Designing With FLASH370i for PC Cable Programming

Introduction

This application note presents how to design with the Cypress In System Reprogrammable (ISR™) family of complex PLDs, the FLASH370i family, for programming from a PC with the ISR programming cable. The main issues addressed are those related to programming and reprogramming the devices in-system (i.e., while they are soldered onto a printed circuit board). Primary among these is interfacing to the pins used to program the devices, both when those pins are used only for programming and when they are used both for programming and for I/O in normal operating mode.

To address this issue, we categorize designs into three types: designs using devices with single-function pins; designs using devices with dual-function pins used in single-function mode; and designs using devices with dual-function pins used in dual-function mode. We will cover all three cases in this application note. The three cases are explained further below.

Single-Function Pins / Single-Function Mode: Some of the FLASH370i devices have pinout/package combinations such that the pins used for programming are single-function only; i.e., they are used as programming pins only. When the device is operating (i.e., not being programmed) they are not in use; they are just extra pins.

Dual-Function Pins / Dual-Function Mode: The rest of the devices in the FLASH370i family have pinout/package combinations such that the pins used for programming have dual functionality. They are used as programming pins when the device is being programmed, and they are used as I/Os when the device is in normal operating mode. To use both of these functions, you must design interface logic to isolate programming signals from other devices on the board.

Dual-Function Pins / Single-Function Mode: Alternatively, the designer can decide to use these dual-function pins as programming pins only and not connect them as I/Os for normal operation. The design is simpler in this case as no special interface logic is required.

All FLASH370i devices can also be cascaded into a single chain for programming purposes so that one cable and one connector can be used to program all of the FLASH370i devices on a board. We will show a simple example of this in this application note, but this topic is covered in depth in an application note called "Cascading FLASH370i Devices."

In addition to the topic of interfacing to the programming pins, several other topics are covered in this application note. These include:

- handling the 12V signal on the board;
- the state of the FLASH370i devices' I/Os at power-up (and why you need to know about this);
- the state of the ISR programming cable's pins when not programming and the state of the FLASH370i's ISR programming pins when no cable is attached;
- and the structure of the ISR software programming configuration file.

Throughout this application note, assume that the ISR devices are programmed in system by means of the ISR cable that connects the board to a PC. This ISR programming cable is provided by Cypress, and the details of the cable and the connector on the board to which it interfaces are explained in detail in this application note. There are other ways to program or reprogram the parts in-system as well, and many of the topics covered and solutions shown in this application note also apply to those methods.

The ISR Programming Cable

The pins on a FLASH370i device used for programming are: ISRVPP, SDI, SDO, SMODE, and SCLK. Their names and functions are defined below.

ISRVPP In System Reprogramming Voltage

During programming, this pin supplies the device with the voltage needed to program it, which is 12.0V ± 0.6V. During normal operation this pin must be between 0V and 5V.

SDI Serial Data Input

During programming, this pin is the serial input to the device.

SDO Serial Data Output

During programming, this pin is the serial output from the device.

SCLK Serial Clock

During programming, this pin is the clock input. SDI and SMODE are sampled on the rising edge of SCLK, while SDO changes following the falling edge of SCLK.

SMODE Serial Mode Control

During programming, this is the mode select control input that directs the TAP (test access port) controller state machine contained within the ISR interface.

The FLASH370i devices are programmed using a PC as shown in *Figure 1*: the ISR programming cable connects the parallel port of the PC to a cable header on the board on which the FLASH370i devices are soldered. The header on the board connects to the traces that go to the ISRVPP, SDI, SDO, SCLK, and SMODE pins on the FLASH370i devices themselves. The ISR software runs on the PC and drives these pins on the board, through the cable, header, and traces, to program the devices with the appropriate JEDEC files. The cable has a DC/DC converter

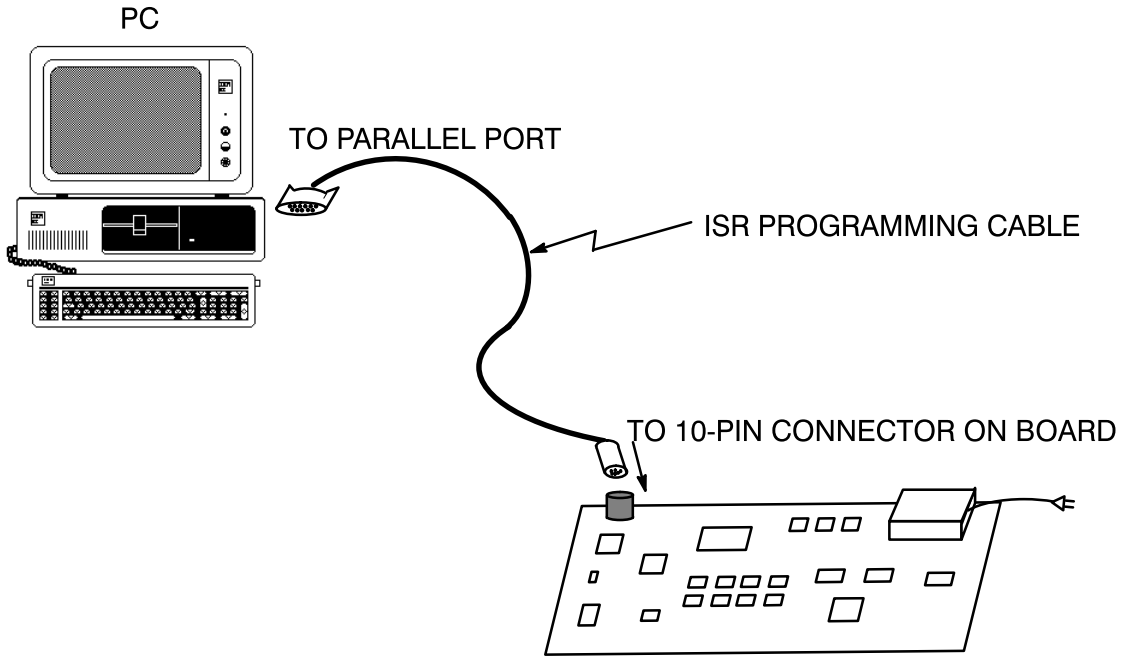


Figure 1. ISR Programming Cable

built into it that receives 5V from the board and supplies the 12V programming voltage to the ISRVPP pin.

A 10-pin, 2 x 5, boxed header connector is used on the board for the ISR programming cable to plug into. This boxed header connector has a small opening in the box on one side (the key) that allows the ISR programming cable to be plugged in one way only. The pins are on 0.100" centers. The length of each pin is 0.230", and the cross-section is 0.025" x 0.025". This boxed header connector is available as a straight-pin connector and as a right-angle connector.

Additionally, an open header can be used. Part numbers for two compatible connectors are:

DIGI−KEY part # S2012−05−ND
(straight-pin connector)

DIGI−KEY part # S2112−05−ND
(right-angle connector)

The ISR programming cable provided by Cypress has a female end which plugs into these connectors. The position of the signal pins on the connector is shown in *Figure 2*.

To program a FLASH370i device using the ISR programming cable described here, all you need to do is route the ISRVPP, SDI, SDO, SCLK, and SMODE pins from the cable connector to the ISRVPP, SDI, SDO, SCLK, and SMODE pins of the FLASH370i device, respectively.

In addition to these programming pins, there is an additional signal available from the cable called ISR*. The purpose of this signal is to allow the user to know the state of the signals coming from the cable. If ISR* is a logic '0', it indicates that ISRVPP is 12V and one or more FLASH370i devices on the board are being programmed; if ISR* is a logic '1', it indicates ISRVPP is not 12V and no FLASH370i devices on the board are being programmed. This is particularly useful in the "Dual-Function Pins/ Dual-Function Mode" designs mentioned in the introduction and explained in detail later in this application note. The logic '0' and logic '1' levels on ISR* are 0V and 5V, respectively. The logic '0' level is driven actively while the logic '1' level is not driven but pulled up using a 20 kΩ pull-up resistor to the VCC pin of the ISR connector.

SMODE	SCLK	SDI	NC	GND
GND	ISRVPP	ISR*	VCC	SDO

Figure 2. Layout of Connector for Cable on Board, Top View

There are three other connection points on the cable and cable header: VCC, GND, and NC. VCC is the the pin through which the VCC plane on the board containing the FLASH370i devices supplies 5V to the DC/DC converter in the ISR cable. This is necessary for the ISR programming cable to be able to generate the 12V voltage level on ISRVPP needed to program the FLASH370i devices. GND provides a common ground reference between the board and the ISR programming cable. NC is a no connect that is not used.

Designs That Use Devices With Single-Function Programming Pins

Single-function programming pins refers to the case where the pins used for programming the device are only dedicated to that function and are extras—not in use—when the device is in normal operating mode. Two members of the FLASH370i family currently have this feature, the 64-macrocell CY7C373i in the 100-pin TQFP package and the 128-macrocell CY7C374i in the 100-pin TQFP package. These two devices, which have identical pinout, have 64 I/O pins and 6 input-only pins for a total of 70 I/Os. Therefore, in the 100-pin package, there are still plenty of pins to accommodate the five ISR pins on separate pins and still have ample power and ground connections. As the pinout diagram of these two devices in *Figure 3* shows, pins 88, 75, 50, 1, and 26 are, ISRVPP, SDI, SDO, SCLK, and SMODE, respectively.

Designing with these devices is the easiest of the three cases. You simply connect these ISR pins to

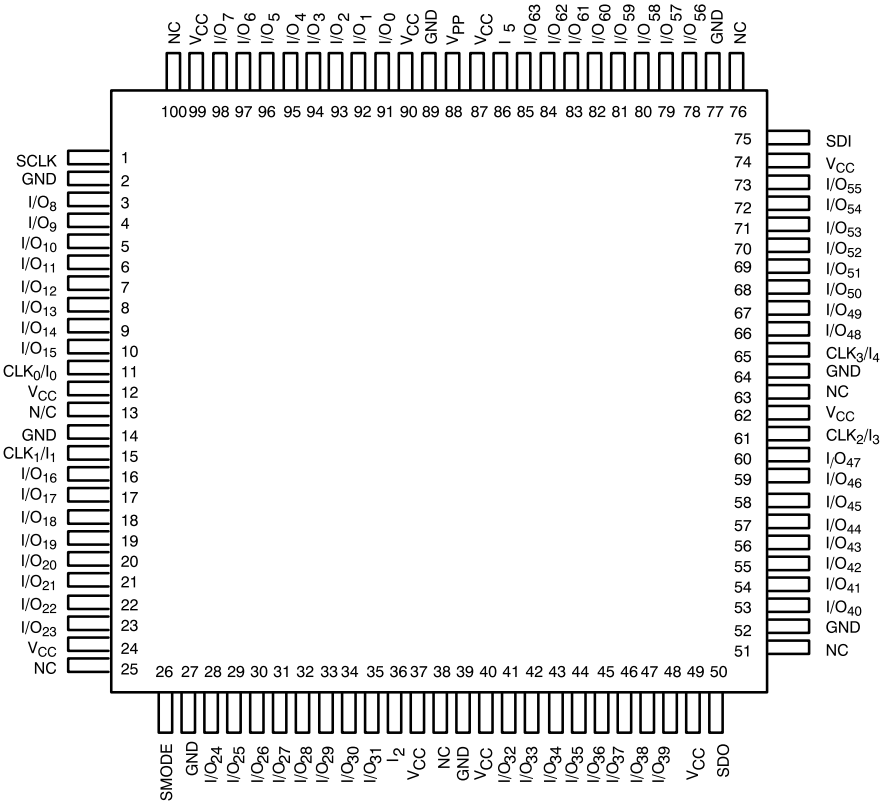


Figure 3. Pinout of CY7C373/4i in 100-Pin TQFP

the corresponding pins on the ISR programming cable connector, and you now have access to in system reprogramming.

There are other issues related to this, such as the best way to route the 12V signal to the ISRVPP pin, the state of the ISR programming pins when the cable is disconnected, and the format of the ISR software configuration file. These issues apply to all three types of designs (single-function pins, dual-function pins used in single-function mode, and dual-function pins used in dual-function mode), and will be treated in a later section.

Designs That Use Devices With Dual-Function Programming Pins

Dual-function programming pins refers to the case where the ISR pins on a FLASH370i device are used for programming the device when it is in programming mode and are used as normal I/Os in normal operating mode. The devices that have this characteristic are the CY7C371i and CY7C372i in 44-pin packages, the CY7C373i and CY7C374i in 84-pin packages, and the CY7C375i in 160-pin packages—in other words, all of the devices in the family except the two mentioned in the single-function section above.

For example, look at the CY7C373i and CY7C374i 84-pin PLCC pinout in *Figure 4*. These are similar to the devices described in the single-function section above, but with 64 and 128 macrocells respectively, and with 64 I/O pins. In this case, the programming functions share pins with normal I/O functions. Pin 72 is SDI in programming mode and I/O54 in normal operating mode; pin 51 is SDO in programming mode and I/O38 in normal operating mode; pin 14 is SCLK in programming mode and I/O10 in normal operating mode; and pin 35 is SMODE in programming mode and I/O26 in normal operating mode. Pin 83 is ISRVPP and differentiates the function of these other four ISR interface pins.

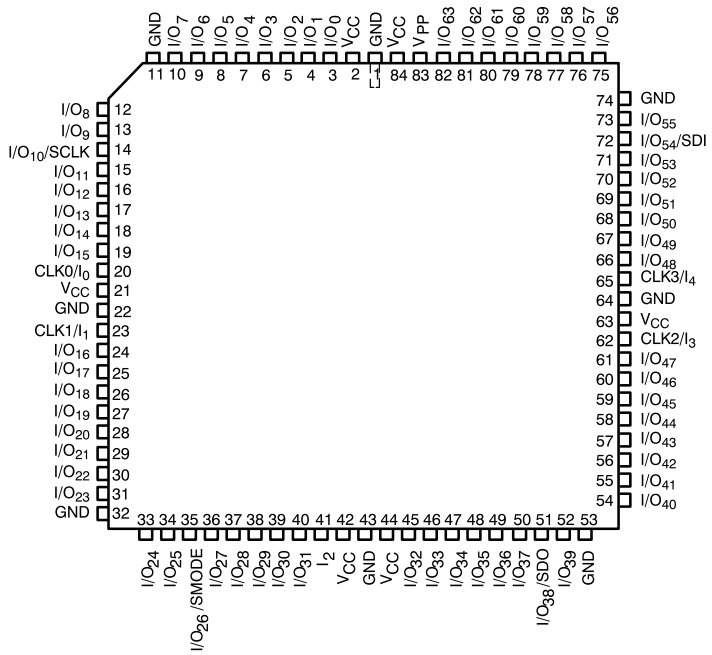


Figure 4. Pinout of CY7C373/4i in 84-Pin PLCC

There are two ways to design with devices that have dual-function programming pins. First, you could use the dual-function pins as single function pins. That is, you could decide to use only the programming function of the pins and not use those pins as I/Os in your design. The other way to use them, of course, is to use them as true dual-function pins, where you use them both as programming pins in programming mode and as I/Os in normal operating mode.

Devices With Dual-Function Programming Pins Used in Single-Function Mode

To use the FLASH370i devices in this way, with the dual-function pins used as programming pins only, you need the total number of I/Os used in your design to be equal to or less than (n−4), where n is the total number of input and I/O pins available on the device. This way is the preferred method of design. It is much easier and will save both time and components over implementing the kind of logic described in the next section for dual-function pins used in dual-function mode.

To design with the dual-function pins used in single-function mode, all you really need to do is make sure no I/Os get assigned to those dual function pins. Two ways to do this are described below.

First, if you are using the Cypress *Warp* VHDL compiler, you can use a simple synthesis directive called "pin_avoid" to make sure the compiler does not assign signals to whatever pins you specify. In this case, of course, you would specify the dual function pins. An example of the exact text to include in your VHDL code appears in *Figure 5*. This example assumes you are using the CY7C373i or CY7C374i in the 84-pin PLCC package where pins 14, 35, 51, 72, and 83 are the ISR pins.

If you prefer, or if the software you are using does not have a capability similar to the "pin_avoid" directive in *Warp*, you can also ensure the dual-function pins do not get used as I/Os in normal operating mode by explicitly assigning all of the signals to pins in your design. You just need to make sure you assign all of the signals to pins other than the dual-function pins. An example showing how to do this in *Warp* using the "pin_numbers" directive is shown in *Figure 6*. Again, this example assumes you are using the CY7C373i or CY7C374i in the 84-pin PLCC package, so pins 14, 35, 51, 72, and 83 are not being

```
-- example of using “pin_avoid” for single-function mode of
-- dual-function devices

entity cpuctl is port (
     a           :     in        bit_vector (31 downto 0);
     rd, wr      :     out       bit;
     hold        :     buffer    bit;
     status      :     out       bit_vector (7 downto 0));
end cpuctl;

attribute pin_avoid of cpuctl:entity is “14 35 51 72 83”;

-- architecture would follow
```

Figure 5. VHDL Code Fragment Showing pin_avoid Attribute

```
-- example of explicit pin assignments that avoid ISR pins
-- to facilitate single-function mode of dual-function devices

entity cpuctl is port (
     a           :     in        bit_vector (15 downto 0);
     rd, wr      :     out       bit;
     hold        :     buffer    bit;
     status      :     out       bit_vector (7 downto 0));
end cpuctl;

-- assign pins below and avoid pins 14, 35, 51, 72, and 83

attribute pin_numbers of cpuctl:entity is
“a(15) :12 a(14) :13 a(13) :15 a(12) :16 a(11) :17 a(10) :18 a(9) :19 “ &
“a(8) :24  a(7) :25  a(6) :26  a(5) :27  a(4) :28  a(3) :29  a(2) :30 “ &
“a(1) :31 a(0) :33 rd:36 wr:37 hold:38 status(7) :54 status(6) :55 “ &
“status(5) :56 status(4) :57 status(3) :58 status(2) :59 status(1) :60 “ &
“status(0) :61 “;

-- architecture would follow
```

Figure 6. VHDL Code Fragment Showing pin_numbers Attribute

used. Notice that none of the signals used in the example in *Figure 6* are assigned to these pins.

This approach can be more time-consuming than using the “pin_avoid” directive, especially if your design has a large number of I/Os. When you do this, you also need to take some device-specific resource information into account, such as block-reset and preset or half-block output-enable signal allocation. Since the compiler can account for all of this for you automatically, it is usually easier to just use the “pin_avoid” directive.

Devices With Dual-Function Programming Pins Used in Dual-Function Mode

There are cases where you may need or want to take advantage of the dual functionality of the dual-function programming pins. For example, you may not

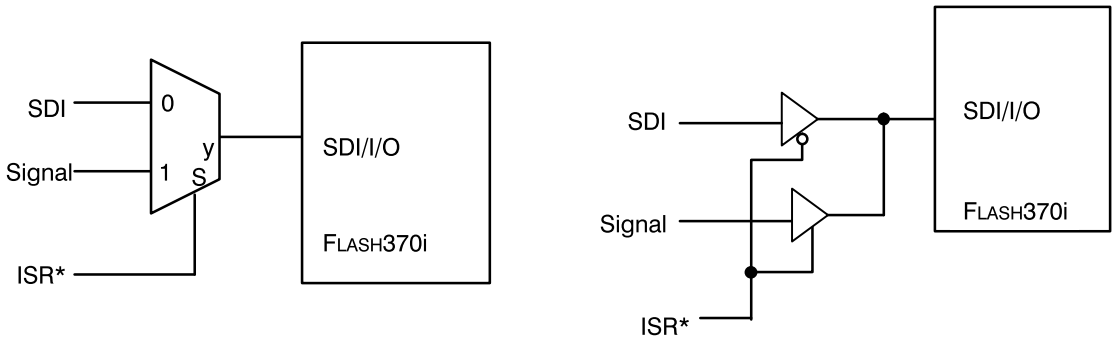
have enough I/O pins for your design if you do not use the dual-function ISR programming pins as I/Os when the device is in normal operation. Other times, you may want to use the dual-function ISR pins as I/Os in normal operation because their physical position makes your board layout easier. If you want to do this in your design, you can do it fairly easily; it simply requires a little bit of extra logic and possibly some additional small components. This section shows you how to do this.

The SDI, SCLK, and SMODE programming pins are all inputs to the device during programming, and they always share pins with bidirectional I/Os when they are dual-function pins. The SDO programming pin, on the other hand, is an output pin from the device during programming. It, too, always shares a pin with a bidirectional I/O when it is a dual-function pin. These I/O pins, in turn, can be used as input only, output only, or bidirectional I/Os in any design, based upon the functionality that is described for these pins in the programmable logic chip's design description. The result is that there are six different cases to consider: you can have an ISR input programming pin (SDI, SCLK, SMODE) sharing a pin with a signal that is an input, an output, or an I/O; and you can have an ISR output programming pin (SDO) sharing a pin with a signal that is an input, an output, or an I/O. We next look at each of these six cases individually.

What you are trying to accomplish in all of these cases is fundamentally the same. You are trying to isolate the programming signals from the normal operating signals on the board. You do not want the programming signal to drive or affect anything else on the board when you are programming the FLASH370i device, and you do not want the normal operating signal to drive, affect, or be affected by the programming logic when the FLASH370i device is operating normally in the system. The basic strategy in all of the cases listed above is to use three-state buffers or multiplexers on these signals, and to have those buffers or multiplexers controlled by the ISR* signal from the programming cable. The ISR* signal, recall, is a signal from the programming cable that is a logic '0' when ISRVPP is 12V (when the FLASH370i device is being programmed), and it is a logic '1' when ISRVPP is not 12V (when the FLASH370i device is not being programmed).

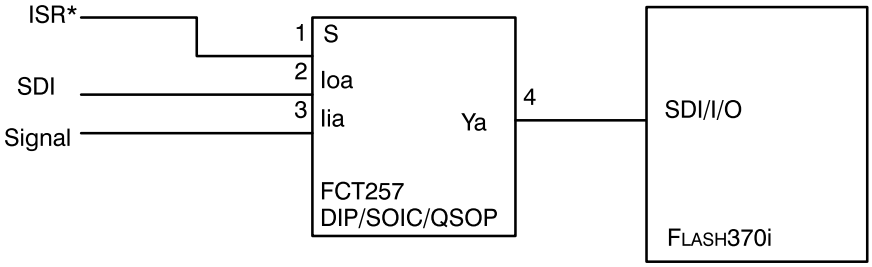
First, consider the case of the ISR programming pins that are inputs to the device during programming, SDI, SCLK, and SMODE. When one of these device pins is being used as an input during normal operating mode, you simply have to select between one of two inputs based on whether you are in programming mode or in operating mode. This is implemented very easily by using a 2:1 multiplexer where ISR* is the select line, as shown in *Figure 7(a)*. Alternatively, you could implement this by having two three-state buffers whose inputs are SDI (or SMODE or SCLK) and *signal*, whose outputs are tied together and to the SDI / I/O pin, and whose enable lines are controlled by opposite values of ISR*. This is shown in *Figure 7(b)*. One way you could implement this logic is with FCT-family devices. For example, you could use one of the four 2:1 multiplexers in a CY74FCT257 to implement the logic shown in *Figure 7(a)*. Alternatively, you could use a pair of transceivers or pass-transistors from a CY74FCT244 or CYBUS3384 to implement the logic shown in *Figure 7(b)*. The connections for the FCT257, FCT244, and CYBUS3384 are shown in *Figure 7(c)*, *7(d)*, and *7(e)*, respectively. The FCT devices shown are just one possible way of implementing this logic, of course. There are others, including using extra pins and gates from an ASIC, FPGA, CPLD, or PAL already on the board. Regardless of whether the buffer or multiplexer is in an FCT device, ASIC, FPGA, or other device, there will be some additional propagation delay for the normal operating signal due to the presence of that logic. This must be accounted for in your design, and using the CYBUS3384 provides the smallest extra delay. The issue of extra delay holds true for the other cases presented next.

In the case of the SDI, SCLK, or SMODE sharing a pin with an I/O that is used only as an output during normal operating mode, the logic is slightly different. Much like the case above, you can just use a pair of three-state buffers or pass-transistors to separate the signals that are used for the two different functions. In this case, however, instead of tying the two outputs together, you tie the output of one

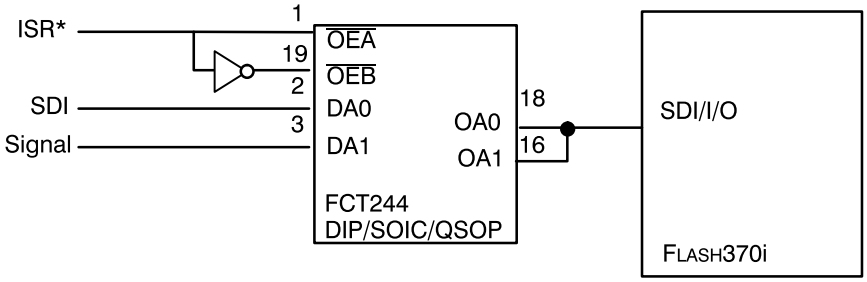


(a) Multiplexer Solution

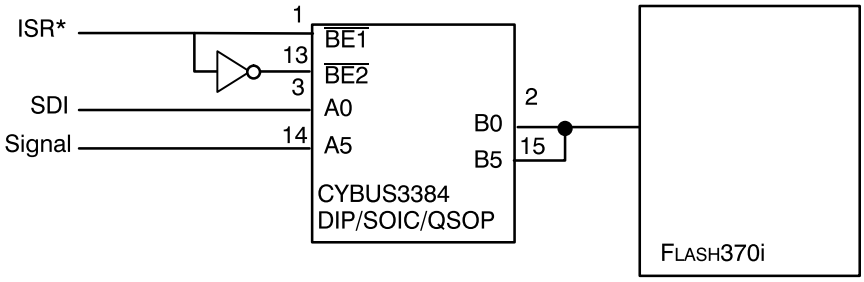
(b) Buffer Solution



(c) FCT257 Implementation



(d) FCT244 Implementation



(e) CYBUS3384 Implementation

Figure 7. Design for Dual-Function Pins: SDI/SCLK/SMODE used with Input

buffer both to the dual-function pin of the FLASH370i device and to the input of the other buffer. The input to the first buffer is the programming function signal, and the output from the other buffer is the normal operation output *signal*. The first buffer is enabled when ISR* is asserted and is disabled otherwise, and the second buffer is enabled when ISR* is deasserted and is disabled otherwise. This is shown in *Figure 8*. Thus, when the device is being programmed, SDI (or SMODE or SCLK) is driving the SDI / I/O pin and *signal* is in the high-impedance state, and when the device is not being programmed, the SDI / I/O pin is not driven as an input allowing the FLASH370i output to drive *signal*. Because *signal* is in the high-impedance state during programming, you may need to have a pull-up or pull-down resistor on *signal* depending on how you use it on your board.

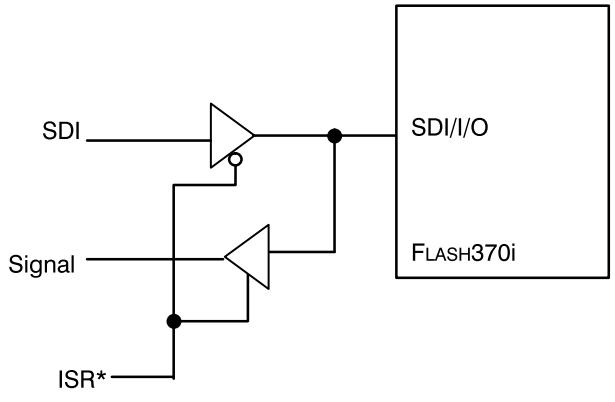


Figure 8. Design for Dual-Function Pins: SDI/SCLK/SMODE used with an Output

You can also use a CYBUS3384 as an alternative to the buffers, just as in solution 7(e) in the previous case. This would be the most flexible solution because it would work for all configurations—the pin used as an I, O, or I/O—and allows you to decide later exactly how to use that pin.

Another alternative exists for the case when SDI (or SMODE or SCLK) has dual functionality with an I/O pin that is used as an output during normal operating mode. You can simply connect *signal* and SDI (or SMODE or SCLK) together directly without any isolating gates. Since *signal* is an output from the FLASH370i in normal operation, no damage should result from *signal* being driven identically to SDI during ISR programming. The only precaution you must take is to ensure the circuitry that is being driven is not affected logically by the values on SDI, such as a state machine being put into the wrong state. If that circuity can ignore SDI's values during programming, then this would be the preferred solution. It does not add any extra delay to *signal*'s path and does not require any additional devices or logic, thus allowing you to use the three ISR dual-function pins, SDI, SMODE, and SCLK "for free."

In the case of the SDI, SCLK, or SMODE sharing a pin with an I/O that really is used as a bidirectional I/O, the logic needed is a little more complicated. As seen in *Figure 9*, part of the logic is a combination of the two solutions for the two individual cases above in the way it uses ISR* to separate the programming function of SDI (or SMODE or SCLK) from the input and output functions of *signal* during

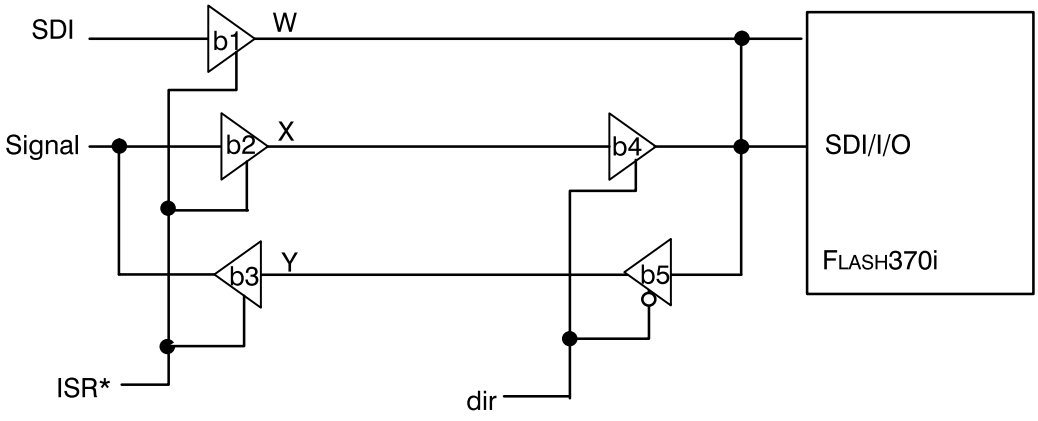


Figure 9. Design for Dual-Function Pins: SDI/SCLK/SMODE used with an I/O

normal operating mode. There is more than just this logic required, however. It is also necessary to use an extra pair of buffers to separate the input and output functionality of *signal* itself. This is required to keep from unintentionally building a feedback loop and is accomplished with the help of an extra signal that indicates the direction of the I/O pin. In this example, we assume we have a signal called *dir*, and that *dir* = '1' when the I/O pin is being used as an input and *dir* = '0' when the I/O pin is being used as an output.

To understand why this is necessary, consider just combining the logic from *Figure 7(b)* and *Figure 8*. The result would be the logic shown in *Figure 10*, which is different from *Figure 9* in that buffers *b4* and *b5* were eliminated and intermediate signals *w*, *x*, and *y* are now all simply connected together and to the SDI / I/O pin. In the logic of *Figure 10*, when the FLASH370i device is in normal operation mode and ISR* is HIGH, buffers *b2* and *b3* would both be enabled. If *signal* were an input at that time, it would drive the input to buffer *b2*, whose output would drive the input to buffer *b3*. The output of buffer *b3* would be driving the input of *b2* again, resulting in a feedback loop that could produce undesired affects. The same thing would happen if *signal* were an output at the time.

Buffers *b4* and *b5* in *Figure 9* prevent this. In the logic of *Figure 9*, when *signal* is an output from the FLASH370i device, *b5* is enabled and *b4* is disabled; when *signal* is an input to the device, *b4* is enabled and *b5* is disabled. In both cases, both the function and value at the pin of the device and the function and value of *signal* are the same, correct, and only driven by one source. There is no dangerous self-driving feedback system like there is in *Figure 10.*

The limitation of this solution is that it requires the extra signal *dir*. This signal may be already available; in fact, it may be an input to the FLASH370i device itself for use as the $\overline{OE}$-control on the pin in question. If it is not already available, you will need to generate it using other logic on the board. If you cannot do it using other logic on your board, you should certainly be able to generate it using logic inside the FLASH370i device itself, because, as pointed out above, it should be the same signal as the $\overline{OE}$ used on that pin internally. To get the signal out of the FLASH370i, however, requires an additional pin, so if you are using the logic in *Figure 9* to save a pin, having to use a pin on the device to generate *dir* will not gain you anything. If generating one *dir* will help you save two or three pins by allowing you to use two or three of SDI, SCLK, and SMODE as dual-function pins, then you will still have a net savings of one or two pins and it may be worth it.

As was mentioned in the case where the SDI (or SMODE or SCLK) dual-function pin was being used with an input-only pin or with an output-only pin, you can also use the CYBUS3384 solution of *Figure 7(e)* when trying to use the SDI (or SMODE or SCLK) dual-function pin as a bidirectional I/O pin in normal operating mode.

The logic for using the dual-functionality of the SDO / I/O pin is essentially the same as is shown in the above three cases. The only difference is that

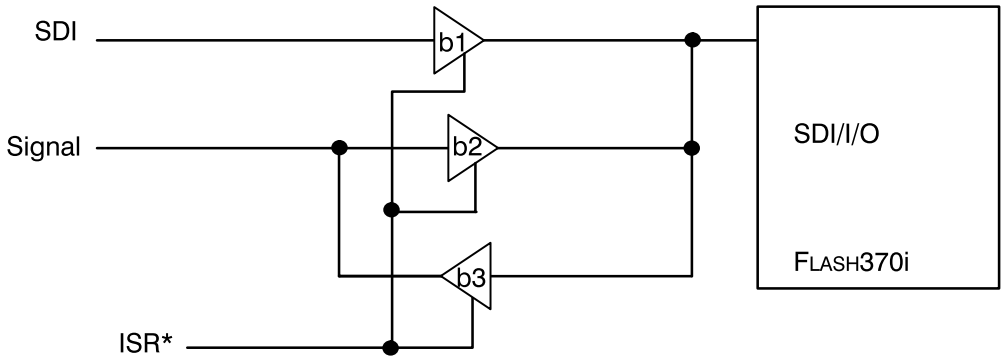


Figure 10. SDI/SCLK/SMODE used with an I/O: Example of Incorrect Solution

SDO is an output during programming mode instead of an input. Therefore, the only difference in the logic is the orientation of some of the buffers. The solutions for the SDO case are presented without further explanation. The logic diagram for the case where SDO is connected to an I/O used only as an input is shown in *Figure 11*; the logic diagram for the case where SDO is connected to an I/O used only as an output is shown in *Figure 12*; and, the logic diagram for the case where SDO is connected to an I/O really used as a bidirectional pin is shown in *Figure 13*. You can alternatively use the CYBUS3384 solution presented in *Figure 7(e)* in each of these three cases.

To summarize this section, there are many reasons to use the members of the FLASH370i family that have dual-function ISR programming pins and many different ways to accomplish it. The easiest way is to use the dual-function device in the single-function mode. This uses the dual-function pins as programming pins only, and is easily accomplished using the pin_avoid and pin_numbers directives in your *Warp* design file. There are also going to be cases where you will want to use the dual-functionality, most likely because you need some or all of the four ISR programming pins as inputs, outputs, or I/Os during normal operation to get all the signals you need into and out of the device for your design. The circuits needed to share these pins are relatively straightforward and require nothing or only buffers or pass transistors. These are circuits you can implement using FCT or other logic, or you may be able to implement them using extra gates and pins of an ASIC, FPGA, or another PLD you already have on the board.

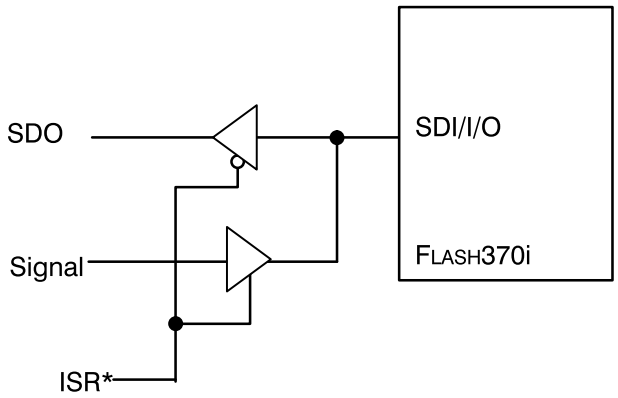


Figure 11. Design for Dual-Function Pins: SDO used with an Input

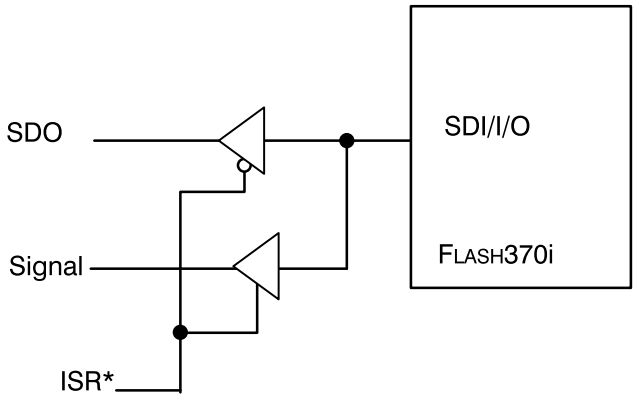


Figure 12. Design for Dual-Function Pins: SDO used with an Output

Simple Cascading and the ISR Software Configuration File

Until now, we have talked about programming just a single FLASH370i device in the system. You can cascade an unlimited number of FLASH370i devices in a system. That is, you can daisy-chain the devices together and connect their programming pins in such a way that all of the devices can be programmed from a single connection to the ISR programming cable. To do this, you simply tie all of the SCLK, SMODE, and ISRVPP pins of each device to those same pins, respectively, on all of the other devices, and then connect them to the corresponding pins of the ISR cable connector. You then connect the SDI pin from the cable connector to the SDI pin of the first device in the chain, then connect the SDO output of that device to the SDI input of the next device in the chain, then connect the SDO output of that device to the SDI input of the next device in the chain, and so forth, until you finally connect the SDO output of the last device in the chain to the SDO pin of the cable connector (see *Figure 14*).

We have taken for granted so far that there is ISR programming software running on the PC that accesses the FLASH370i JEDEC files and drives the ISR programming cable to program the FLASH370i devices. This ISR software, which is provided by Cy-

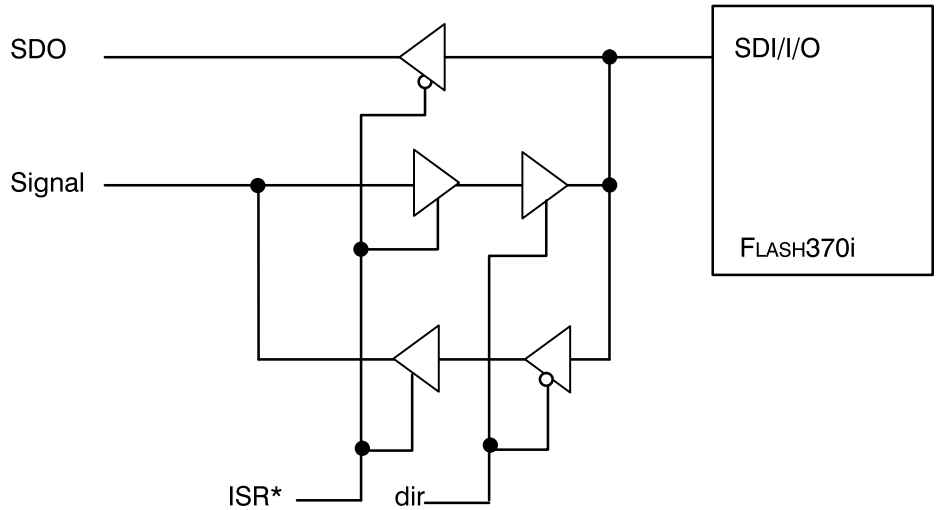


Figure 13. Design for Dual-Function Pins: SDO used with an I/O

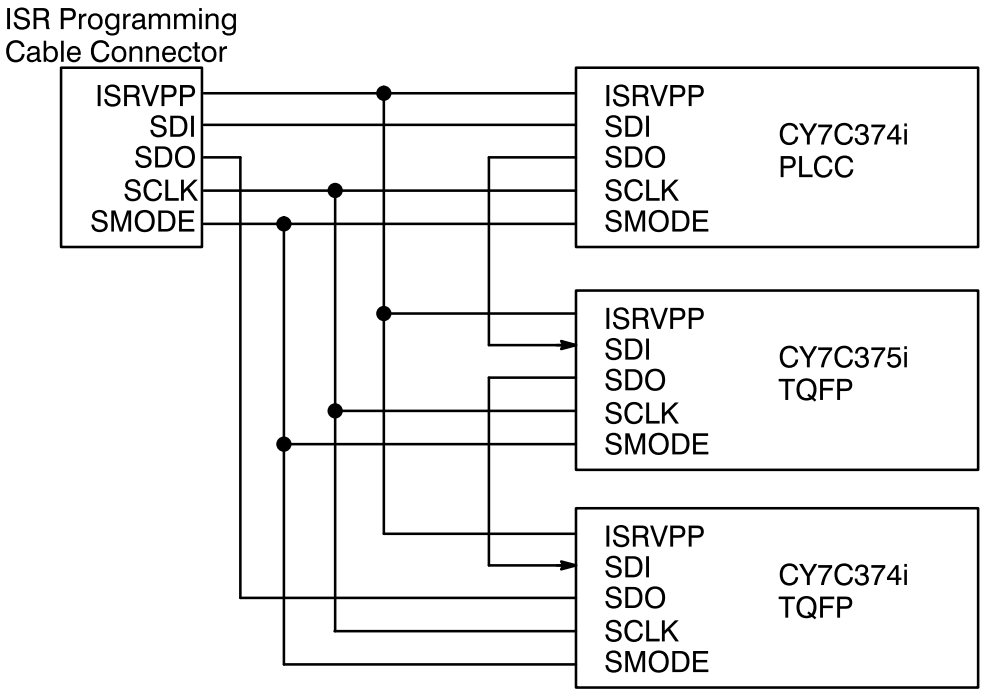


Figure 14. Simple Cascading Example

press, reads a simple configuration file provided by the user. This configuration file is where the user specifies what the JEDEC files are, what the FLASH370i devices to be programmed are, and what other operations, if any, are to be performed. For example, to program a single CY7C374i with a JEDEC file called sarctl.jed, the configuration file would just be the single line shown below:

```
CY7C374i p c:\sarctl.jed;
```

The first field specifies the FLASH370i device; the second field specifies the operation; and the third

field, if necessary, specifies the full path of the name of the file to be used.

As previously mentioned, multiple FLASH370i devices can be chained together for ISR programming purposes. The configuration file in this case contains all of the devices in that chain, which may also include non-Cypress devices. The first line of such a configuration file corresponds to the first device in the chain, and the last line to the last device in the chain. As is the case in a single-device configuration file like the one described before, the first field in each line specifies the device type; the second field specifies the operation; and the third field, if necessary, specifies the full path of the file name to be used. The full list of operation codes that can be used in the second field along with their meaning is shown below:

p program device with listed JEDEC file

v verify device contents against listed JEDEC file

r read device and create listed JEDEC file

e erase device

d get silicon ID

u get user code

n no operation

s program security bit

i initiate power-on-reset

A non-Cypress device can be included in the chain with FLASH370i devices if the device has a JTAG interface, contains the standard JTAG TAP controller state machine, and supports a bypass instruction which uses the specified code of all '1's. In the case where the chain of cascaded devices includes a non-Cypress device, the line in the configuration file for that device has the key word "NONCYPRESS" as the first entry and an integer that specifies the number of bits in that device's instruction register in the second field. The JTAG specification defines the bypass instruction to be all '1's, so the ISR programming software will insert the number of '1's necessary to fill the non-Cypress device's instruction register in the appropriate place in the bitstream that is sent.

An example of a multiple-device, multiple-operation configuration file is shown here:

```
CY7C375i v c:\dramctl.jed;
CY7C374i n;
CY7C373i r c:\cpuctl.jed;
CY7C374i p c:\sarctl.jed;
NONCYPRESS 4;
CY7C372i u;
```

This configuration file specifies that there are six devices in the cascaded chain, that a CY7C375i is the first in the chain, that a CY7C372i is the last in the chain, and that there is one non-Cypress device in the chain. The first device, a CY7C375i, is to have its JEDEC file read and verified against the file c:\dramctl.jed; the second device, a CY7C374i, is to be passed over—no operation is to be performed; the third device, a CY7C373i, is to have its JEDEC file read and stored in the file c:\cpuctl.jed; the fourth device, a CY7C374i, is to be programmed with the file c:\sarctl.jed; the fifth device is the non-Cypress device, and it has a four-bit instruction register; and the sixth device is a CY7C372i, and the user code already programmed into the device is to be read.

For more detailed information on cascading FLASH370i devices with other JTAG devices for ISR programming and system diagnostics, refer to the application note titled "Cascading FLASH370i Devices."

Other Considerations

In addition to the topics covered above, there are several other miscellaneous board-level design issues you should consider when using FLASH370i devices. These include the state of the FLASH370i devices at power-up (and why you need to know about this); the state of the FLASH370i programming pins when the ISR programming cable is not attached; and handling the 12V ISR programming signal on the board. We address these considerations here.

State of the FLASH370i Device I/Os at Power-Up

When FLASH370i devices are shipped from Cypress, the devices have already been programmed, erased, and programmed again as part of the testing process. They will not, therefore, be blank when they first come out of the tube. They will, however, be programmed such that all of the I/Os are three-stated. Furthermore, the I/Os (except SDO) are also all three-stated during device programming, that is, when ISRVPP is at 12V. This allows you to solder FLASH370i devices directly on your board without having to erase them first, and it will allow you to power-up your board and program the FLASH370i devices on it without having to worry about their initial, non-blank state causing any problems.

The most likely way people would want to use these parts is to take FLASH370i devices directly from the tube, solder them onto their board, turn on the power to the board, and then program the devices for the first time using the ISR programming cable connected to a PC. Since many of the I/Os on the FLASH370i device(s) to be programmed will undoubtedly be inputs, other devices on the board could be driving those pins immediately upon powering up the system. By having all of the FLASH370i I/Os initially programmed to be three-stated, and by having them also be guaranteed to be three-stated during ISR programming, you are assured that the FLASH370i device will not be also trying to drive those pins. This prevents bus contention that could otherwise arise, and it prevents the damage to a FLASH370i device or other devices on your board that could result from it. When the 12V ISRVPP signal goes away, the FLASH370i device will then start driving some of its output pins, based upon the FLASH370i device being programmed according to your design. At this point, it is no different from powering up a board with preprogrammed non-ISR PALs, PLDs or CPLDs on it.

State of the FLASH370i's Programming Pins When the ISR Programming Cable is Not Attached

It is likely that the ISR programming cable will not always be plugged into the connector on your board, so it is important to understand that this will not be a problem when the board is powered up and expected to be running. The reason it will not be a problem is that the FLASH370i devices have been designed with bus-hold structures on every input, input/clock, and I/O pin, including the programming pins whether they are single-function or dual-function. The exception to this is the SDO pin on single-function devices, which is an output only. This eliminates the need to use external pull-up resistors or any other technique for handling the case where the ISR programming pins are left floating due to the ISR programming cable being disconnected.

Bus-hold structures enable an I/O pin to maintain its most recent logic value even when it is three-stated, whether that value was being driven in as an input pin or driven out as an output pin. This is done with an extremely weak latch connected to the pin. Since the ISR programming pins have these bus-hold structures, if the ISR programming cable is disconnected when the board is powered on, the ISR programming pins will all maintain a logic '0' or logic '1' value even though they are no longer being driven. These pins will not be floating between these logic levels, and therefore, they will not be subject to oscillation and will not be sourcing or sinking any more current than the bus-hold currents specified in the data sheet. If the ISR programming cable is disconnected when the board is powered-down, or if the board is powered down and then back up after the cable has been disconnected, there is still no problem. The FLASH370i bus-hold structures have been designed to always power-up with a logic '1' level maintained on the pins to emulate an internal pull-up, but this does not interfere with the capability of maintaining the last state when not driving or not being driven.

The advantages of the bus-hold structures apply both to the case of ISR programming pins used as single-function pins and as dual-function pins. In the single-function case, they can be taken advantage of exactly as described above. In the dual-function case, they also behave as described above, and in addition they do not interfere with the normal function of the pin. Once the device is no longer being programmed, the normal function of the pin be-

comes its main use. If, in that normal function, the I/O is first three-stated, the bus-hold structure operates as described before; if, in that normal function, the I/O is first driven as an output, the bus-hold structure does not interfere; and if, in that normal function, the I/O is first driven as an input, the bus-hold structure holds the last value until a new one is driven.

The bus-hold structures are also useful on normal pins (i.e., non-programming pins). Just as the bus-hold eliminates the need for external pull-up resistors, or any other method for dealing with floating ISR programming pins, they also eliminate the need for dealing with floating non-programming, general-purpose input and I/O pins. This can be especially useful during the debugging of a design where you may want to leave all unused pins on a CPLD unconnected in case you need to add more signals as the design evolves. You can take advantage of the bus-hold structures to make that as easy as possible. You simply do not use those pins when describing and compiling your design, and you then leave those unused pins unconnected on your board. The bus-hold structures then act as pull-ups on these pins, but they are better because there is no external resistor connecting the pin to VCC or GND. This most importantly saves you time. When you need to add a signal to a formerly unused pin, you simply make that connection on your board; you do not also have to break the connection to VCC or GND like you would have to if you used an external pull-up.

Note that the ISR* pin from the ISR programming cable connector does not necessarily connect to a FLASH370i I/O pin. Since it does not, you must use a pull-up on the ISR* signal on your board so that it is not left floating when the ISR programming cable is disconnected.

Handling the 12V Signal on the Board

There are two requirements on the ISR programming voltage that necessitate special handling. The first is that its voltage must be in the range $11.4V \leq ISRVPP \leq 12.6V$ during programming, and the second is that its maximum current is 40 mA per FLASH370i device during programming. The 5V/12V DC/DC converter and other components in the ISR programming cable described in this application note have been chosen to ensure that these specifications are met. Because of this, if you are using the ISR programming cable for programming the FLASH370i devices, it would be a good idea to just use the 12V supplied by the ISR programming cable for ISRVPP even if 12V is available on the board on which the devices are being used or from the backplane to which that board is connected.

Because of the higher than usual current and voltage requirements on the ISRVPP signal, the trace on the printed-circuit board connecting the ISRVPP pin from the ISR programming cable to the ISRVPP pin on the FLASH370i device(s) also deserves special attention. First, to handle the current, the trace should be double the width of the standard traces. Second, the trace should be kept as short as possible. In general, this means the connector for the ISR programming cable should be place as close as possible to the FLASH370i devices on the board. Since the connector is small, it is much easier to move the connector to be close to the devices than change the whole board layout to place the devices close to the chosen spot for the connector.

Conclusion

In-system reprogrammability (ISR) in a CPLD has several benefits. It allows engineering development and debugging without having to socket the CPLDs and without having to remove them and reprogram them in a device programmer. This saves time regardless of the package type you decide to use. ISR is especially valuable when you decide to use fine-pitch packages like TQFPs. Like before, it allows you to use them without sockets, which means, again, no handling of devices for reprogramming. Not only does that save time, but in this case, it also avoids the higher potential for bending leads on very fine-leaded devices. Also, by allowing you to solder TQFP packages directly onto a board without sockets, it helps you avoid spending time simply checking device-to-socket-lead connections during debugging. ISR also allows for designs which can be reconfigured in the field, either by a software update or by an input from the system they are in. The supe-

rior routability and flexible architecture of the Cypress FLASH370i CPLDs enhance the value of all of these benefits greatly by allowing you to actually make design changes during prototyping, debugging, or field operation and still successfully route to the already-defined pinout, even on designs that are utilizing most or all of many of the device's resources.

This application note shows how to take advantage of the in-system reprogrammability of the FLASH370i family by using a cable connected to the parallel port of a PC for programming the CPLDs on a board. This is typically the way these parts are used during development and debugging, which, as described above, is an area where the FLASH370i architecture and routability are particularly useful. This application note explains all of the details of the programming cable and the signals it uses, and it also covers many design techniques and considerations that show how to most easily use the desired capabilities of these parts. These include logic designs for using the dual-function pins on FLASH370i devices whose programming signals share pins with I/O signals, tips for handling the 12-volt programming signal on your board, and details of the ISR programming software configuration file.