

AMT0 BitString Programming

June 15, 2000

J. Hollar and R. Ball, University of Michigan

Version 1.1

1. Introduction

The BitString utility is a subset of the ATLAS CSM0 MiniDAQ software, which allows users to program the AMT0 mezzanine cards without using a graphical user interface or the LabWindows/CVI software set. The code is portable to any standard C/C++ compiler, and has been tested in both LabWindows/CVI 4.0.1 and Microsoft Visual C++ 6.0.

2. Building the Project

The BitString program includes the following standard MiniDAQ files: *100F_JTAG.c*, *100F_JTAG.h*, *CSMDriver.c*, *CSMDriver.h*, *CSMRegisters.h*, *CSM_RW.c*, *JTAG_LL.c*, *JTAG_LL.h*, *JTAG_Mezzanine.c*, *JTAG_Mezzanine.h*, and *RW_100F.c*. These contain all the functionality needed to create setup and control bit strings for multiple AMT0s and scan them via JTAG. For detailed descriptions of these files see reference [1]. Within these files, calls to CVI libraries such as *utility.h* and *userint.h*, as well as the *ATLASinterface.lib* library, are enclosed using the conditional compilation directive `#ifdef _CVI_`. Corresponding Visual C++ libraries will be included under `#ifdef _MSC_VER`.

In addition, a source file containing 'main' and any required compiler-specific files should be included. In Visual C++, this can be done automatically by creating a Win32 Console Application from the **File > New > Projects** menu. The files listed above must then be included in the project, and a call to the external *JTAG_Mezzanine.c* function `No_GUI(void)` included in main as below. Basic initialization of the CSM-0 board is also illustrated.

```
extern "C" void No_GUI(void);
#include "CSMDriver.h"
#include "CSMRegisters.h"
#include "JTAG_mezzanine.h"

int main(int argc, char* argv[])
{
    MasterReset();
    SetJTAGRate(4);
    EnableSerial();
    No_GUI();
    return 0;
}
```

As in the regular ATLAS MiniDAQ, hardware-specific VME/VXI calls are isolated at the lowest level of the call tree in *CSM_RW.c*, and any *CSMDriver.c* function can be called.

3. Running the Project

To avoid the CVI graphical user interface, this project reads from formatted ASCII text files and interacts through the standard IO window to assemble the appropriate bit strings. Upon running the project, the user is prompted to enter the total number of attached TDCs, and either the TDC number to program or all TDCs should be programmed. The user is then prompted to enter a text file from which to read the 200-bit AMT0 setup string. Following this, another text file which contains the 24-channel settings must be entered. It is important that these text files be prepared in the format described below in section 4. The user will also be prompted to enter text files in which to save the AMT0 settings, if desired. Following this, programming will be performed automatically, with any error messages displayed to Standard IO.

If multiple CSM-0 modules are in use, then each will have a different base address in VME space, as set by the DIP switch on the VME board. The `No_GUI()` function as programmed above automatically programs the module resident at the default base address of `0x2000000`. The other CSM-0 modules can be used to program their attached AMT-0 using the same bit strings by first changing the base address then calling a routine from the `JTAG_mezzanine` suite to rerun the programming sequence. This is illustrated below.

```
No_GUI()           // Program the CSM at 0x2000000
SetCSM_Base(0x3000000); // Second CSM module is at 0x3000000
AMT0_Chain(3, 0);  // Three AMT-0 on this one (arg=3)...
                  // ... and program all of them (arg=0)
```

4. Creating Text Files

To use this software, the user must have two text files prepared, one for the setup bits and one for the channel control bits. The format expected for these files consists of separate formatted text lines for each parameter: a line order number followed by whitespace followed by a decimal value followed by whitespace followed by an optional comment followed by a carriage return/end line for each setting. For example, the first five lines of the setup string text file might look like:

```
1      160    asd array (8-bit value)
2      1      error mark (1-bit value)
3      0      error bypass (1-bit value)
4      1      vernier error (1-bit value)
5      0      coarse error (1-bit value)
```

The in-line comments are helpful but not required, since after reading the decimal value the program will skip over everything until it reaches an end-of-line. For more on AMT0 setup and control settings, see reference [2].

The setup and control values need not be listed in order in the text files. The order number preceding the actual setting determines where the value will be placed in the array. For example, any line numbered “1” should contain the asd setting, no matter where that line is placed in the text file. Mismatching line numbers or skipping values will result in incorrect setup and control settings. `BitString` also does not perform error checking for allowed settings, so the text files must be prepared correctly before running the program.

One method for doing this is to create the text files from the standard ATLAS MiniDAQ’s graphical user interface. After setting values via the `JTAG Mezzanine Programming` subpanel, the user has the option to write the setup and control settings to two text files. These will automatically be formatted to be compatible with the `BitString` program, with text comments added for each value and lines listed in numerical order. The text files created with this method can then be used immediately by any `BitString` program, or customized with any text editor before use. Standard versions of these files are distributed with the MiniDAQ. These files are named `setup.txt` and `control.txt`, containing the setup and control bit strings, respectively.

References

[1] “ATLAS MiniDAQ Reference Guide”, R. Ball, J. Chapman, J. Kuah, J. Hollar, University of Michigan, June 2000

[2] “AMT-0: ATLAS Muon TDC version 0 v1.0”, J. Christiansen, CERN, EP-MIC, July 1998