http://marc.info/?l=openocd-development&m=137182292914653

<b>[prev in list] [<a
List: openocd-development
Subject: <a href="http://marc.info/?t=137182295200005&r=1&w=2">[OpenOCD-devel] Instructions on
From: <a href="http://marc.info/?a=122743986700004&r=1&w=2">Paul Fertser <fercerpav () gmai
Date: <a href="http://marc.info/?l=openocd-development&r=1&w=2&b=201306">2013-06-21 13:51:2
Message-ID: <a href="http://marc.info/?i=20130621135124.GC2326%20%28%29%20home%20%21%20lan">20130
[<a href="http://marc.info/?l=openocd-development&m=137182292914653&q=raw">Download message RAW</

Hi,

It wasn't the first time I tried and failed to find any guides on
using JTAG for its original purpose, so I felt like trying it on my
own. It's still unclear how to communicate with several different
devices on a chain at the same time as OpenOCD seems to require to
have only one TAP in non-bypass mode at a time.

Here go draft instructions:

1. You need a BSDL file for the components you're using. For STM32s
it's readily available from the vendor's website.

2. From the BSDL file you need to figure out the Boundary Scan
Register Length, e.g. for STM32F100 it's shown in this line:
attribute BOUNDARY_LENGTH of STM32F1_Low_Med_density_value_LQFP64 : entity is 232;

3. It's followed by
attribute BOUNDARY_REGISTER of STM32F1_Low_Med_density_value_LQFP64 : entity is

which describes which bits of BSR correspond to which device's ports.

4. Read the description of the port you're interested in. E.g. PC8
is described by

```
      "75       (BC_1,  *,              CONTROL,        1),                             " &
      "74       (BC_1,  PC8,            OUTPUT3,        X,      75,     1,      Z),     " &
      "73       (BC_4,  PC8,            INPUT,          X),                             " &
```

which means that bit 73 reflects port's input (when it's configured as
input), bit 74 defines port's output (when it's configured as output),
bit 75 sets PC8 to Z-state when set to 1 and to output when set to 0.

5. Decide on what mode you need: in SAMPLE/PRELOAD mode the buffers
are disconnected from the boundary scan logic and are controlled by
the cpu as usual but you can still sample their values. In EXTEST mode
the buffers are fully controlled by the boundary scan logic. Some SoCs
(including STM32) allow to do boundary scan while SRST is held low,
that makes it impossible for CPU to interfere with the test. You can
control SRST state with "jtag_reset" command.

6. Source manual_bs.tcl (attached) and call "init_bs <bstap>
<bsrlength>". This should be done after "init" call.

7. Proceed with your tests by calling "sample_get_bit_bsr <bitn>" and
other functions from manual_bs.tcl

An example of a semi-automated boundary scan test for an
STM32VLDiscovery board is attached, here follows the log:

```
$ sudo openocd -f interface/raspberrypi-native.cfg -f target/stm32f1x.cfg -f stm32vldiscovery_bs.
Open On-Chip Debugger 0.8.0-dev-00011-g7b21292-dirty (2013-05-09-23:11)
Licensed under GNU GPL v2
For bug reports, read
        <a href="http://openocd.sourceforge.net/doc/doxygen/bugs.html" rel="nofollow">http://open
Info : only one transport option; autoselect 'jtag'
BCM2835 GPIO config: tck = 11, tms = 25, tdi = 10, tdi = 9
```

```
adapter speed: 1000 kHz
adapter_nsrst_delay: 100
jtag_ntrst_delay: 100
cortex_m3 reset_config sysresetreq
Info : clock speed 1006 kHz
Info : JTAG tap: stm32f1x.cpu tap/device found: 0x3ba00477 (mfg: 0x23b, part: 0xba00, ver: 0x3)
Info : JTAG tap: stm32f1x.bs tap/device found: 0x06420041 (mfg: 0x020, part: 0x6420, ver: 0x0)
Info : stm32f1x.cpu: hardware has 6 breakpoints, 4 watchpoints


Starting basic STM32VLDiscovery JTAG boundary scan test

All LEDs should be OFF, press Enter

Green LED should be ON, blue LED OFF, press Enter

Green and blue LEDs should be ON, press Enter

Blue LED should be ON, green LED OFF, press Enter

Green and blue LEDs should be ON, do NOT press the USER button, press Enter

Green and blue LEDs should be ON, DO press the USER button, press Enter

Green and blue LEDs should be ON, do NOT press the USER button, press Enter

Green and blue LEDs should be ON, DO press the USER button, press Enter

All tests passed SUCCESSFULLY, exiting
shutdown command invoked

--
Be free, use free (<a href="http://www.gnu.org/philosophy/free-sw.html" rel="nofollow">http://www
mailto:fercerpav@gmail.com

<a href="http://marc.info/?l=openocd-development&m=137182292914653&q=p3"><b>["manual_bs.tcl" (app

# Init global variables to work with the boundary scan register
# the first argument is tap name, the second is BSR length
proc init_bs {tap len} {
    global bsrtap bsrlen
    set bsrtap $tap
    set bsrlen $len
    init_bsrstate
    # disable polling for the cpu TAP as it should be kept in BYPASS
    poll off
    sample_mode
}

# In this mode BSR doesn't control the outputs but can read the current
# pins' states, the CPU can continue to function normally
proc sample_mode {} {
    global bsrtap
    # SAMPLE/PRELOAD
    irscan $bsrtap 2
}

# Connect BSR to the boundary scan logic
proc extest_mode {} {
    global bsrtap
    # EXTEST
    irscan $bsrtap 0
}

# Write bsrstateout to target and store the result in bsrstate
proc exchange_bsr {} {
```

```
    global bsrtap bsrstate bsrstateout
    update_bsrstate [eval drscan [concat $bsrtap $bsrstateout]]
    return $bsrstate
}

# Check if particular bit is set in bsrstate
proc get_bit_bsr {bit} {
    global bsrstate
    set idx [expr $bit / 32]
    set bit [expr $bit % 32]
    expr ([lindex $bsrstate [expr $idx*2 + 1]] & [expr 2**$bit]) != 0
}

# Resample and get bit
proc sample_get_bit_bsr {bit} {
    exchange_bsr
    get_bit_bsr $bit
}

# Set particular bit to "value" in bsrstateout
proc set_bit_bsr {bit value} {
    global bsrstateout
    set idx [expr ($bit / 32) * 2 + 1]
    set bit [expr $bit % 32]
    set bitval [expr 2**$bit]
    set word [lindex $bsrstateout $idx]
    if {$value == 0} {
        set word [format %X [expr $word & ~$bitval]]
    } else {
        set word [format %X [expr $word | $bitval]]
    }
    set bsrstateout [lreplace $bsrstateout $idx $idx 0x$word]
    return
}

# Set the bit and update BSR on target
proc set_bit_bsr_do {bit value} {
    set_bit_bsr $bit $value
    exchange_bsr
}

proc init_bsrstate {} {
    global bsrtap bsrlen bsrstate bsrstateout
    set bsrstate ""
    for {set i $bsrlen} {$i > 32} {incr i -32} {
        append bsrstate 32 " " 0xFFFFFFFF " "
    }
    if {$i > 0} {
        append bsrstate $i " " 0xFFFFFFFF
    }
    set bsrstateout $bsrstate
    return
}

proc update_bsrstate {state} {
    global bsrstate
    set i 1
    foreach word $state {
        set bsrstate [lreplace $bsrstate $i $i 0x$word]
        incr i 2
    }
}

<a href="http://marc.info/?l=openocd-development&m=137182292914653&q=p4"><b>["stm32vldiscovery_bs

# Example script to test STM32VLDiscovery with boundary scan
```

```
init

echo "\n\nStarting basic STM32VLDiscovery JTAG boundary scan test\n"

source manual_bs.tcl

init_bs stm32f1x.bs 232
extest_mode
exchange_bsr
echo "All LEDs should be OFF, press Enter"
read stdin 1

# Set PC9 to output 1
set_bit_bsr 72 0
set_bit_bsr_do 71 1
echo "Green LED should be ON, blue LED OFF, press Enter"
read stdin 1

# Set PC8 to output 1
set_bit_bsr 75 0
set_bit_bsr_do 74 1
echo "Green and blue LEDs should be ON, press Enter"
read stdin 1

# Set PC9 to output 0
set_bit_bsr_do 71 0
echo "Blue LED should be ON, green LED OFF, press Enter"
read stdin 1

# Set PC9 to output 1
set_bit_bsr_do 71 1
foreach i {0 1} {
    echo "Green and blue LEDs should be ON, do NOT press the USER button, press Enter"
    read stdin 1
    # Read PA0 state, there's a pulldown on board
    if {[sample_get_bit_bsr 187] == 1} {
        echo "Button is stuck at 1: ERROR, aborting"
        shutdown
        return
    }

    echo "Green and blue LEDs should be ON, DO press the USER button, press Enter"
    read stdin 1
    if {[sample_get_bit_bsr 187] == 0} {
        echo "Button is stuck at 0: ERROR, aborting"
        shutdown
        return
    }
}

echo "All tests passed SUCCESSFULLY, exiting"
shutdown
```
