Performing Internal In-System Programming Using Actel's ProASIC^{PLUS} Devices

Actel's ProASIC^{PLUS} FPGA family is the only FPGA family to combine the high density of an FPGA with the nonvolatility and re-programmability of the FLASH technology. Unlike SRAM-based FPGAs, the contents of the devices are not lost when the system is powered down. Therefore, there is no external ROM needed to reload the device at power up. The contents remain in the device until there is a reason to change them. The FLASH technology makes it possible for a system containing a ProASIC^{PLUS} device to reprogram the device itself, with no outside influence (external programmer) except for the delivery of a new data file.

The ability to reprogram a device that has already been mounted onto a system board is referred to as In-System Programming (ISP). There are two types of ISP to accommodate different system requirements:

- External ISP: The device is mounted onto the system, but an outside programmer is used to implement the programming. Although the device is in-system, the programming control comes from outside the system, thus the name, External ISP. Actel's Flash Pro device programmer can be used for this purpose.
- Internal ISP: There is no outside programmer used to configure the FPGA. All programming control comes from the system itself, using the native processor, operating system, and memory. The data used for configuring the device could be resident in ROM (as in the case of pre-designed configuration alternatives), arrive via some other media (such as a floppy disk), or come through a communication channel like a network connection. The only external influence is the data itself.

This application note focuses solely on Internal ISP. See Actel's web site for future application notes concerning information on using External ISP with ProASIC and ProASIC^{PLUS} devices.

Note: One additional concept, too complicated to summarize in this application note, is known as In-Circuit Reconfiguration (ICR). ICR refers to the real-time changing of all or part of the contents of a device in order to implement on-the-fly context switches or function changes while the system continues to function normally. This capability requires consideration far more complex than ISP and will not be addressed in this application note.

Overview

Internal ISP uses its own system for reconfiguration, and the details of the implementation depend on the nature of the system. Details vary according to the processor, operating system, bus structure, and memory architecture. This application note does not discuss all of the possible variations, but instead focuses on important considerations for successful implementation of internal ISP.

There are three main aspects to Internal ISP implementation:

- The hardware structure
- The software program that will execute the programming process
- The data file that contains the programming instructions

Internal ISP is accomplished by designing access from the system processor to the IEEE 1149.1 (JTAG) port of the ProASIC^{PLUS} device and then running a program that will take a data file and configure the FPGA. The data format used for programming ProASIC^{PLUS} devices is a JEDEC standard known as the Standard Test And Programming Language (STAPL) format. For more information on JTAG go to www.ieee.org; the STAPL standard can be obtained through JEDEC at www.jedec.org.

The program that interprets the contents of the STAPL file is called a STAPL Player. The STAPL Player reads the STAPL file and executes the programming instructions contained within the file. Because all programming details are in the STAPL file, the STAPL Player is completely device-independent. In other words, the system does not need to implement any programming algorithm details; the STAPL file provides all of the details.

Because ProASIC^{PLUS} devices are programmed through a JTAG port, it is possible to program multiple devices on a JTAG chain. Some considerations for chain programming are addressed in "Chain Programming" on page 6.

A high-level drawing of the Internal ISP process is shown in Figure 1 on page 2.

The hardware and software considerations will be addressed separately in the following sections.

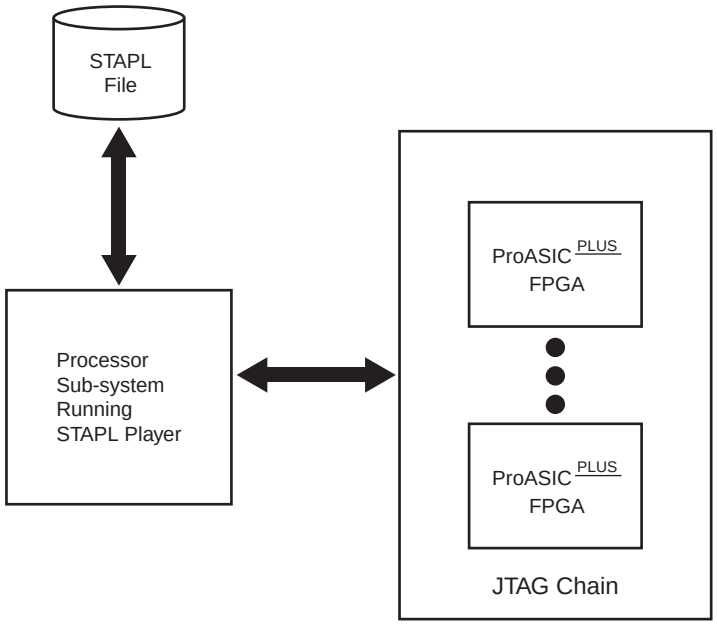


Figure 1 • *Internal ISP Elements*

Hardware Considerations

Internal ISP requires the following components:

- Processor for running the STAPL Player
- Nonvolatile memory for storing the STAPL Player
- RAM for storing and manipulating the STAPL file
- Access to the JTAG port of the FPGA or to the JTAG chain
- DC voltage generator
- Optional free-running oscillator

Figure 2 on page 3 illustrates an example of how a system can be interconnected. Access to the JTAG port is shown via an interface block. This is one way of mapping the JTAG port into the memory map. If the processor has general I/Os, they could also be connected directly to the JTAG signals.

Note the optional inclusion of an external JTAG port would be required if the JTAG port was also being used for system test and diagnostics. In this diagram, the interface block decides whether the control of the JTAG chain should come from the processor or from the external port.

Although not shown in Figure 2 on page 3, the JTAG standard allows for an optional TRST pin. This is not shown, but can be used. The usage of this pin would be accounted for in the low-level API function that implements the reset function. Refer to "Software Considerations" on page 4 for information on API.

FPGA Interaction with the System

A very important aspect of Internal ISP is the behavior of the FPGA before, during, and after programming. The signal connections shown in Figure 2 on page 3 are only those required for programming. Not shown are all of the connections between the FPGAs and other system components. These other connections constitute the normal function of the FPGAs when they are not being programmed. When other FPGAs not are programmed correctly, there is the possibility of system damage due to incorrect operation or signal contention. During programming itself, the FPGAs have no program, and therefore, cannot be expected to operate as they would during normal system operation. The ProASIC PLUS devices have been designed with this in mind and therefore, have the following characteristics:

- As soon as the programming sequence is started, all user I/Os are put into a tristate condition and are weakly pulled to the V_{CC} level used for the I/Os. (V_{DDP})
- As soon as the programming sequence has successfully been completed, the device will wake up into the state determined by the contents of the program, and should therefore be consistent with desired system behavior.
- If the programming sequence is not completed successfully, the I/Os will remain in a tristate condition, eliminating the risk of unexpected signals.

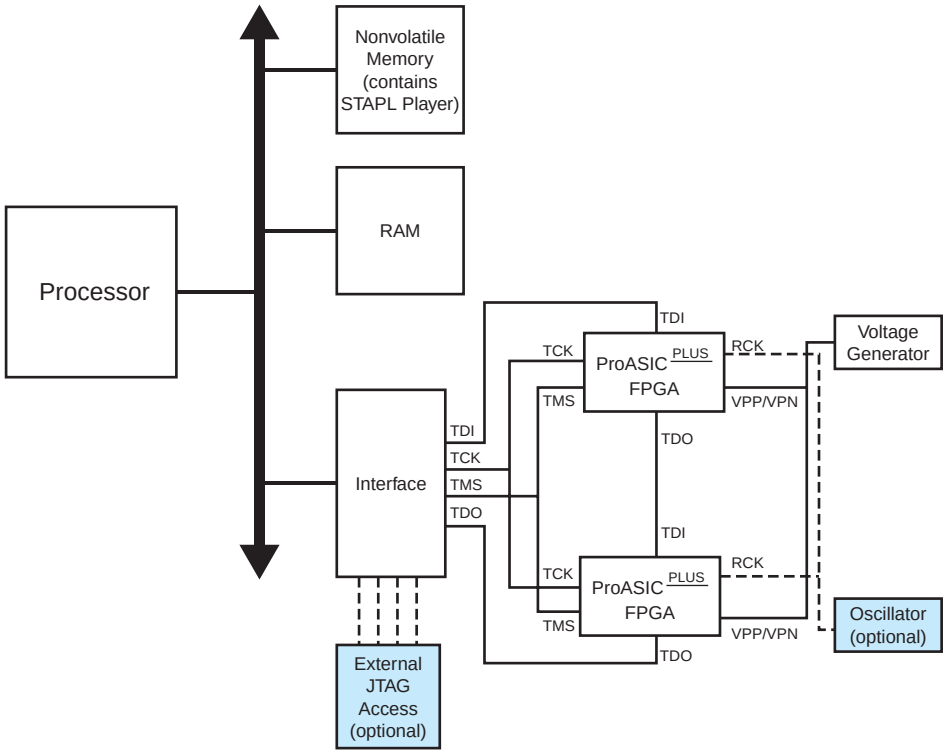


Figure 2 • *Example of System Configuration*

Note that these features only apply to the correct programming of the data in a STAPL file. If the STAPL file itself is incorrect, there are several possible outcomes:

- If there was corruption resulting in an illegal file, then the STAPL Player will reject the file, programming will not complete successfully, and the device will remain in a tristate condition.
- If there was corruption resulting in a legal file but an incorrect programming sequence, then programming will not complete successfully, and the device will remain in a tristate condition.
- If the STAPL file is legal and contains a valid programming sequence but the data is incorrect, then the device will successfully be programmed with incorrect data. In reality, this will not occur due to corruption, but due to logic errors in the design. The implications of this are the same as the implications of any logic error in an FPGA; the system will not work as expected. Thorough validation of the design is always important before configuring an FPGA.

Power Supplies

The ProASIC PLUS devices require four power supplies:

- V_{DD} to power the core (used in normal operation)
- V_{DDP} to power the I/Os (used in normal operation)
- V_{PP} as the positive voltage for programming
- V_{PN} as the negative voltage for programming.

V_{PP} and V_{PN} are generated by the Voltage Generator block, consisting of DC-DC converters and some passive elements to generate the appropriate voltage levels. "Appendix A: Suggested Voltage Generator Circuit" on page 9 provides a diagram of the required circuits and recommended components for the Voltage Generator block. Table 1 provides the voltage levels required during normal operation; Table 2 on page 4 provides the voltage levels required during programming. The DC current that must be available on the voltages is shown in Table 3 on page 4.

Table 1 • *Voltages during Normal Operation*

Power Supply	Voltage Range (V)
V_{DD}	2.5
V_{DDP}	2.3 - 2.7 or 3.0 - 3.6
V_{PP}	0 - 16.5 or floating
V_{PN}	-13.8 - 0 or floating

Table 2 • Voltages during programming

Power Supply	Voltage Range (V)
V_{DD}	2.3 - 2.7
V_{DDP}	2.3 - 2.7 or 3.0 - 3.6
V_{PP}	15.9 - 16.5
V_{PN}	-13.8 - -13.4

Table 3 • Current Requirements

Power Supply	Voltage (V)	Maximum Current (mA)
V_{DD}	2.7	20
V_{DDP}	3.6	20
V_{PP}	16.5	35
V_{PN}	-13.8	15*

Note: *Absolute value; current is negative.*

Required Bypass Capacitors:

Bypass capacitors are required for the V_{PP} and V_{PN} pads. The capacitors should be placed directly next to the device to be effective. To filter low frequency noise, use a 4.7uF (low ESR, <1 Ω, tantalum, 25V or greater rating) capacitor. To filter high frequency noise, use a 0.01µF to 0.1µF ceramic capacitor with a 25V or greater rating. The smaller high frequency capacitor should be placed closer to the device pins than the larger low frequency capacitor. The capacitors should be located as close to the device pins as possible (within 2.5cm, if possible). The same dual capacitor circuit should be used on both the V_{PP} and V_{PN} pins (Figure 3).

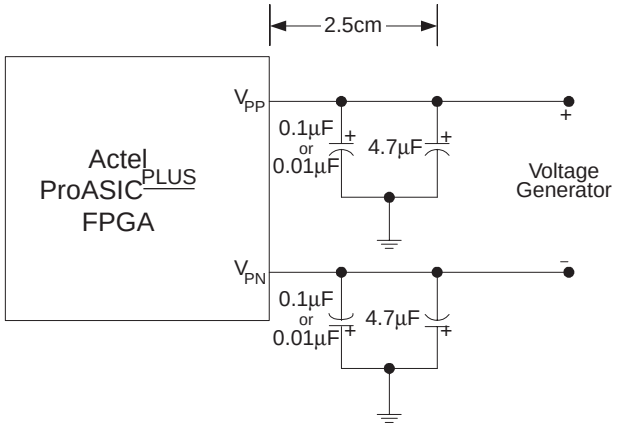


Figure 3 • Bypass Capacitors

Power-Up Sequences

For ProASIC PLUS ES devices, a special power-up sequence must be observed. The four power supplies must be brought up in the following sequence:

V_{DD}, V_{DDP}, V_{PN}, V_{PP}

Interruptability

There are several critical portions of the programming sequence that should not be interrupted. Timing during those portions is very important, and an uninterruptible clock must be provided. If the system is set up such that the processor can be guaranteed not to be interrupted while programming, then the JTAG pins are sufficient to implement the programming.

If it is possible that the sequence could be interrupted, then the JTAG clock should not be used to time the programming. Instead, there is an RCK signal to which an oscillator circuit can be attached. This signal will then provide the needed clock for timing inside the chip.

In order for RCK to be used, you must edit the STAPL file. Open it in an ASCII editor and find the line.

BOOLEAN USE_RCK=0

Change that line to

BOOLEAN USE_RCK=1

In addition, you should enter the frequency of RCK into the STAPL file using the following line:

INTEGER freq=X

where X is the frequency in megahertz.

Clock Considerations

Because TCK is a bussed signal in JTAG, it could have high fanout. The quality of the clock network layout impacts the quality of the clock signal. As with any other clock network, care should be taken during board design to ensure that the clock signals are clean, as clean clock signals enable the JTAG engine to operate reliably.

Software Considerations

The STAPL Player

The STAPL Player consists of two basic portions: the high-level C code that executes the main program, and a low-level API that is called by the higher-level program. The high-level portion is intended to be portable across systems, and then API allows adaptation into a particular system.

Because of this feature, the majority of the work to develop a STAPL Player can be used across all systems. Such a STAPL Player has been written and is available in C code on Actel's web site at
http://www.actel.com/products/proasicplus/info.html.
While every attempt has been made to make the code generic enough to use across many kinds of systems, the designer should inspect the code to ensure that there is nothing incompatible with any special situations that might exist in his system.

The STAPL Player Interface

Actel's STAPL Player can be run in command-line mode in a DOS or Windows system. The command has the form

```
STAPL.exe [optional switches]
```

There are several switches that might be used in an embedded context (Table 4).

Table 4 • STAPL Player Command-line Switches

Optional Switches	Description
-h	Show help message
-v	Show verbose messages
-a<action>	Specify action name (with STAPL file)
-d<var=val>	Initialize variable to specified value

The -a switch indicates to the Player that a specific action, like programming or verification, is to be undertaken. Actions are defined in the STAPL file.

The -d switch allows pre-initialization of variables in the STAPL file. A thorough understanding of the variables defined in the STAPL file and their usage should be obtained before using this switch.

The STAPL Player API

The API is a series of low-level functions that interact with the operating system. The API used with Actel's STAPL Player is shown below. Each user must write the low-level routines that will provide these services to the STAPL Player.

There are two categories of function: those that manipulate the JTAG port, and those that provide operating system services (Table 5 and Table 6).

After the API has been written, it is linked with the C-base STAPL Player, and the final executable can then be run in the system.

Table 5 • High Level JTAG API Functions

Function	Description
JtagReset	Send TMS=1,1,1,1,1 to go to Test Logic Reset State, or send TRST=1 if TRST is implemented
JtagIdle	Traverse the state machine from Pause to Idle
JtagIrEnter	Traverse the state machine from Idle or Reset to Shift-IR or Pause-IR
JtagDrEnter	Traverse the state machine from Idle or Reset to Shift-DR or Pause-DR
JtageShift	Shift <bits> data bits starting at bit 0 of data[0]
JtagRead	Return data output on TDO from previous shift sequence
JtagWait	Pause until <ticks> TCK's have been output

Table 6 • Low Level JTAG and OS API Functions

Function	Description
stp_getc	Returns a single character from the STAPL files
stp_seek	Moves the pointer within the STAPL file
stp_jtag_io	Low level I/O JTAG function
stp_message	Returns a message to the STAPL player
stp_delay	Generates a delay loop for n seconds
stp_malloc	Allocates memory
stp_free	Frees up memory

Exception Handling

There are two kinds of errors that must be addressed. The STAPL file creates the first type of error. After the STAPL file has been executed, it returns an exit code that indicates the result of the programming. Actel's STAPL file supports the following exit codes (Table 7).

Table 7 • STAPL File Exit Codes

Exit code	Description
0	Success
5	Entering ISP failure
6	Unrecognized device ID
7	Unsupported device version
8	Erase failure
11	Verify failure
12	Read failure
90	Unexpected RCK detected
91	Calibration data parity error

Unlike the first type of error, the STAPL Player finds the second type of error. For example, if the STAPL Player reads a STAPL file and runs across a syntax error, it will generate an error. Table 8 on page 6 lists the error codes that the STAPL Player is capable of returning.

Table 8 • STAPL Player Exit Codes

Exit code	Description
0	Success
1	Out of memory
2	I/O error
3	Syntax error
4	Unexpected end
5	Undefined symbol
6	Redefined symbol
7	Integer overflow
8	Divide by zero
9	CRC error
10	Internal error
11	Bounds error
12	Type mismatch
13	Assign to const
14	Next unexpected
15	Pop unexpected
16	Return unexpected
17	Illegal symbol
18	Vector map failed
19	User abort
20	Stack overflow
21	Illegal opcode
22	Phase error
23	Scope error
24	Action not found

Chain Programming

Chain programming is defined as programming several devices that are on the same JTAG chain. There are several device configurations for such a chain, and different ways to approach programming:

- ProASIC^PLUS devices ONLY, programmed one at a time
- ProASIC^PLUS devices ONLY, programmed concurrently
- ProASIC^PLUS devices along with other non-programmable JTAG devices
- ProASIC^PLUS devices along with other programmable (and potentially non-programmable) JTAG devices

The Actel STAPL file currently supports the first type of chain programming. Concurrent programming will be supported in a future release. If a chain has multiple programmable devices, they can successfully be programmed one at a time by bypassing the devices not being programmed.

In the case where other non-programmable devices are in the chain, they must be bypassed and then the programmable device(s) can be programmed. If there are multiple ProASIC^PLUS devices, then they should be programmed individually.

It is not possible to concurrently program devices from different vendors. If non-ProASIC^PLUS programmable devices coexist on the same chain, then the non-ProASIC^PLUS devices should be treated just like non-programmable devices when programming the ProASIC^PLUS devices.

All JTAG-compliant devices have a bypass mode that bypasses the data register. Any device in bypass mode will have a data register length of one. If an entire chain is in bypass mode, then the apparent length of the entire chain's data register is equal to the number of devices in the chain. When programming a single device, bypass all other devices. In this case, the complete length of the chain data register will be the length of the programmable device's data register plus one bit (corresponding to bypass mode) for other devices in the chain. This means that the bit-stream intended for the data register of the programmable device must be padded on the front and back end by as many bits as there are devices before and after the programmable device. (Figure 4 on page 7).

A device can be placed into bypass mode by loading an instruction of all ones. This will vary from device to device only because different devices have different instruction registers lengths. Therefore, it is important to know the length of the instruction register for each device in the JTAG chain. Figure 5 on page 8 illustrates this concept.

In order to account for more than one device on a chain, there are some instructions in the STAPL language that allow for the necessary padding of the bit-streams. The instructions can be added to the STAPL file, as shown in Table 9 on page 8. Note when configuring a bit-stream, the first bits end up in the last device of the chain, so the padding at the front end of the bit-stream is for the devices at the end of the chain.

These instructions must be added to the Procedure Initialize section of the STAPL file.

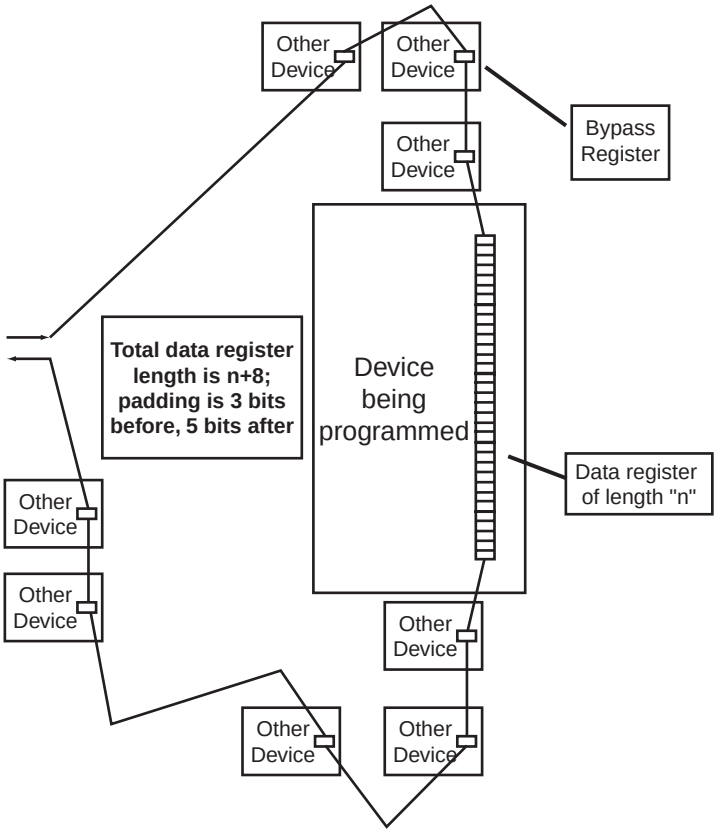


Figure 4 • *Illustration of Data Register Padding*

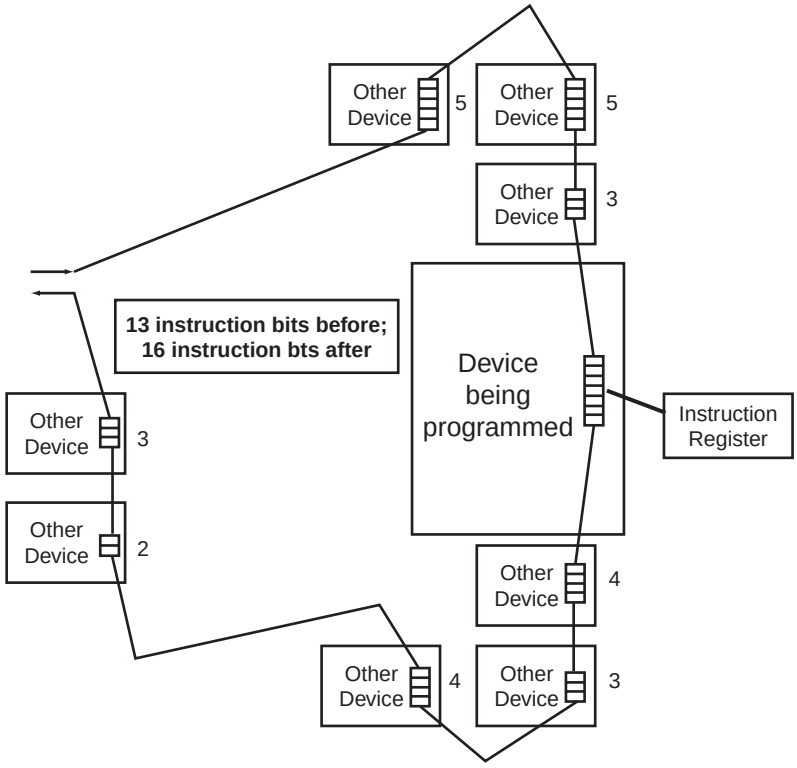


Figure 5 • *Putting Devices into Bypass Mode*

Table 9 • *Register-padding instructions*

Instruction	Purpose
PREIR n	Used to put the devices after the programmable device into Bypass mode; n is the number of instruction bits after the programmable device
POSTIR n	Used to put the devices before the programmable device into Bypass mode; n is the number of instruction bits before the programmable device
PREDR n	Used to pad the data by the number of bypassed devices after the programmable device; n is the number of devices after the programmable device
POSTDR n	Used to pad the data by the number of bypassed devices before the programmable device; n is the number of devices before the programmable device

Summary

Internal ISP can be performed on the ProASIC^PLUS family. The user must make the JTAG chain accessible to the system processor, and memory must be available both to house the STAPL Player and to manipulate the STAPL file. A simple, inexpensive voltage generator circuit is used to provide the required voltages. The STAPL Player is created by linking the high-level STAPL Player code available from Actel with user generated low-level API routines. In sum, a well-designed system including these elements can provide robust and reliable reprogramming for an unlimited number of ProASIC^PLUS devices in a JTAG chain.

Appendix A: Suggested Voltage Generator Circuit

A circuit for generating the required voltages for Internal ISP is shown in Figure 6 and Figure 7. See Table 10 and Table 11 on page 10 for a list of Building Materials required for building circuits.

There are two Linear Tech voltage converters in the circuit. Application notes for each of them can be found at:

www.linear-tech.com/pdf/1930f.pdf for the positive converter

www.linear-tech.com/pdf/1931f.pdf for the negative converter

Schematics

Figure 6 • *Boost Schematic*

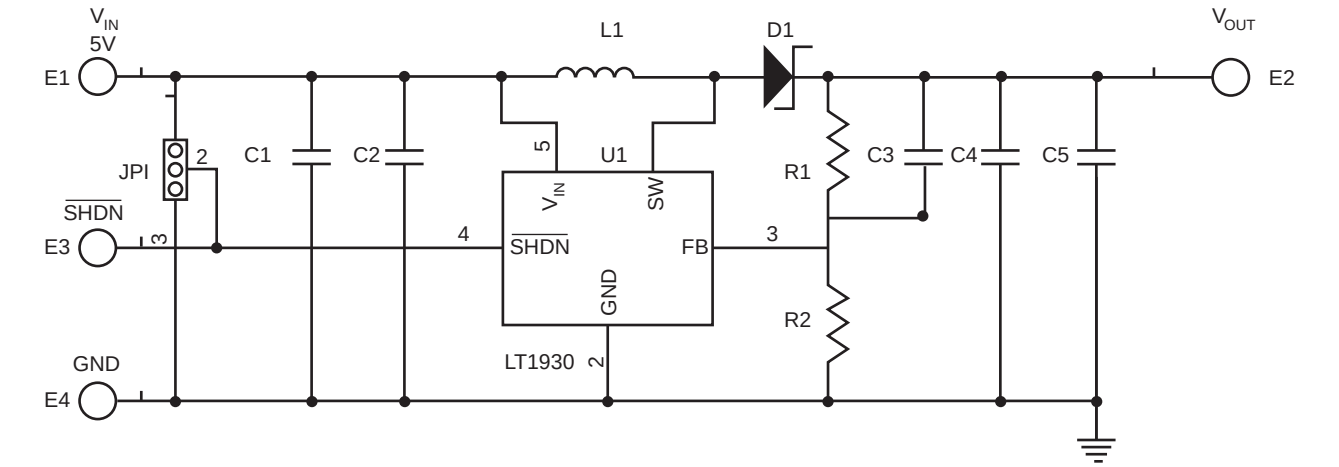
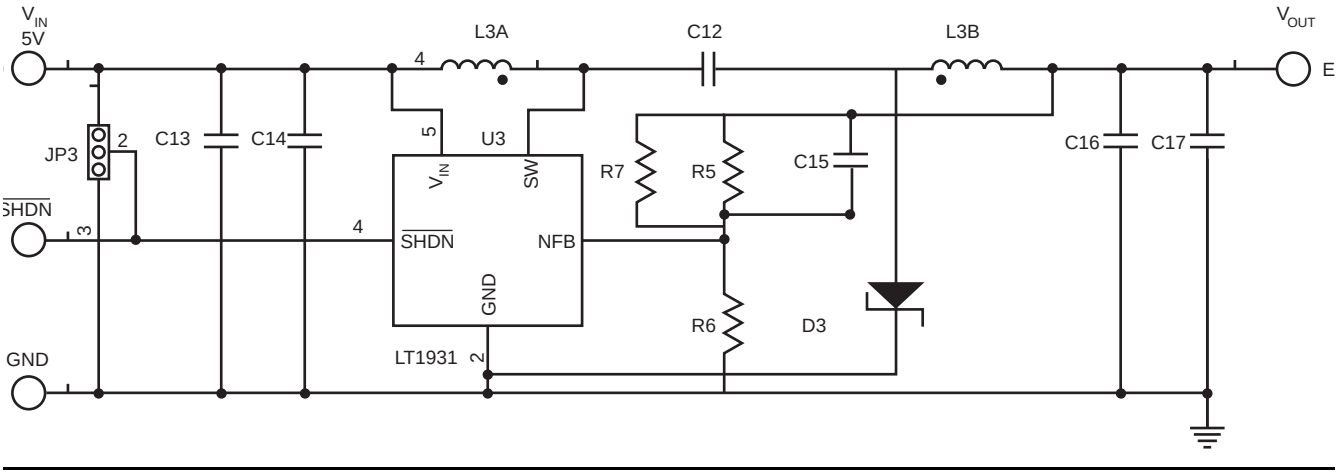


Figure 7 • *Inverter Schematic*



Bill Of Materials

Table 10 • Boost Circuit (positive converter)

Item	QTY	Reference	Part Description
1	2	C1, C5	CAP, X5R, 1µF, 16V, 0805
2	1	C2	CAP, X5R, 2.2µF, 16V, 1206
3	1	C3	CAP, NPO, 10pF, 25V, 10%, 0402
4	1	C4	CAP, X5R, 4.7µF, 16V, 1206
5	1	D1	DIO, SCHOTTKY, 30V, 0.5A
6	1	L1	Inductor, 10µH, 20%
7	1	R1	RES, CHIP, 156K, 1/16W, 1%, 0402
8	1	R2	RES, CHIP, 13.3K, 1/16W, 1%, 0402
9	1	U1	I.C., LINEAR, LT1930ES5#25117

Table 11 • Inverter Circuit (negative converter)

Item	QTY	Reference	Part Description
1	2	C13, C17	CAP, X5R, 1µF, 16V, 0805
2	2	C14, C16	CAP, X5R, 4.7µF, 16V, 1206
3	1	C12	CAP, X5R, 1µF, 25V, 1206
4	1	D3	DIO, SCHOTTKY, 30V, 0.5A
5	2	L3A, L3B	Inductor, 10µH, 30%
6	1	R5	RES, CHIP, 10K, 1/16W, 0.1%, 0402
7	1	R6	RES, CHIP, 1K, 1/16W, 0.1%, 0402
8	1	R7	RES, CHIP, 402K, 1/16W, 1%, 0402
9	1	U3	I.C., LINEAR, LT1931ES5#25118

http://www.actel.com

Actel Europe Ltd.
Maxfli Court, Riverside Way
Camberley, Surrey GU15 3YL
United Kingdom
Tel: +44 (0)1276 401450
Fax: +44 (0)1276 401590

Actel Corporation
955 East Arques Avenue
Sunnyvale, California 94086
USA
Tel: (408) 739-1010
Fax: (408) 739-1540

Actel Asia-Pacific
EXOS Ebisu Bldg. 4F
1-24-14 Ebisu Shibuya-ku
Tokyo 150 Japan
Tel: +81-(0)3-3445-7671
Fax: +81-(0)3-3445-7668

5192691-4/10.02