XAPP571 (v1.0.1) January 27, 2005

DEBUGHALT Controller for PowerPC Boot and Reset Operations

Author: Peter Ryser

Summary

The DEBUGHALT controller is a small, yet versatile piece of FPGA logic that simplifies the startup process of the PowerPC™ 405 (PPC405) processors in systems that cannot have any memory at the reset vector, or in systems that completely run out of cache.

This application note is accompanied by a reference design that demonstrates *debug halt mode* implemented in the embedded PPC405 processor available on Virtex-II Pro™ FPGAs. The DEBUGHALT controller design enables external control of the PPC405 processor through the JTAG interface. The DEBUGHALT controller example design targets the ML310 Embedded Development Platform.

Introduction

The PPC405 debug interface allows external control of the processor for altering normal program execution. It provides the ability to debug system hardware as well as software through the JTAG port. The DEBUGHALT controller utilizes this external-debug feature to allow the user to control when and how the processor starts and stops executing instructions at power-up, reset, or any other time.

With the DEBUGHALT controller, the user can keep the processor at the reset vector (after configuration of the FPGA) until all processor registers, processor caches, peripheral registers, block RAM, external registers, and external memory have been configured and loaded with data and/or executable code.

The DEBUGHALT controller can be used in systems without memory at the reset vector, or in systems that completely run out of cache. See XAPP575, *UltraController II: Minimal Footprint Embedded Processing Engine*, for an example of the latter. Through the processor's JTAG interface, the user loads code and data, writes the start address of the program into the program counter, and starts execution.

Mode of Operation

Figure 1 illustrates the DEBUGHALT controller interface signals and the internal logic.

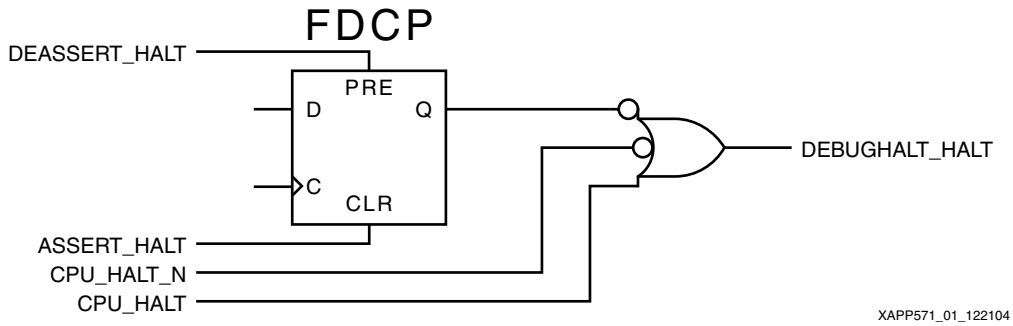

Figure 1: DEBUGHALT Controller Block Diagram

The DEBUGHALT controller consists of a single flip-flop used in set/reset mode, followed by an OR-gate, to combine other sources for the DEBUGHALT_HALT output signal with the output of the flip-flop. Upon configuration of the FPGA the flip-flop is zero; DEBUGHALT_HALT is asserted until the DEASSERT_HALT signal is set high or is toggled at least once. When the ASSERT_HALT signal is asserted high, or toggled at least once, DEBUGHALT_HALT is

asserted. In a typical application, DEBUGHALT_HALT connects to the DBGC405DEBUGHALT signal of the PowerPC processor.

Controller I/O Signals

Table 1 lists the I/O signals for the DEBUGHALT controller.

Table 1: **DEBUGHALT Controller I/O Signals**

Signal Name	Direction	Description
CPU_HALT_N	Input	Negated user halt input. Can be directly connected to the CPU_HALT pin on the BDM connector. Tie high if not used.
CPU_HALT	Input	User halt input. Tie low if not used.
ASSERT_HALT	Input	Asserts the DEBUGHALT_HALT output. Tie low if not used.
DEASSERT_HALT	Input	Deasserts the DEBUGHALT_HALT output.
DEBUGHALT_HALT	Output	DEBUGHALT output. Typically, directly connected to the DBGC405DEBUGHALT pin of the processor.

Table 2 shows the logic table for the DEBUGHALT controller.

Table 2: **DEBUGHALT Controller Logic Table**

CPU_HALT_N	CPU_HALT	ASSERT_HALT	DEASSERT_HALT	DEBUGHALT_HALT
0	X	x	X	1
x	1	x	X	1
1	0	1	X	1
1	0	0	1	0

Controller Instantiation

To instantiate the DEBUGHALT controller in an EDK design, copy the debughalt_v1_00_a directory from the xapp571.zip to the pcores directory of your design. Modify the MHS file of your project and add the following lines:

```
BEGIN debughalt
 PARAMETER INSTANCE = debughalt_0
 PARAMETER HW_VER = 1.00.a
 PORT DEBUGHALT_HALT = dbgc405debughalt_0
 PORT DEASSERT_HALT = c405dbgmsrwe_0
END
```

Connect the DEBUGHALT_HALT and DEASSERT_HALT signals to the PPC405 processor in your design by adding two lines to the ppc405 instance:

```
BEGIN ppc405
 [...]
 PORT DBGC405DEBUGHALT = dbgc405debughalt_0
 PORT C405DBGMSRWE = c405dbgmsrwe_0
END
```

In this example, only the DEBUGHALT_HALT and the DEASSERT_HALT signals are used. The other signals are not used and are initialized to their defaults.

Example Applications

Pass-through Mode

Tie DEASSERT_HALT high and ASSERT_HALT low to pass through CPU_HALT_N and/or CPU_HALT signals. This disables the DEBUGHALT controller, making the logic function depend only on the CPU_HALT_N and CPU_HALT input signals. Table 3 shows the truth table for this application.

Table 3: **DEBUGHALT Controller in Pass-through Mode**

CPU_HALT_N	CPU_HALT	ASSERT_HALT	DEASSERT_HALT	DEBUGHALT _HALT
0	0	0	1	1
0	1	0	1	1
1	0	0	1	0
1	1	0	1	1

Asserting DEBUGHALT_HALT after Power-Up until Counter Expires

The carry-out of a counter is hooked up to the DEASSERT_HALT pin. ASSERT_HALT is tied low. DEBUGHALT_HALT will be asserted until the counter flows over for the first time. This setup is used to keep the CPU halted for a specific amount of time after power-up.

Asserting DEBUGHALT_HALT after Power-Up and Reset until Counter Expires

The carry-out of a counter is hooked up to the DEASSERT_HALT pin. A reset signal, for example system reset (SYS_RST), is hooked up to ASSERT_HALT and the reset port of a counter. DEBUGHALT_HALT will be asserted as long as reset is active plus the time until the counter flows over for the first time. This setup is used to keep the CPU halted for a specific amount of time after power-up or reset.

Asserting DEBUGHALT_HALT after Power-Up until CPU Pulses C405DBGMSRWE

This setup is ideal to configure the software-related part of the system after the FPGA becomes active. Additionally, it enables a processor to boot without having executable code at the reset vector. See "Controller Instantiation" to connect the DEBUGHALT controller in this mode.

The processor stays halted after power-up until the configuration data pulses the C405DBGMSRWE signal, for example, as part of an ACE file, a concatenated hardware and software bitstream, or manual interaction through an external debugger.

Tie ASSERT_HALT low and connect C405DBGMSRWE to DEASSERT_HALT. Figure 2 shows the setup of the DEBUGHALT controller in combination with the PowerPC processor.

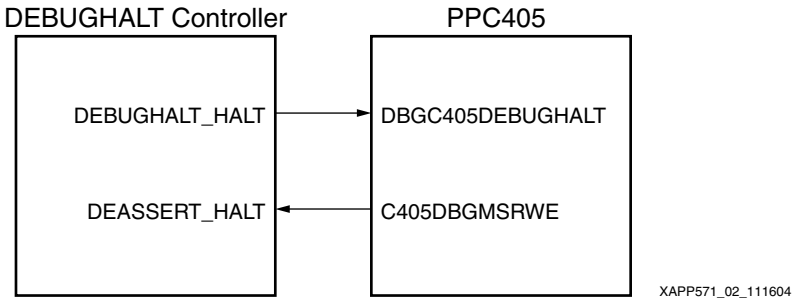

Figure 2: **Typical Connection of the DEBUGHALT Controller and the PPC405**

Example Session

Using a JTAG Debugger

The following is an example session using iMPACT and XMD to configure the hardware and load code into caches:

1. Use iMPACT to download the bitstream.
2. Start XMD and connect to the CPU mapping the caches into the memory space.

```
XMD% connect ppc hw -debugdevice icachestartadr 0x70000000 dcachestartadr 0x78000000
```

3. Pulse DEASSERT_HALT by writing 0x40000 and 0 to the MSR register. The CPU will remain stopped because the debugger keeps the processor stopped.

```
XMD% rwr msr 0x40000
XMD% rwr msr 0
```

4. Load the software into the caches and set the program counter to the start address.

```
XMD% dow
```

5. Start the processor from the new start address.

```
XMD% run
```

Using an ACE File

Modify the genace.tcl script provided by EDK to generate an ACE file that works properly with the DEBUGHALT controller. Open a xygwin shell and copy the $XILINX_EDK/data/xmd/genace.tcl into the data directory of your project. Edit genace.tcl and add three lines to toggle the DEASSERT_HALT signal in front of the commands that download the ELF file to the processor. The modified genace.tcl will look like this:

```
#puts "xconnect Done..\n\n"

#puts "toggling DEASSERT_HALT\n\n"
xwreg $tgt msr 0x40000
xwreg $tgt msr 0x0

xdownload $tgt $elffile
```

Using the DEBUGHALT Controller in a Multiprocessor Environment

In an FPGA with multiple PPC405 processors, all processors can share a single DEBUGHALT controller, or each processor can have its own DEBUGHALT controller.

In the case where a single DEBUGHALT controller is used, one of the processors asserts the DEASSERT_HALT signal and starts both processors simultaneously. Such behavior may be desirable in cases where the processor need to run synchronized.

If each processor has its own DEBUGHALT controller, the processors can be started independently of each other. In one scenario one processor controls the DEASSERT_HALT pin of the DEBUGHALT controller of the second processor.

Example Projects

Two example projects located in the xapp571.zip file demonstrate the DEBUGHALT controller on an ML310 board. You must have EDK properly installed and set up to implement the example projects. Both projects are intentionally simple. The goal of this application note is to explain how the DEBUGHALT controller works. Examples that are too sophisticated would distract from that goal.

The first project (ml310/single) shows the most typical case, using an example of one processor. It demonstrates the use case where DEBUGHALT_HALT is asserted after power-up until the CPU pulses C405DBGMSRWE as shown in Figure 2, page 4. The EDK design consists of one active PowerPC 405 processor, a UART, and the DEBUGHALT controller. The software application is a program that runs directly out of the processor caches and prints **Hello World** through the UART to a serial console. In this case, the DEBUGHALT controller is used to delay the start of the processor until the debugger (XMD) has connected to the processor to load the software into the caches.

The second project (ml310/dual) makes use of both processors inside the Virtex-II Pro FPGA on the ML310 board. It demonstrates the use case of the DEBUGHALT controller in a multiprocessor environment where one processor controls the start-up of the second processor. The EDK design consists of two active PowerPC 405 processors, a UART, and two DEBUGHALT controllers. In other words, each of the processors has a DEBUGHALT controller attached, but both processors share the same UART. The debugger or System ACE™ CF controls the DEBUGHALT core on the first processor. The first processor then controls the DEBUGHALT core of the second processor through some GPIO output pins. The software for both processors operate out of caches that are controlled through the debugger. The software on the second processor, as in the single processor example of the previous project, prints **Hello World** to the serial console. The software for the first processor prints a message, starts the second processor, and then goes into an endless loop.

The setup of the second project shows how one processor can influence and delay the startup of the second processor. Such a behavior can be very useful if both processors use the same code base. One of the processors should only start *after* the first processor has initialized the system.

The build.sh and the run.sh scripts build and execute either one of the two projects. The build.sh script builds the hardware design, the software, and the ACE file for a given project. The run.sh script downloads the bitstream to the board and runs XMD to load and start the software. You are encouraged to study the scripts to learn about the steps to implement and run the reference designs.

References

1. EDK documentation, http://www.xilinx.com/ise/embedded/edk_docs.htm
   - Xilinx, Inc., UG018: PowerPC 405 Processor Block Reference Guide
2. ML310 evaluation platform website, http://www.xilinx.com/ml310
   - Xilinx, Inc., UG068: ML310 User Guide
3. XAPP575: UltraController II: Minimal Footprint Embedded Processing Engine, http://www.xilinx.com/bvdocs/appnotes/xapp575.pdf

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
12/16/04	1.0	Initial Xilinx release.
01/27/05	1.0.1	Updated to 2005 template and added references to XAPP575.