XILINX®

XAPP326 (v1.1) August 6, 2000

Simplified "In-System Programming" for Embedded Systems Using CoolRunner Devices

Summary

This document describes a procedure for implementing embedded In-System Programming (ISP) using a simplified binary file. This simplified binary file is produced by the XPLA™ ISP software tool and removes the requirement for detailed understanding of ISP programming algorithms. At this time, this method only supports Bulk Erase and Programming of devices; Verify, UES operations, ID Code and other ISP procedures are not supported.

Introduction

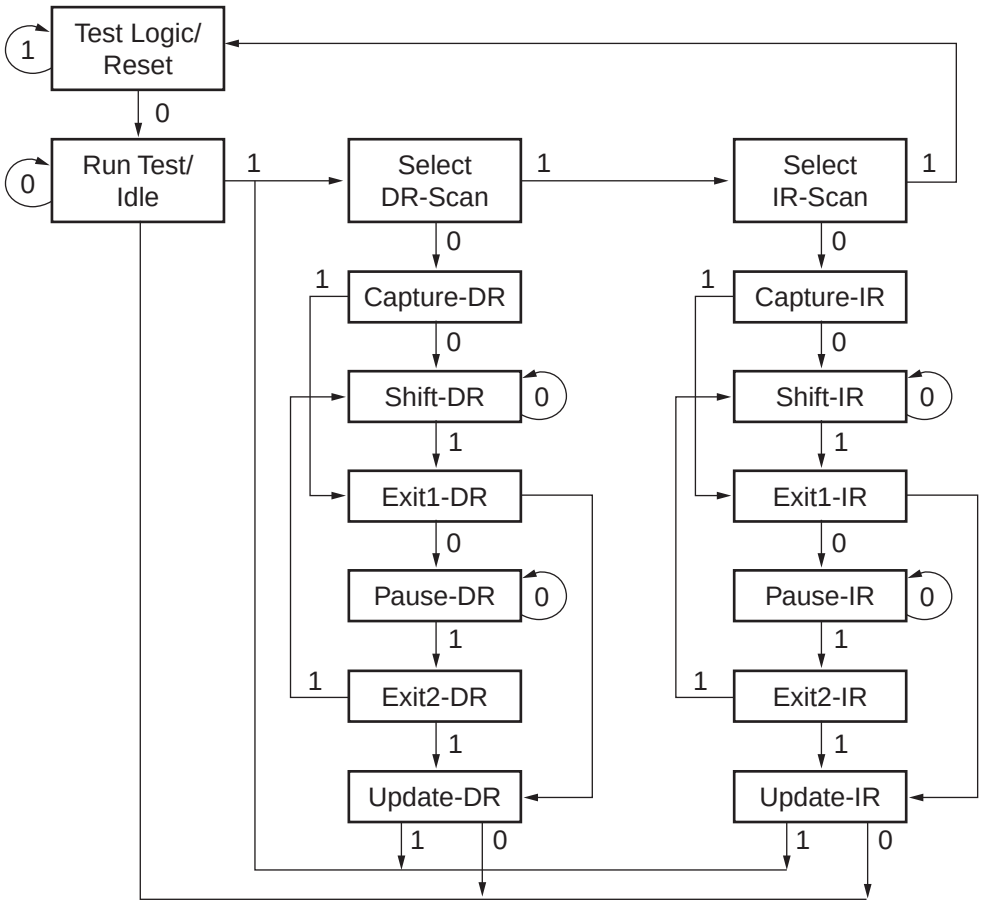
In-System Programming was initially created to remove the dependence that non-volatile programmable logic had upon stand-alone programming hardware. With the advent of ISP devices, engineers were provided with an enabling technology, providing a new method of programming which has resulted in faster time to market design cycles. The traditional benefits of ISP include rapid prototyping changes, decreased component inventory, reduced handling related component damage (such as pin non-coplanarity), programming at time of board test (ATE) and generic system modules that may be modified without the need for PCB redesign.

This ability to modify a product without hardware redesign is a benefit that additionally allows system manufacturers to update their existing (in the field) products with corrected or enhanced algorithms in a rapid and relatively benign fashion. There are different methods for implementing these changes, varying from "sneaker net" technician supervised programming, to fully automated remote programming. While deploying technicians to update and maintain hardware in the field can not be entirely eliminated, this practice is obviously costly. In some environments, such as in process control or remote monitoring, there may be risk of injury to the technician, risk of damage to the system, or issues related to "down time" associated with manual maintenance. "Smart systems" that are capable of self maintenance and automatic updates are becoming more popular; many systems now update their programmable logic automatically whenever new software is released to the end user. Other systems update via modem or telemetry, and internet reconfigurable ISP systems are poised to make an impressive entrance. The cost associated with these "auto configuration" systems is a burden of increased complexity to the system, which may mean that more engineering hours are initially required to get the product to market. Once the configuration technique is learned and implemented however, the overall cost margin of including auto configuration in systems is relatively trivial. This document describes an innovative approach to simplifying embedded programming.

Traditional Embedded ISP Programming

In order to fully appreciate the ease of use that the simplified binary format method provides, it is important for a user to understand how traditional ISP programming is accomplised. While the physical JTAG port used by ISP devices is well defined by the 1149.1 standard, the actual ISP algorithms implemented by various semiconductor manufacturers vary tremendously. Programming differences are required in order to support multiple memory technologies such as Flash, EE, or SRAM, and architecture of ISP components (storage array size and configuration, and feature set) also varies, even within same product families of a single manufacturer. ISP algorithms are noted for being complex and rather difficult to learn and implement, and the hardware and firmware required to embed ISP programming can be technically substantial.

For an example of a typical programming algorithm, refer to Table 1. This table shows the steps required to perform the programming of a CoolRunner CPLD. Figure shows the 1149.1 TAP controller state diagram which may be referenced as an aid for tracing the JTAG commands during operation. JTAG commands do not occur until the Run / Test / Idle state is entered.



Note: 1 or 0 are values of TMS at each transition.

Figure 1: **IEEE 1149.1 JTAG TAP Controller**

Table 1: **128 Macrocell ISP Programming Algorithm**

Step	Transition Conditions	TAP State	CPLD Event Description	Programmer Action
0	TMS = 1	Test-Logic/Reset	BEGIN	BEGIN
loop0	TMS = 1, TCK = ↑	Test-Logic/Reset	Ensure device in Test-Logic/Reset State	Loop five times
1	TMS = 0, TCK = ↑	Run-Test/Idle		
2	TMS = 1, TCK = ↑	Select DR-Scan		
3	TMS = 1, TCK = ↑	Select IR-Scan		
4	TMS = 0, TCK = ↑	Capture-IR		
5	TMS = 0, TCK = ↑	Shift-IR		
loop1	TMS = 0, TCK = ↑	Shift-IR	Shift-in instruction bits [0-2]	TDI = 010 (Preload)
6	TMS = 1, TCK = ↑	Exit1-IR	Shift-in instruction bit 3	TDI = 0 (Preload MSB)
7	TMS = 1, TCK = ↑	Update-IR	Load the Instruction Register	

Table 1: **128 Macrocell ISP Programming Algorithm *(Continued)***

Step	Transition Conditions	TAP State	CPLD Event Description	Programmer Action
8	TMS = 1, TCK = ↑	Select DR-Scan		
9	TMS = 0, TCK = ↑	Capture-DR		
10	TMS = 0, TCK = ↑	Shift-DR		
loop2	TMS = 0, TCK = ↑	Shift-DR	Shift-in bits [0-278]	TDI = bit 0 – bit 278 = 0
11	TMS = 1, TCK = ↑	Exit1-DR	Shift out bit 279	TDI = bit 279 = 0
12	TMS = 1, TCK = ↑	Update-DR	Load JTAG B/S update register	
13	TMS = 1, TCK = ↑	Select DR-Scan		
14	TMS = 1, TCK = ↑	Select IR-Scan		
15	TMS = 0, TCK = ↑	Capture-IR		
16	TMS = 0, TCK = ↑	Shift-IR		
loop3	TMS = 0, TCK = ↑	Shift-IR	Shift-in instruction bits [0-2]	TDI = 100 (Enable)
17	TMS = 1, TCK = ↑	Exit1-IR	Shift-in instruction bit 3	TDI = 1 (Enable MSB)
18	TMS = 1, TCK = ↑	Update-IR	Load the Instruction Register; set Enable flip-flop	
19	TMS = 1, TCK = ↑	Select DR-Scan		
20	TMS = 1, TCK = ↑	Select IR-Scan		
21	TMS = 0, TCK = ↑	Capture-IR		
22	TMS = 0, TCK = ↑	Shift-IR		
loop4	TMS = 0, TCK = ↑	Shift-IR	Shift-in instruction bits [0-2]	TDI = 110 (Program)
23	TMS = 1, TCK = ↑	Exit1-IR	Shift-in instruction bit 3	TDI = 1 (Program MSB)
24	TMS = 1, TCK = ↑	Update-IR	Load the Instruction Register	
loop5	TMS = 1, TCK = ↑	Select DR-Scan		
loop5	TMS = 0, TCK = ↑	Capture-DR		
loop5	TMS = 0, TCK = ↑	Shift-DR		
loop6	TMS = 0, TCK = ↑	Shift-DR	Shift-in Data bits [1027-0]	TDI = Data bits [1027-0]
loop7	TMS = 0, TCK = ↑	Shift-DR	Shift-in Address bits [6-1]	TDI = Address bits [6-1]
loop5	TMS = 1, TCK = ↑	Exit1-DR	Shift-in Address bit 0	TDI = Address bit 0 (LSB)
loop5	TMS = 1, TCK = ↑	Update-DR		
loop5	TMS = 0, TCK = ↑	Run-Test/Idle	Program data in EEPROM (10 ms)	Wait 10 ms
	Execute loop5 82 times to PROGRAM the entire device.			
36	TMS = 1, TCK = ↑	Select DR-Scan		
37	TMS = 1, TCK = ↑	Select IR-Scan		
38	TMS = 1, TCK = ↑	Test-Logic/Reset		
	TMS = 1	Test-Logic/Reset	DONE	DONE

Notice that instructions and data are loaded on the rising edge of TCLK. The internal state machine of the JTAG port is controlled by the TMS line; precise control of the TAP controller is required for proper configuration and use of ISP devices. The algorithm in Table 1 details the steps required for programming only. Other In-System Programming functions may be performed in a similar manner.

The above algorithm is an example of a typical programming algorithm such as the one used for XPLA3 devices. Each device will have a unique algorithm; If an engineer were to change devices and go to a lower density device, or change families, several modifications would need to be made to the algorithm in order to correctly program the new device.

Embedding the necessary control algorithm along with the configuration data can be a complex and time consuming task. Nevertheless, many designs exist that implement ISP in this fashion. Thankfully, an easier method now exists to perform embedded ISP.

Simplified Binary Programming

All programmable logic manufacturers who offer ISP products also have download and configuration software to assist in the implementation of In-System Programming. ISP software typically takes a source file (typically a JEDEC file) and converts it into a binary file, which is then accessed during the programming of the ISP device. These download tools are traditionally PC based, although workstation and ATE solutions also exist. These tools relieve end users from the requirement of intimate knowledge of the ISP algorithms, however, these PC based tools do not port easily into embedded systems.

The XPLA ISP software outputs a specialized binary file that may be used for *simplified* embedded programming. This file consists of both data and control information interleaved into two bit groups. These two bit groups form bytes in the binary file; data is arranged by least significant nibble and least significant pair. To use this binary file to perform ISP programming, the user must assign the first bit in each pair to the TMS pin of the CoolRunner device, and the second bit in each pair to the TDI pin. Once this data has been applied, a rising edge on the TCLK is required to complete the transfer. Bytes are then sequentially used in ascending order until the file is exhausted. The following line shows how bytes are used. As an example of the proper decoding order of each byte, please refer to Table 2.

Byte = Bit7(MSB), Bit6, Bit5…Bit0(LSB)

Table 2: **Byte Data Sequence Order**

Sequence Order	TMS	TDI
1	Bit 1	Bit 0
2	Bit 3	Bit 2
3	Bit 5	Bit 4
4	Bit 7	Bit 6

The following line may be data from a simplified binary file that is output from the ISP download tool:

Line # 0000:0000	Data AA 4A 0A C1 2A 10 9B 42 B4 A9 AA etc.

This data is decoded in Table 3.

Table 3: **Example of Binary File**

Nibble	TMS	TDI	TAP State / Instruction
A	1	0	At least five clocks with TMS = 1 will reset the tap controller
	1	0	
A	1	0	
	1	0	
A	1	0	
	1	0	Reset completed
4	0	0	Run Test/Idle
	0	1	Run Test/Idle
A	1	0	Select DR-Scan
	1	0	Select IR-Scan
0	0	0	Capture-IR
	0	0	Shift-IR
1	0	1	Shift-in Enable instruction = 1001
	0	0	..
C	0	0	..
	1	1	..
A	1	0	Update Instruction Register
	1	0	Select DR-Scan
2	1	0	Select IR-Scan
	0	0	Capture-IR
0	0	0	Shift-IR
	0	0	Shift-in Erase instruction = 0101
1	0	1	..
	0	0	..
B	1	1	..
	1	0	Update IR
9	0	1	Run Test/Idle: (delay pattern present here)
	1	0	Select DR Scan
2	1	0	Select IR Scan
	0	0	Capture IR
4	0	0	Shift IR
	0	1	Shift-in "Init" instruction = 1011
4	0	0	..
	0	1	..
B	1	1	..
	1	0	Update IR
9	0	1	Run Test Idle; (delay pattern present here)
	1	0	Select DR-Scan

Table 3: **Example of Binary File *(Continued)***

Nibble	TMS	TDI	TAP State / Instruction
A	1	0	Select IR-Scan
	1	0	Test-Logic/Reset
A	1	0	Test-Logic/Reset
	1	0	Test-Logic/Reset
A	1	0	Test-Logic/Reset
	1	0	Test-Logic/Reset

Note that at certain points in the programming of the device a delay is required. This is the delay necessary to erase or program the EE array. The first time the delay pattern is encountered, a 100 ms pause is required; subsequent occurences of this pattern require only a 10 ms pause. A pause of greater than the required time will not affect the device negatively. A pause of shorter than the required time may result in improper programming.

This delay is required any time the following conditions occur:

1. By monitoring the TMS/TDI binary pair and testing for a "11" followed by a "01".
2. By monitoring the TMS/TDI binary pair and testing for a "10" followed by a "01" .

Generating the Binary File

To create the simplified binary file, launch version 4.05 or later of the XPLA ISP tool. Configure the opening GUI to reflect the type of part that will be programmed, and indicate which JEDEC file is to be used. Refer to Figure 2 for an example of the configured GUI.

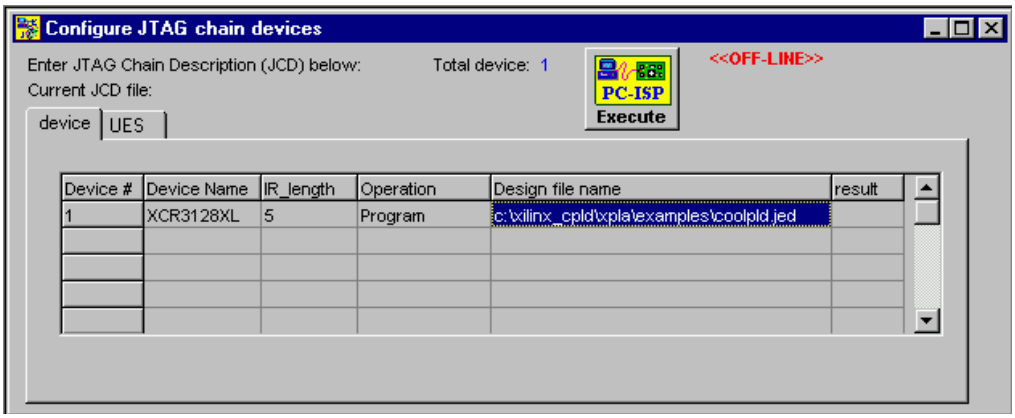


Figure 2: **Configured ISP Tool**

Once the ISP Download tool has been configured with device, operation (program only) and file, pull down the "Help" file which is located on the main GUI (not shown in Figure 2) and select the Output / Debug options. Select "Offline Programming", and acknowledge with "OK".

Pull down the "File" menu and "Save" a JCD file as a `[jcdfilename].jcd`. When the binary file is generated, it will be named `[jcdfilename].bif`, and it will reside in the same directory as the saved JCD file. Select the ATE Output menu, and highlight the "Binary JTAG File" as the output option. Click on "Execute" to generate the binary file.

Conclusion

The application of In-System Programmable devices enables engineers to prototype and debug more quickly, allows for shortened development cycles, and promotes the implementation of generic modules that can be configured as needed to satisfy system requirements. The ability to remotely program via embedded ISP adds an additional dimension to the required flexibility of today's engineering solutions, and the CoolRunner devices and software provide a simple solution to a technical issue that was once time consuming to learn and even more difficult to master.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
02/18/00	1.0	Initial Xilinx release.
08/06/00	1.1	Added TAP diagram, modified binary pattern, updated ISP screen shot.