Using the TINI JTAG Library and SVF File to Program Xilinx PROM Devices

This application note explains how to use the TINI JTAG library to program Xilinx PROM devices, using a Serial Vector Format (SVF) file. It is assumed that the reader has some familiarity with JTAG and programmable logic.

Summary

This application note explains how to use the the TINI JTAG library to program Xilinx PROM devices, using a Serial Vector Format (SVF) file. It is assumed that the reader has some familiarity with JTAG and programmable logic.

Introductions

Tiny InterNet Interfaces (TINI) is a platform developed by Dallas Semiconductor. It is a combination of a small but powerful chip-set and a Java programmable runtime environment. The chip-set provides processing, control, device-level communication and networking capabilities. To communicate with any JTAG device, the TINIs400 socket board provides 4-pin JTAG output at the J21 terminal. These pins will wire directly to the standard JTAG pins TDI, TDO, TMS, and TCK of the JTAG device.

In-System Programmable PROMs can be programmed individually, or can be chained. All devices in the chain share the TCK and TMS signals. The TINI TDI signal is connected to the TDI input of the first device in the Boundary Scan Chain. The TDO signal from the first device is connected to the TDI input of the second device in the chain and so on. The last device in the chain has its TDO output connected to the TINI TDO pin as illustrated in Figure 1.

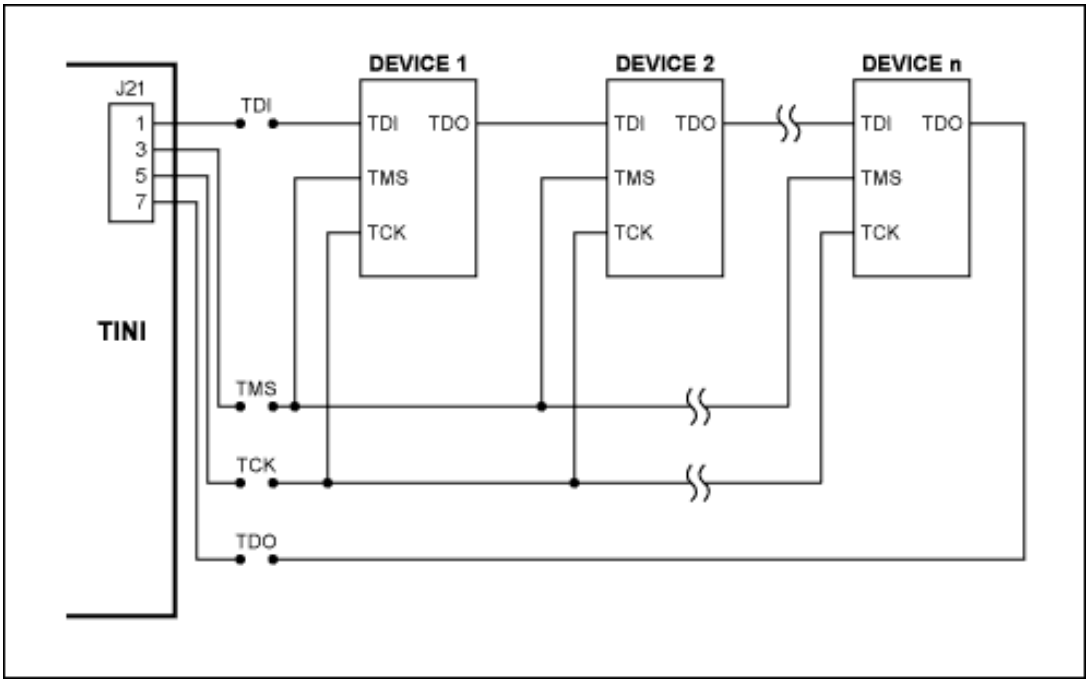

Figure 1. All JTAG operations are controlled through a device's Test Access Port

All JTAG operations are controlled through a device's Test Access Port (TAP). The TAP consists of four signals: TMS, TDI, TDO, and TCK. These signals interact with the device through the TAP controller, a 16-state finite state machine. The JTAG TMS signal controls transitions between states. Instructions and data are shifted into the device on the TDI

pin and are shifted out on the TDO pin. All state transitions and activity on the TDI and TDO signals are synchronous to TCK. See Figure 2.

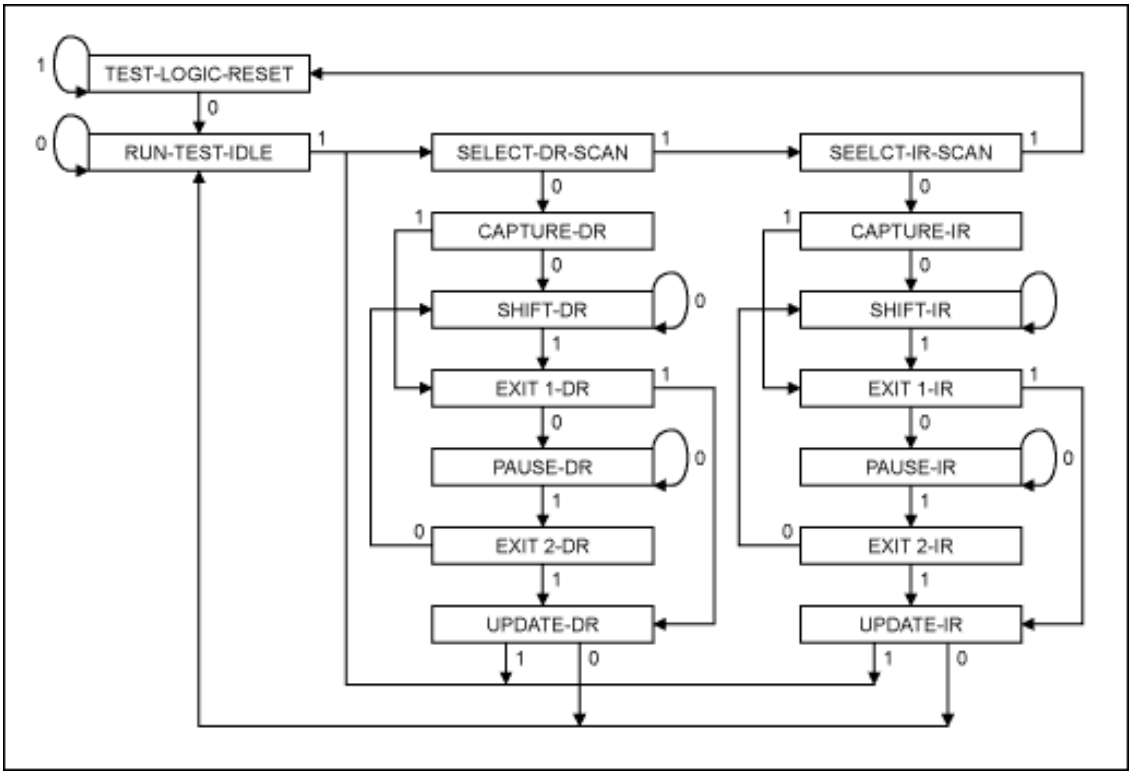

Figure 2

All JTAG operations shift data into or out of JTAG instruction and data registers. The TAP controller provides direct access to all of these registers. There are two classes of JTAG registers: the instruction register (IR) and data registers (DR). Access to the IR is provided through the Shift-IR state, while access to DR is provided through the Shift-DR state. The IR length is normally arbitrary size greater than 2 bits. Except BYPASS instruction defined by IEEE Std. 1149.1 is all ones, the manufacturer defines all other instructions bit codes.

In this application note, the JAVA sample code will interpret the Serial Vector Format (SVF) file to perform the programming. The SVF used here is a syntax specification for describing high-level IEEE 1149.1 (JTAG) bus operations. SVF has been adopted as a standard for data interchange by JTAG equipment and software manufacturers. It is used to describe JTAG chain operations in a compact, portable fashion. SVF files record JTAG operations by describing the information that needs to be shifted into device chain. The JTAG operations are recorded in the SVF file by the Xilinx iMPACT (details below) software. The SVF files are written as ASCII text and, therefore, can be read, modified, or written manually in any text editor. "Many third-party programming utilities use an SVF file as input and can program Xilinx devices in a JTAG chain with the information contained in the SVF file."

JTAG Library Brief Descriptions

The JTAG class library is developed to assist a user who uses TINI to communicate with JTAG devices. Dynamic, in-system update of programmable logic from a remote location is one possible application. A user can initialize TAP controller to initial state, browse 16 states, get or set any TAP state, generate clock signal, send commands, data and more. Below are brief descriptions of the JTAG class methods:

- public jtag(): loads jtag library.
- public String getVersion(): returns version of the jtag class.
- public int runClock(int numticks): runs clock with number specified by numticks.
- public byte initialize(): involves driving TMS high for five TCK pulses, and performs extra 1 clock with TMS low to enter default state Run-Test-Idle.
- public byte setState(byte aState): sets the object state to the state specified by aState.

- public byte getState(): returns the object state.
- public byte scanState(byte state): transitions the TAP controller state machine to the state indicated by the parameter state.
- public String displayState(byte state): displays the state of TAP controller.
- public byte waitState(int msecs): delays process in number of millisecond.
- public byte getTDO():gets the current state of pin TDO.
- public void setTDI(byte logic): sets pin TDI at the assigned logic.
- public byte getTDI(): gets the current state of pin TDI.
- public void setTMS(byte logic): sets pin TMS at the assigned logic.
- public byte getTMS():gets the current state of pin TMS.
- public void setTCK(byte logic): sets pin TCK at the assigned logic.
- public byte getTCK():gets the current state of pin TCK.
- public byte sendNrcv(byte[] data, int offset, int size, byte numberOfBits, boolean state, boolean update, byte extraHeaderClock, byte HeaderBitVal,byte extraTrailerClock, byte TrailerBitVal): sends byte array of data and receives byte array back.
- public byte sendNrcv(int[] data, int offset, int size, byte numberOfBits, boolean state, boolean update,byte extraHeaderClock, byte HeaderBitVal,byte extraTrailerClock, byte TrailerBitVal): sends integer array of data and receives integer array back.

For more details about JTAG class, visit ftp://ftp.dalsemi.com/pub/tini/appnotes/jtaglib

Hardware & Software Requirements

The following hardware & software are required:

- A TINI400 Evaluation board configured with TINI OS 1.12 or higher (See http://www.maxim-ic.com/appnotes.cfm/appnote_number/612)
- A JTAG device (programmable logic).
- SVF file that models the JTAG device within the device chain. For more information about SVF file, please view http://www.xilinx.com/bvdocs/appnotes/xapp503.pdf
- The TINI JTAG library. For more details about JTAG class, go to ftp://ftp.dalsemi.com/pub/tini/appnotes/jtaglib

Programming Procedures

Step 1: Wire 4 JTAG pins from TINI board to 4 JTAG pins of the JTAG device.
Step 2: Follow commands from SVF file and use JTAG library to write a JAVA application to program JTAG device, compile and load into TINI.
(Appendix A: A sample Idcode.svf file to read IDCODE from stand-alone XC18V02 Xilinx device) (Appendix B: Sample AppJtag.java program processes SVF file as input to generate JTAG signals to read IDCODE. The sample program is written to process stand-alone device (for chained device see Appendix C for details).
(Appendix C: Demonstrate procedures to program chained device from existing SVF file.)
Step 3: Execute JAVA program. The executable AppJtag.tini will be loaded to TINI board. Type "java AppJtag.tini Idcode.svf" at TINI prompt to run the program.) Program will generate JTAG signals to communicate with JTAG device.
Notes:
1/ The SVF file of the programmed JTAG device should be generated with exact model as designed in the device chain.
2/ All opcodes such as READ, WRITE, ERASE, and others of the programmed JTAG device are defined by manufacturer. The SVF file should contain all opcodes the user needs.

Example

The below example illustrates how to use JTAG library to read Idcode from Xilinx JTAG device XC18V02. For complete sample codes, please review appendix B.
1/ Generate device model SVF file:
Use Xilinx WEB START to generate SVF file.

- Execute iMPACT and select radio button "Prepare Configuration Files" option, then hit next. Select radio button "Boundary Scan file" and click next. Select "SVF File" then click finish.
- Type name for SVF file such as example in the dialog and click OK.
- Choose name_of_mcs_file.mcs file to load and select PROM device from the list (XC18V02_vq44)
- XC18V02 has been added to the model.
- Click on the model the device we want to program XC18V02. The device will be hi-lighted.
- Right mouse click and choose program option. Check "Get Idcode"
- Click the mouse outside the model to disable hi-lighted device and right click and choose option to close VSF. At this point the SVF file of the model was generated.

2/ Read VSF file and use JTAG library to program XC18V02 device

- Below is partial sample of SVF file:

```
// Created using Xilinx iMPACT Software [ISE WebPACK - 5.1i]
TRST OFF;
ENDIR IDLE;
ENDDR IDLE;
STATE RESET IDLE;
TIR 0 ;
HIR 0 ;
TDR 0 ;
HDR 0 ;
// Validating chain...
TIR 0 ;
HIR 0 ;
TDR 0 ;
HDR 0 ;
SIR 8 TDI (ff) SMASK (ff) ;
TIR 0 ;
HIR 5 TDI (1f) SMASK (1f) ;
HDR 1 TDI (00) SMASK (01) ;
TDR 0 ;
//Loading device with 'idcode' instruction.
SIR 8 TDI (fe) SMASK (ff) ;
SDR 32 TDI (00000000) SMASK (ffffffff) TDO (05025093) MASK (ffffffff) ;
//Loading device with 'conld' instruction.
SIR 8 TDI (f0) ;
RUNTEST 110000 TCK;
```

- Follow SVF command and use library to send command to XC18V02.
- SVF specification provides four global padding instructions: Header Instruction Register (HIR), Trailer Instruction Register (TIR), Header Data Register (HDR), and Trailer Data Register (TDR). These global commands specify the number of bits to pad the beginning and the end of a shift operation, to account for bypassed devices, and provide a simple method of SVF file compression. Once specified, these bits lead or follow every set of bits shifted for the SIR or SDR commands.
- Below are two examples how to interpret from SVF command to JTAG library command.

Example 1: SVF syntax with global padding instructions

```
a/ SVF file commands:
STATE RESET IDLE;
TIR 0 ;
HIR 5 TDI (1f) SMASK (1f) ;
HDR 1 TDI (00) SMASK (01) ;
```

```
TDR 0 ;
//Loading device with 'idcode' instruction.
SIR 8 TDI (fe) SMASK (ff) ;
SDR 32 TDI (00000000) SMASK (ffffffff) TDO (05025093) MASK (ffffffff) ;
//Loading device with 'conld' instruction.
SIR 8 TDI (f0) ;
RUNTEST 110000 TCK;
//Check for Read/Write Protect.
SIR 8 TDI (ff) TDO (01) MASK (ff) ;
//Loading device with 'idcode' instruction.
SIR 8 TDI (fe) ;
SDR 32 TDI (00000000) TDO (05025093) ;
//Loading device with 'conld' instruction.
SIR 8 TDI (f0) ;
RUNTEST 110000 TCK;
//Check for Read/Write Protect.
SIR 8 TDI (ff) TDO (01) ;
TIR 0 ;
HIR 0 ;
TDR 0 ;
HDR 0 ;
```

b/ Sample JAVA program using JTAG library:

```
import java.io.*;
import javax.comm.*;
import com.dalsemi.comm.*
public static void main(String[] args)
{
myJtag = new jtag();
int SIZE = 0x1000;
byte[] byteArray = new byte[SIZE];
byte HeaderInstBitVal = (byte)0x00;
byte TrailerInstBitVal = (byte)0x00;
byte HeaderDataBitVal= (byte)0x00;
byte TrailerDataBitVal= (byte)0x00;
// STATE RESET IDLE;
myJtag.initialize();//This JTAG library method will initialize
                    // XC18V02 TAP controller and stay at
                    // Run-Test/Idle
// TIR 0 ;
byte TIR = (byte)0x00;
// HIR 5 TDI (1f) SMASK (1f) ;
byte HIR = (byte)0x05;
HeaderInstBitVal = (byte)0x01;
// HDR 1 TDI (00) SMASK (01) ;
byte HDR = (byte)0x01;
HeaderDataBitVal = (byte)0x00;
// TDR 0
byte TDR = (byte)0x00;
// SIR 8 TDI (fe) SMASK (ff) ;
byteArray[0] = (byte)0xfe;
myJtag.sendNrcv(byteArray,0,1,(byte)8,true,false,(byte)HIR,
(byte)HeaderInstBitVal,(byte)TIR,(byte)TrailerInstBitVal);
// SDR 32 TDI (00000000) SMASK (ffffffff) TDO (05025093) MASK
// (ffffffff) ;
byteArray[0] = (byte)0x00;
byteArray[1] = (byte)0x00;
```

```
byteArray[2] = (byte)0x00;
byteArray[3] = (byte)0x00;
myJtag.sendNrcv(byteArray,0,4,(byte)8,false,false,(byte)HDR,
(byte)HeaderDataBitVal,(byte)TDR,(byte)TrailerDataBitVal);
// SIR 8 TDI (f0) ;
byteArray[0] = (byte)0xf0;
myJtag.sendNrcv(byteArray,0,1,(byte)8,true,false,(byte)HIR,
(byte)HeaderInstBitVal,(byte)TIR,(byte)TrailerInstBitVal);
// RUNTEST 110000 TCK;
myJtag.waitState(100); // Depend on operation frequency, user can
                        // calculate how many miliseconds to wait.
}
```

Example 2: SVF syntax without global padding instructions

```
a/ SVF file commands:
STATE RESET IDLE;
TIR 0 ;
HIR 0 ;
TDR 0 ;
HDR 0 ;
SIR 13 TDI (1fff) SMASK (1fff) ;
SDR 2 TDI (00) SMASK (03) ;
SIR 13 TDI (1fff) TDO (0021) MASK (1c63) ;
// Loading devices with 'ispen' or 'bypass'  instruction.
SIR 13 TDI (1d1f) ;
SDR 7 TDI (68) SMASK (7f) ;
// Loading device with 'faddr' instruction.
SIR 13 TDI (1d7f) ;
SDR 17 TDI (000002) SMASK (01ffff) ;
RUNTEST 1 TCK;
// Loading device with 'ferase' instruction.
SIR 13 TDI (1d9f) ;
RUNTEST 100000 TCK;
// Loading device with a 'faddr' instruction.
SIR 13 TDI (1d7f) ;
SDR 17 TDI (000002) ;
RUNTEST 1 TCK;
// Loading device with 'serase' instruction.
SIR 13 TDI (015f) ;
RUNTEST 37000 TCK;
// Loading devices with 'conld' or 'bypass' instruction.
SIR 13 TDI (1e1f) ;
RUNTEST 110000 TCK;
// Loading devices with 'ispen' or 'bypass'  instruction.
SIR 13 TDI (1d1f) ;
SDR 7 TDI (68) SMASK (7f) ;
// Loading device with a 'fdata0' instruction.
SIR 13 TDI (1dbf) ;
b/ Sample JAVA program using JTAG library:
import java.io.*;
import javax.comm.*;
import com.dalsemi.comm.*
public static void main(String[] args)
{
```

```
myJtag = new jtag();
int SIZE = 0x1000;
int[] intArray = new int[SIZE];
byte HeaderInstBitVal = (byte)0x00;
byte TrailerInstBitVal = (byte)0x00;
byte HeaderDataBitVal= (byte)0x00;
byte TrailerDataBitVal= (byte)0x00;
// STATE RESET IDLE;
myJtag.initialize();//This JTAG library method will initialize
          // XC18V02 TAP controller and stay at
          // Run-Test/Idle
// TIR 0 ;
byte TIR = (byte)0x00;
// HIR 0 ;
byte HIR = (byte)0x00;
// HDR 0 ;
byte HDR = (byte)0x00;
// TDR 0
byte TDR = (byte)0x00;
// SIR 13 TDI (1fff) SMASK (1fff) ;
intArray[0] = (int)(0x1fff&0x1fff);
myJtag.sendNrcv(intArray,0,1,(byte)13,true,false,(byte)HIR,
(byte)HeaderInstBitVal,(byte)TIR,(byte)TrailerInstBitVal);
// SDR 2 TDI (00) SMASK (03) ;
intArray[0] = (int)(0x00&0x03);
myJtag.sendNrcv(intArray,0,1,(byte)2,false,false,(byte)HDR,
(byte)HeaderDataBitVal,(byte)TDR,(byte)TrailerDataBitVal);
// SIR 13 TDI (1fff) TDO (0021) MASK (1c63) ;
intArray[0] = (byte)0x1fff;
myJtag.sendNrcv(intArray,0,1,(byte)13,true,false,(byte)HIR,
(byte)HeaderInstBitVal,(byte)TIR,(byte)TrailerInstBitVal);
// SIR 13 TDI (1d1f) ;
intArray[0] = (byte)0x1d1f;
myJtag.sendNrcv(intArray,0,1,(byte)13,true,false,(byte)HIR,
(byte)HeaderInstBitVal,(byte)TIR,(byte)TrailerInstBitVal);
// SDR 7 TDI (68) SMASK (7f) ;
intArray[0] = (int)(0x68&0x7f);
myJtag.sendNrcv(intArray,0,1,(byte)7,false,false,(byte)HDR,
(byte)HeaderDataBitVal,(byte)TDR,(byte)TrailerDataBitVal);
// SIR 13 TDI (1d7f) ;
intArray[0] = (byte)0x1d7f;
myJtag.sendNrcv(intArray,0,1,(byte)13,true,false,(byte)HIR,
(byte)HeaderInstBitVal,(byte)TIR,(byte)TrailerInstBitVal);
// SDR 17 TDI (000002) SMASK (01ffff) ;
intArray[0] = (int)(0x0002 & 0xffff);
TDR = (byte)0x01;
myJtag.sendNrcv(intArray,0,1,(byte)16,false,false,(byte)HDR,
(byte)HeaderDataBitVal,(byte)TDR,(byte)TrailerDataBitVal);
// RUNTEST 1 TCK;
myJtag.waitState(1);
// SIR 13 TDI (1d9f) ;
intArray[0] = (byte)0x1d9f;
myJtag.sendNrcv(intArray,0,1,(byte)13,true,false,(byte)HIR,
(byte)HeaderInstBitVal,(byte)TIR,(byte)TrailerInstBitVal);
// RUNTEST 100000 TCK;
myJtag.waitState(1000);
```

```
// SIR 13 TDI (1d7f) ;
intArray[0] = (byte)0x1d7f;
myJtag.sendNrcv(intArray,0,1,(byte)13,true,false,(byte)HIR,
(byte)HeaderInstBitVal,(byte)TIR,(byte)TrailerInstBitVal);
// SDR 17 TDI (000002) ;
intArray[0] = (int)(0x0002 & 0xffff);
TDR = (byte)0x01;
myJtag.sendNrcv(intArray,0,1,(byte)16,false,false,(byte)HDR,
(byte)HeaderDataBitVal,(byte)TDR,(byte)TrailerDataBitVal);
// RUNTEST 1 TCK;
myJtag.waitState(1);
// SIR 13 TDI (015f) ;
intArray[0] = (byte)0x015f;
myJtag.sendNrcv(intArray,0,1,(byte)13,true,false,(byte)HIR,
(byte)HeaderInstBitVal,(byte)TIR,(byte)TrailerInstBitVal);
// RUNTEST 37000 TCK;
myJtag.waitState(100);
// SIR 13 TDI (1e1f) ;
intArray[0] = (byte)0x1e1f;
myJtag.sendNrcv(intArray,0,1,(byte)13,true,false,(byte)HIR,
(byte)HeaderInstBitVal,(byte)TIR,(byte)TrailerInstBitVal);
// RUNTEST 110000 TCK;
myJtag.waitState(110);
// SIR 13 TDI (1d1f) ;
intArray[0] = (byte)0x1d1f;
myJtag.sendNrcv(intArray,0,1,(byte)13,true,false,(byte)HIR,
(byte)HeaderInstBitVal,(byte)TIR,(byte)TrailerInstBitVal);
// SDR 7 TDI (68) SMASK (7f) ;
intArray[0] = (int)(0x68 & 0x7f);
TDR = (byte)0x00;
myJtag.sendNrcv(intArray,0,1,(byte)7,false,false,(byte)HDR,
(byte)HeaderDataBitVal,(byte)TDR,(byte)TrailerDataBitVal);
// SIR 13 TDI (1dbf) ;
intArray[0] = (byte)0x1dbf;
myJtag.sendNrcv(intArray,0,1,(byte)13,true,false,(byte)HIR,
(byte)HeaderInstBitVal,(byte)TIR,(byte)TrailerInstBitVal);
}
```

Conclusions

It is quite simple to program Xilinx PROMs devices with a provided SVF file. However, it will be better if the user should be able to analyze SVF file to understand how to determine which device within the device chain to be programmed. Knowing this information, programmer can use JTAG library more efficiently. It has practical application for software team that wants to distribute Xilinx JTAG devices programming to remote location. It is easy done by embedded JTAG programming software in the package with mcs file as input file.

APPENDIX A
COMPLETE SAMPLE OF IDCODE.SVF FILE TO GET DEVICE IDCODE

```
// Created using Xilinx iMPACT Software [ISE WebPACK - 5.1i]
TRST OFF;
ENDIR IDLE;
ENDDR IDLE;
STATE RESET IDLE;
TIR 0 ;
HIR 0 ;
```

```
TDR 0 ;
HDR 0 ;
// Validating chain...
TIR 0 ;
HIR 0 ;
TDR 0 ;
HDR 0 ;
SIR 13 TDI (1fff) SMASK (1fff) TDO (0021) MASK (1c63) ;
TIR 0 ;
HIR 5 TDI (1f) SMASK (1f) ;
HDR 1 TDI (00) SMASK (01) ;
TDR 0 ;
//Loading device with 'idcode' instruction.
SIR 8 TDI (fe) SMASK (ff) ;
SDR 32 TDI (00000000) SMASK (ffffffff) TDO (05025093) MASK (ffffffff) ;
//Loading device with 'conld' instruction.
SIR 8 TDI (f0) ;
RUNTEST 110000 TCK;
//Check for Read/Write Protect.
SIR 8 TDI (ff) TDO (01) MASK (ff) ;
//Loading device with 'idcode' instruction.
SIR 8 TDI (fe) ;
SDR 32 TDI (00000000) TDO (05025093) ;
//Loading device with 'bypass' instruction.
SIR 8 TDI (ff) ;
TIR 0 ;
HIR 0 ;
TDR 0 ;
HDR 0 ;
SIR 13 TDI (1fff) SMASK (1fff) ;
SDR 2 TDI (00) SMASK (03) ;
```

APPENDIX B
EXAMPLE TO READ IDCODE OF XILINX DEVICE XC18V02 AS STAND ALONE THE DEVICE.

```
NOTES: This sample program will process SVF file with global padding instructions only.
Users need to modify to work with SVF file without global padding.
import java.io.*;
import java.lang.*;
import javax.comm.*;
import com.dalsemi.comm.*;
public final class AppJtag
{
private static jtag myJtag;
// Starting index of data array
static int Offset = 0;
// Number of byte to send within the data array
static   int Size = 0;
// Number of bits/byte or bits/integer
static   byte NumberOfBits = 8;
// State = true (Instruction); State = false (Data)
static   boolean State = true;
// For Xilinc device this variable always false
static   boolean Update = false;
// For Instruction header and trailer
```

```
static   byte HIR = 0 , TIR = 0 ;
// For Data header and trailer
static   byte HDR = 0 , TDR = 0;
// For Instruction header and trailer bit value
static   byte HeaderBitInstVal = 1, TrailerBitInstVal = 1;
// For Data header and trailer bit value
static   byte HeaderBitDatVal = 0, TrailerBitDatVal = 0;
static   int Key;
static   byte[] myData = new byte[4096];
static   String line;
static   RandomAccessFile SVFFile;
static   File mySVF ;
public static int ProcessSVFLine()
{
int  loop;
int OBracket = 0, CBracket = 0 ;
int myKey = 0x0;
StringBuffer buffer;
byte myAdd = (byte)0x0;
// Check the size in bits
myKey = Integer.parseInt(line.substring(line.indexOf(' ')+1,line.indexOf('
',line.indexOf(' ')+1)));
if (myKey != 0)  // myKey != 0
{
// Find value of data in byte format
OBracket = line.indexOf('(');
if ( myKey/8 > line.length())
{
// Copy data string to buffer starting from open bracket to
// end of data line
buffer = new StringBuffer(line.substring(OBracket+1,line.length()));
// Get another line of data and append to buffer
try
{
line = SVFFile.readLine();
while(line != null)
{
         // Find close bracket
         CBracket = line.indexOf(')',0);
         if ( CBracket < 0) //Can not find close bracket
         {
                buffer.append(line.substring(0,line.length()));
                line = SVFFile.readLine();
         }
         else // Find close bracket
         {
                 buffer.append(line.substring(0,CBracket));
                 line = null;
         }
}
loop = 0;
// Store data into array to send
for (int x = buffer.length(); x > 0; x--)
{
myData[loop] =(byte) Integer.parseInt((buffer.toString()).substring(x-2,x),16);
```

```
loop++;
x--;
}

}
catch(IOException a)
{
System.out.println("Cannot read data "+ a);
}
}
else
{
// Line of data is within one line.");
CBracket = line.indexOf(')',OBracket+1);
if (Key <= 3  || Key == 6) // Process header & trailer
{
  if (Key != 6)
  {
          // determine value of header and trailer
if(Integer.parseInt(line.substring(OBracket+1,CBracket).substring(0,2),16)>0)
          myAdd = (byte)0x01;
         else
          myAdd = (byte)0x00;
  }
}
else // Process SIR & SDR
{
  loop = 0;
  for (int s = CBracket - (OBracket+1); s >0 ; s--)
  {
  myData[loop] =(byte)
Integer.parseInt((line.substring(OBracket+1,CBracket)).substring(s-2,s),16);

  loop++;
  s--;
  }
}
}
}
switch(Key)
{
case 0: //System.out.println("TIR & TrailerBitInstVal ");
TIR = (byte)myKey;
TrailerBitInstVal = myAdd;
break;
case 1: //System.out.println("HIR & HeaderBitInstVal ");
HIR = (byte)myKey;
HeaderBitInstVal = myAdd;
break;
case 2: //System.out.println("TDR & TrailerBitDatVal ");
TDR = (byte)myKey;
TrailerBitDatVal = myAdd;
break;
case 3: //System.out.println("HDR & HeaderBitDatVal ");
HDR = (byte)myKey;
HeaderBitDatVal = myAdd;
```

```
break;
case 4:
case 5: Size = myKey;
break;
default:
System.out.println("Invalid opcode.");
return 0;
}

return 1;
}

public static void main(String[] args)
{
// The key words of the SVF file include:
// TRST OFF: Ignore
// ENDIR IDLE: Ignore
// ENDDR IDLE: Ignore
// STATE RESET IDLE: Ignore
// TIR 5 TDI (1f): Key word = 0, length, pin, value(Instruction
// trailer)
// HIR 5 TDI (1f): Key word = 1, length, pin, value(Instruction
// header)
// TDR 5 TDI (00): Key word = 2, length, pin, value(Data trailer)
// HDR 5 TDI (00): Key word = 3, length, pin, value(Data header)
// SIR 13 TDI (1fff)): Key word = 4, length, pin,
// value(Instruction)
// SDR 32 TDI (00000000)): Key word = 5, length, pin, value(Data)
// RUNTEST 110000 TCK: Key word = 0, clock pulse, pin (Run clock)
/*
// Command send format for Instruction.
myJtag.sendNrcv(Data,Offset,Size,(byte)NumberOfBits,State,Update,
(byte)HIR,(byte)HeaderBitInstVal,(byte)TIR,(byte)TrailerBitInstVal);

// Command send format for Data
myJtag.sendNrcv(Data,Offset,Size,(byte)NumberOfBits,State,Update,
(byte)HDR,(byte)HeaderBitDatVal,(byte)TDR,(byte)TrailerBitDatVal);
*/

myJtag = new jtag();
if(args.length !=  0)
{

for (int x = 0; x < args.length ; x++)
{
mySVF = new File(args[x]);
if (mySVF.exists())
{
if (mySVF.canRead())
{
try
{
   SVFFile= new RandomAccessFile(mySVF.getName(), "r");
   // Process one line at a time and set all variables
   while((line = SVFFile.readLine()) != null)
   {
```

```
        System.out.println(line);
        if(line.regionMatches(false,0,"TIR",0,3))
                Key = 0;
        else if(line.regionMatches(false,0,"HIR",0,3))
                Key = 1;
        else if(line.regionMatches(false,0,"TDR",0,3))
                Key = 2;
        else if(line.regionMatches(false,0,"HDR",0,3))
                Key = 3;
        else if(line.regionMatches(false,0,"SIR",0,3))
                Key = 4;
        else if(line.regionMatches(false,0,"SDR",0,3))
                Key = 5;
        else if(line.regionMatches(false,0,"RUNTEST",0,7))
                Key = 6;
        else if(line.regionMatches(false,0,"STATE",0,5))
                Key = 7;
        else
        {
                Key = 8;
                //System.out.println(line);
        }

switch(Key)
{
case 0: // TIR
if (ProcessSVFLine()==0)
System.out.println("Fail to fetch TIR and TrailerBitInstVal.");
break;
case 1: // HIR
if (ProcessSVFLine()==0)
System.out.println("Fail to fetch HIR and HeaderBitInstVal.");
break;
case 2: // TDR
if (ProcessSVFLine()==0)
System.out.println("Fail to fetch TDR and TrailerBitDatVal.");
break;
case 3: // HDR
if (ProcessSVFLine()==0)
System.out.println("Fail to fetch THR and HeaderBitDatVal.");
break;
case 4: // SIR
State = true;
if (ProcessSVFLine()==1)
myJtag.sendNrcv(myData,Offset,Size,(byte)NumberOfBits,State,Update,
(byte)HIR,(byte)HeaderBitInstVal,(byte)TIR,(byte)TrailerBitInstVal);
else
System.out.println("Fail to process ProcessSVFLine.");
break;
case 5: // SDR
State = false;
if (ProcessSVFLine()==1)
{

// Users need to retrieve read data from byte array if needed.
myJtag.sendNrcv(myData,Offset,Size,(byte)NumberOfBits,State,Update,
```

```
(byte)HDR,(byte)HeaderBitDatVal,(byte)TDR,(byte)TrailerBitDatVal);
}
else
System.out.println("Fail to process ProcessSVFLine.");
break;
case 6:
System.out.println("Run clock");
Size = Integer.parseInt(line.substring
(line.indexOf(' ')+1,line.indexOf(' ',line.indexOf(' ')+1))) ;
myJtag.waitState((Size/1000)+1);
break;
case 7:
System.out.println("Initialize TAP controller.");
myJtag.initialize();
myJtag.initialize();
myJtag.initialize();
break;
default:System.out.println("Ignore key word.");
break;
}
}
}
catch(IOException a)
{
System.out.println("Cannot create random file "+ a);

}
}
else
System.out.println(args[x]+" file cannot be read");
}
else
System.out.println(args[x]+" file does not exist");
}
// Process SVF file

}
else
System.out.println("SVF file is not specified in parameter");
}
}
```

APPENDIX C

HOW TO EXPAND THE SAMPLE SVF FILE TO WORK WITH CHAINED DEVICES.

Appendices A & B shows how to work with stand alone or single device. In real world, user needs to work with many Xilinx devices that chain together. For example, we have a chain of Xilinx devices connected in series as shown below:

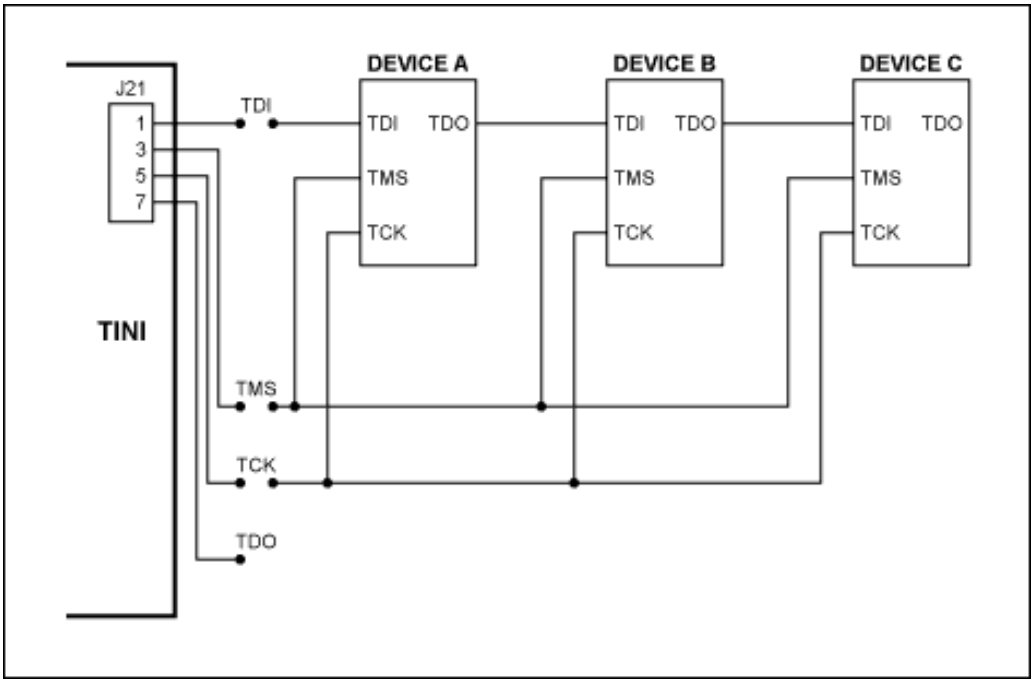

Figure 3.

Follow below procedures to work with device B in the chain:
Step 1: Generate SVF file for single device B as described above.
Step 2: Collect data for Xilinx devices such as Instruction Length Attribute, Instruction Opcode Attribute, and so on from BSDL file for all other device in the chain. For more details go to web site
http://www.xilinx.com/bvdocs/appnotes/xapp503.pdf
Step 3: Edit SVF file to modify the value of TIR, HIR, TDR, and HDR accordingly. Assuming that Device A has IR length of 5 bits, Device B has IR length of 6 bits, and Device C has IR length of 7 bits. If we want to work with device B, then we need to replace values of TIR, HIR, TDR, and HDR in the SVF as follows:

```
TIR 5 TDI (1f) SMASK (1f) ;
HIR 7 TDI (7f) SMASK (7f) ;
HDR 1 TDI (00) SMASK (01) ;
TDR 1 TDI (00) SMASK (01) ;
The same procedures to work with device A with step 3:
TIR 0 ;
HIR 13 TDI (1fff) SMASK (1fff) ;
HDR 2 TDI (00) SMASK (02) ;
TDR 0 ;
The same procedures to work with device C with step 3:
TIR 11 TDI (7ff) SMASK (7ff) ;
HIR 0 ;
HDR 0 ;
TDR 2 TDI (00) SMASK (02) ;
```