following signals:**
  - TMS
  - TDO
  - TDI
  - TCK
  - TRST(optional)
  - SRST(optional)

JTAG

- JTAG is controlled by a state machine



JTAG

- It can be found through the above state machine that two types of registers IR/DR can be accessed through the JTAG interface
- Usually there is only one IR, which is used to select the DR to be accessed.
- The bit width of IR is determined by specific platform
- The bit width of each DR and the function and address of the DR are also determined by the specific platform

JTAG

- **Due to the flexibility of the JTAG interface, openocd maintains a state machine. The relevant code is located at:**
  - src/jtag/interface.h
  - src/jtag/interface.c
- **And abstract the JTAG operation command, the relevant code is located in ：**
  - src/jtag/commands.h
  - src/jtag/commands.c

JTAG

- **The main function to be implemented by the jtag driver is to execute these JTAG commands. The JTAG driver interface is as follows:**

```
struct jtag_interface {
        unsigned supported;
        int (*execute_queue)(void);
};
```

- **The specific debugged Traget realizes the debugging function through these JTAG interfaces**

Target

- **The target corresponds to the object being debugged. Generally need to define the bit width of IR, each DR bit width, address and function. This is called the debug specification**
- **Currently RISC-V has two versions of debugging specifications.**
  - 0.11: The hardware is simple to implement, most of the functions are realized through the monitor program
  - 0.13: rich functions, need more hardware support

Target

- **In order to describe the target, openocd defines a structure target, which records some debugging-related status and control information. The relevant code is located at:**
  - src/target/target.h
- **Because a platform may have multiple debugging specifications, they are described by the structure target_type. Which mainly records debugging-related operation handles. The relevant code is located at:**
  - src/target/target_type.h

Target

- **The target implementation of RISC-V is located at:**
  - src/target/riscv/riscv.h
  - src/target/riscv/riscv.c
- **The target_type related codes corresponding to the two versions of the debugging specifications are located at:**
  - src/target/riscv/riscv-011.c
  - src/target/riscv/riscv-013.c

RISC-V debugging specification

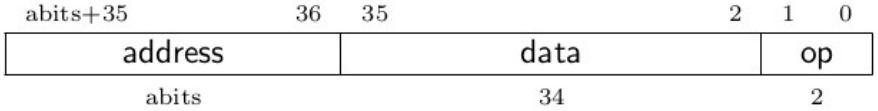
- **RISC-V currently has two debugging specifications 0.11 and 0.13**
  - 0.11: The hardware is simple to implement, most of the functions are realized through the monitor program
  - 0.13: rich functions, need more hardware support

RISC-V debugging specification (0.11)

- **IR bit width: 5 bits**
- **The main registers related to debugging**
  - IDCODE 32bits at 0b00001
  - dtmcontrol 32bits at 0b10000
  - dbus abits+36 bits at 0b10001 (abits from dtmcontrol, used to indicate the size of the address space of the debug module)

RISC-V debugging specification (0.11)

- dbus view



- dbus is readable and writable, and the meaning of op is as follows

op	write meaning	read meaning
0	nop	previous operation completed
1	read from address	Reserved
2	write data to address	previous operation failed
3	Reserved	previous operation is still in progress

RISC-V debugging specification (0.11)

- A memory space that can be read and written by JTAG is created through dbus, which is called DM (Debug Module)
- There are some registers on the DM for controlling debugging and accessing the system bus. These registers are as follows：

Table 4: Debug Module Debug Bus Registers

Address	Name
0x10	Control
0x11	Info
0x12	Authentication Data
0x13	Authentication Data
0x14	Serial Data
0x15	Serial Status
0x16	System Bus Address 31:0
0x17	System Bus Address 63:32
0x18	System Bus Data 31:0
0x19	System Bus Data 63:32
0x1b	Halt Notification Summary
0x3d	System Bus Address 95:64
0x3e	System Bus Data 95:64
0x3f	System Bus Data 127:96

RISC-V debugging specification (0.11)

- There is a small section of memory in the DM, which is used to store debugging instructions. This section of memory is mapped to the system bus at 0x100. This memory is called debug RAM
- System bus 0x800 has a fixed program, here is the entrance for debugging exceptions. This program will jump to the debug RAM for execution according to the type of debug exception. This memory is called debug ROM

RISC-V debugging specification (0.11)

- Many people must be confused why the DM data width is 34 bits. The highest bit is used to trigger the debug interrupt and quickly execute the instructions in the debug RAM. The second highest bit is used for halt notification.
- The hart to be halted is selected by the register dmcontrol.hartid on the DM. Support up to 1024 hart.
- By cooperating with Debug ROM and Debug RAM, you can access the register memory CSR and set software breakpoints, etc. The result of the debug instruction needs to be written back to the debug RAM, and then openocd can read the result through the JTAG interface

RISC-V debugging specification (0.11)

- There are a series of registers on the DM for reading the system bus, which can be used to access memory, peripherals, etc. These registers are as follows: sbaddress0,sbaddress1,sbaddress2,sbdata0,sbdata1,sbdata2,sbdata3

RISC-V debugging specification (0.11)

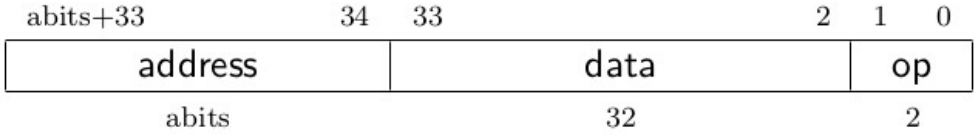
- Trigger is implemented through a series of CSRs, so that it can be accessed through openocd operation and M mode
- Can realize breakpoint watchpoint and instruction count

RISC-V debugging specification (0.13)

- **IR bit width: at least 5 bits**
- **The main registers related to debugging**
  - IDCODE 32 bits at 0x1
  - dtmcs 32 bits at 0x10
  - dmi abits+34 bits at 0x11(abits from dtmcontrol, used to indicate the size of the address space of the debug module)

RISC-V debugging specification (0.13)

- dmi view



- dbus is readable and writable, and the meaning of op is as follows

op	write meaning	read meaning
0	nop	previous operation completed
1	read from address	Reserved
2	write data to address	previous operation failed
3	Reserved	previous operation is still in progress

RISC-V debugging specification (0.13)

- In the debugging specification 0.13, there can be multiple DMs. The first DM is located at address 0. In each DM, there is a nextdm that points to the address of the next DM. When nextdm is 0, it means that there is no next DM.

RISC-V debugging specification (0.13)

- **Hart select(all operations need to select hart first)**
  - select a single hart: set hart id to dmcontrol.hartsel and set dmcontrol.hasel
  - select multiple hart: two registers (hawindowsel, hawindow) are used for selection. The nth bit in hawindow is set to 1, indicating that the hart with the hart id equal to hawindowsel*32+n is selected. Note that dmcontrol.hasel needs to be cleared at this time

RISC-V debugging specification (0.13)

- **Hart status**
  - There are some bits in dmstatus to identify the status of hart：allnonexistent, anynonexistent, allunavail, anyunavail, allrunning, anyrunning, allhalted, and anyhalted.
- **Run control**
  - It is controlled by some bits in dmcontrol, these bits are: haltreq,resumereq and hartreset

RISC-V debugging specification (0.13)

- **Abstract Command**
  - The abstract command is implemented through a command and a series of data0...data11
  - The highest 8 bits of command are used to identify the command type, and the definition of other bits is different when the command type is different
  - Through virtual commands, you can access the registers, the system bus, and the code in the Program Buffer has been executed

RISC-V debugging specification (0.13)

- **Program Buffer**
  - The 16 instructions in the DM space allow the hart to execute the instructions given by the debugger
- **System Bus Access**
  - Supports 8/16/32/64/128-bit wide access to the system bus.
  - sbsc: system bus access control and status
  - sbaddres0-sbaddress3: address need to access
  - sbdata0-sbdata3: read data for sbaddress


Think You !