

TAP Master Utility Programs

This is an archive with source code in it that hopefully will help one write code that uses the 74LVT8980A JTAG controller. The JTAG tapmaster this code was used with was designed for the now obsolete ISA bus, but if anything that should make the code very easy to understand. The 74LVT8980A was mapped into the processor's I/O space with base address 0x0280. The register addresses are defined in the file TAP.H. You will see in the source reference to a driver called GIVEIO that under non-windows platforms can be ignored. A version of this code ran under windows NT4, and so needed to use the GIVEIO driver to unlock direct access to the processor's physical I/O space. The code was written to do JTAG transactions on a StrongARM SA1110 processor target, and unfortunately the relevant BSDL definitions are hard coded into it in the library source ARMPIT.C. But overall the code will show how things work and how the registers in the 74LVT8980A can be set up. In addition to the actual application programs described below, there are 4 useful support files.

TAP.H This has definitions for addressing the 74LVT8980A.

ARMPIT.H This has JTAG command definitions for the StrongARM processor.

TAPLIB.C Library of routines that control the 74LVT8980A.

ARMPIT.C Library of routines that perform JTAG transactions with the StrongARM target.

The Tap Master application programs allow you to perform the following functions:

- Verify proper operation of the tap master hardware.
- Read and write to FLASH memory on BadgePAD.
- Read and write to SRAM on BadgePAD.
- Obtain information about and contents of AXF files produced by the ARM development tools.

Each program is described below. All programs run in a simple command prompt window. Requirements to run these programs are Windows NT version 4.0 or later, and you must install and run the GIVEIO device driver. GIVEIO was originally documented in *Dr. Dobbs Journal* [1], and allows programmed I/O calls to be made from MSC. In addition to windows, these applications are easily ported to Linux.

Finally, note that the sources described here are free software; you can redistribute them and/or modify them under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

The programs described here are distributed in the hope that they will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

A copy of the GNU General Public License is available from the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

DUMPAXF

DUMPAXF dumps the contents of an AXF file from the ARM tools to stdout. This is very handy in order to compare the contents of an AXF file with what is actually in BadgePAD's FLASH or SRAM. The command syntax is:

```
DUMPAXF filename {-aPseudo_Start_Addr_in_hex -s16|32}
```

The `-a` and `-s` arguments are optional. DUMPAXF will dump out an AXF file in exactly the same format as DUMPFL or DUMPSR, and consists of address/data pairs on each line of output. *Filename* is the name of the file to dump, ie foo.axf. The `-s` argument allows you to dump out the data as 16 or 32 bit data items. The default is `-s16`. Note that DUMPFL will dump out data as 16 bit items, and DUMPSR will dump out data as 32 bit items, so you will use `-s16` or `-s32` as appropriate to compare data from an AXF file with FLASH or SRAM. The `-a` argument allows you to override the starting address information in the AXF header, and dump out address/data pairs with the first item in the file seeming to exist at the address you specified with the `-a` option. This is useful when the data in FLASH or SRAM that you wish to compare the AXF file to is stored in a different address than were it actually runs, either because it is dynamically relocated, or due to MMU considerations. By using the `-a` option, you can make the output of DUMPAXF correspond to the output of DUMPFL or DUMPSR, making it much easier to compare the resulting files using some tool like WINDIFF. The default for `-a` is whatever starting address is specified in the AXF file header.

DUMPFL

DUMPFL dumps the contents of FLASH memory to stdout. The command syntax is:

```
DUMPFL {-aStart_Addr_in_hex -cNumber_of_Bytes}
```

The `-a` and `-c` arguments are optional. DUMPFL will dump out the contents of FLASH as address/data pairs on each line of output. The addresses are 2 byte aligned reflecting the 16 bit FLASH width used on BadgePAD. The `-c` option allows you to specify how many *bytes* to dump out. Default is 64 bytes or 32 16 bit words. The `-a` option allows you to select the physical address in FLASH to start dumping from. The `-a` option defaults to the physical beginning of FLASH at 0x00000000. The `-a` option is specified using hexadecimal.

DUMPSR

DUMPSR dumps the contents of SRAM memory to stdout. The command syntax is:

```
DUMPSR {-aStart_Addr_in_hex -cNumber_of_Bytes}
```

The `-a` and `-c` arguments are optional. DUMPSR will dump out the contents of SRAM as address/data pairs on each line of output. The addresses are 4 byte aligned reflecting the 32 bit SRAM width used on BadgePAD. The `-c` option allows you to specify how many *bytes* to dump out. Default is 64 bytes or 16 32 bit words. The `-a` option allows you to select the physical address in SRAM to start dumping from. The `-a` option defaults to the physical beginning of SRAM at 0x08000000. The `-a` option is specified using hexadecimal.

ERASEFL

ERASEFL erases the entire FLASH memory system on BadgePAD. The command syntax is simply ERASEFL.

GETID

GETID reads and displays on stdout the identification register inside the StrongARM SA-1100 processor used on BadgePAD. The syntax is just GETID. When GETID is run the identification returned is: 0xV108406B where V is a 4 bit number corresponding to the silicon revision of the processor. GETID is useful to determine that all TAP master hardware is hooked up properly and is talking to BadgePAD. If GETID doesn't work, nothing else will either.

LOADFL

LOADFL will program the FLASH memory on BadgePAD. The syntax is:

```
LOADFL filename {-aStart_Addr_in_hex}
```

The `-a` argument is optional and is always a physical address in hexadecimal. *Filename* is the name of the AXF file to load into FLASH. The `-a` argument allows you to load an AXF file starting at a physical FLASH address other than that specified in the AXF file's header. This is useful when you are loading an application that will be either dynamically relocated, for example to SRAM, or will be assigned to a virtual address using the MMU. For example, an application compiled to run in SRAM starting at physical address 0x08000000 can be stored in FLASH and dynamically moved to SRAM. If the application is to be stored in FLASH at location 0x000a0000, the command

```
LOADFL filename -a0x000a0000
```

will put the application into FLASH as desired. The `-a` option defaults to the starting address specified in the AXF file.

LOADSR

LOADSR will store an AXF into the SRAM on BadgePAD. The syntax is:

```
LOADSR filename {-aStart_Addr_in_hex}
```

The `-a` argument is optional and is always a *physical* address in hexadecimal. *Filename* is the name of the AXF file to write into SRAM. The `-a` argument allows you to load an AXF file starting at a physical SRAM address other than that specified in the AXF file's header. This is useful when you are loading an application that will be either dynamically relocated, or will be assigned to a virtual address using the MMU. For example consider an application compiled to run in SRAM that will be configured by the MMU to appear at virtual address 0x00000000. We desire the application to be stored in SRAM starting from physical location 0x08000000. The command to do this would be:

```
LOADSR filename -a0x08000000
```

The `-a` option defaults to the starting address specified in the AXF file.

LOOPIT

LOOPIT is a handy utility that checks out the TAP master hardware in the host computer. LOOPIT will put the TAP master into a loopback mode, and read and write random bit streams to verify that the hardware is working properly. The syntax is just LOOPIT. The program starts by asking you to input an integer that it will use to seed a random number generator, and then off it goes, printing out diagnostics from time to time until you stop it by hitting ctrl-C. As with the program GETID, LOOPIT is handy to run after you get the TAP master board first installed into your computer and the GIVEIO driver running. If LOOPIT doesn't run reporting zero errors, nothing will work right.

SHOWAXF

SHOWAXF will open an AXF file and reveal useful data contained in the AXF file header, such as sizes of the various images that make up the file and address mode flags. SHOWAXF is run with the syntax:

SHOWAXF *filename*

Where *filename* is the name of the AXF file you want data revealed about. The most common uses of SHOWAXF are to determine if an AXF file will fit into available memory, or to determine the correct argument to use with the `-c` command of DUMPFL and DUMPSR. SHOWAXF will tell you the total byte count of the read only, read/wrote and zero init image spaces of an AXF file. This number can be used directly with the `-c` argument of DUMPFL or DUMPSR in order to generate a file that can be compared to the original file using DUMPAXF.

[1] *Direct Port I/O and Windows NT*, Dale Roberts, Dr. Dobb's Journal, May 1996 pp14-24