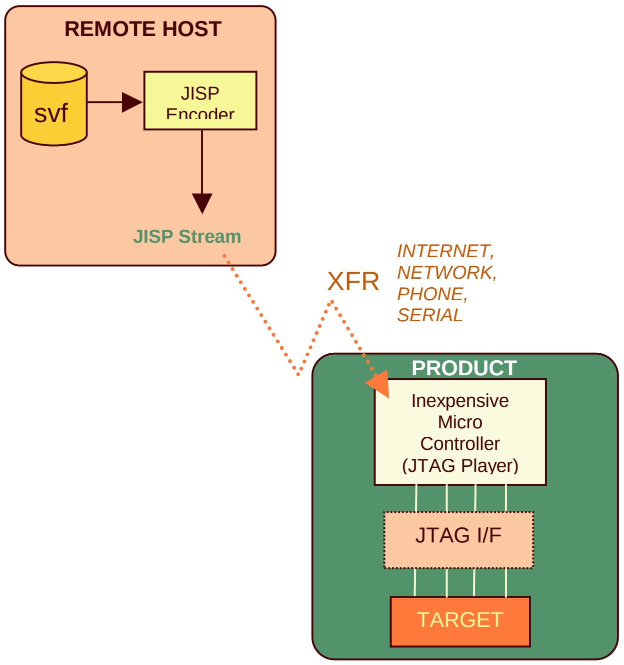
Design Advantage™
We build them for YOU.℠
Comit Systems, Inc. Newsletter
Volume 4 Number 1
Jan – Feb 2001

Simplifying ISP

By Venkat Iyer

In System Programming -- more popularly, ISP -- is gaining popularity today. More and more products are built that can be reprogrammed in system, making firmware upgrades inexpensive and convenient - over the internet, for example.

The main idea in ISP is that there is a host machine, which generates a stream, which is consumed by a player on the target product. This player causes the product to be reprogrammed. The data to be programmed may be microcode, or an FPGA bitstream.



If one is planning a range of products that need to be programmable in system, typically, one would like to build an ISP interface once and reuse it for all one's products, thereby reducing costs. This article describes a way to increase the portability of ISP across a product range, providing that the products are capable of being programmed through JTAG.

The traditional JTAG programming format, SVF, is built for consumption by relatively complex machines (like testers). However, ISP solutions for products need to be as inexpensive as possible. Many of the traditional solutions for simplifying ISP either take away the portability of JTAG, or blow up the data size, or require a significantly complex player. The scheme presented here is portable across devices and is flexible enough to accommodate special requirements of programming components.

The following are the most common causes of problems associated with SVF files:

- large memory requirement: the svf bits are all in the JTAG chain order: reverse of what is required to be shifted in. This means that the player has to store the whole data in memory and shift it out in reverse. If there is data to be compared as it is being shifted out of the TDO pin, the memory requirement is double the size of the data.
- timing requirements: svf has instructions which require the player to wait for some time. This requires the player to keep track of time and be able to pause the input stream for the duration of the delay.
- TAP controller knowledge: svf requires the player to be knowledgeable about the JTAG TAP controller. Instructions such as ENDIR, require the player to go to a specific state of the TAP controller.

If we know the system we are programming, we can simplify the player so that it can be implemented very inexpensively by moving most of this computing to *before* we generate the programming stream.

Here is the encoding of the stream. The main aims are to:

- reduce the computing power and memory requirements of the player
- limit the increase in the stream size to prevent bandwidth wastage.

The JISP Data Format is based on the following Instruction Table. An '(optional)' in the table indicates that the instruction can be left out and you will still have a functionally complete (but less efficient stream encoding).

JISP Encoding Format

1IIIIIII	7 TDI bits - no checking
01IIIOOO	3 TDI bits - checking
0011IIII	4 TDI bits - no checking
0010IIOO	2 TDI bits - checking
00011IMO	1 TMS, 1 TDI - checking
000101MM	2 TMS
000100IM	1 TMS, 1 TDI - no checking
0000111I	1 TDI no checking
0000110M	1 TMS
00001011	NOOP
00001010	VERSION <followed by 1 byte version>
0000011I	Repeat 16 TDI bits - no checking (optional)
0000010I	Repeat 16 TDI bits - check same as TDI (optional)
0000001E	Set ebit to E (optional)
00000001	START/END
00000000	Unassigned PREFIX

For each JTAG clock, the player can do one or more of these actions:

- clock in a bit on the TDI pin
- clock in a bit on the TMS pin
- clock out a bit on the TDO pin and compare

To accommodate this, the instructions allow:

- clock in data into TDI. The only data is the bits that are clocked into TDI. This is represented as I in the instructions.
- clock in data into TMS. The only data is the bits that are clocked into TMS. This is represented as M in the instructions.

Check data clocked out of TDO. This is represented as O in the instructions. If there is no O in the instruction, it means that the output from TDO is ignored.

For all instructions the bit to be shifted out first is the most significant bit in the instruction.

The NOOP instruction is a no operation. The player is required to do nothing when it gets this instruction. This is important. If the data rate to the player is fixed and the player cannot pause the input, then the stream generator inserts as many NOOPs as required to cause a delay in the programming sequence.

The VERSION is a rudimentary check that the player and the receiver are in sync, so that incompatibilities can be flagged.

The two Repeat instructions are there, in case the programming stream has lots of repeated bits. This will reduce the size of the programming stream. This adds a restriction that the player be able to clock out sixteen bits in the period that one stream byte comes in.

The ebit instructions are for the player to be able to set or reset an external bit, e.g. to toggle the reset on the target board, or to power cycle it.

If you want to add more instructions, you could use 0 as the first byte and decode another byte.

The pseudo code for a JISP player is available for download, free of charge from Comit System's website at www.comit.com. Look under 'FREE STUFF'. It can easily fit into the on-chip code memory of a simple micro-controller. The data memory requirement of the player itself is just a few bytes.

The download also includes an encoder which will use SVF based API to generate a JISP stream. ■