How to create a simple ColdFire and Altera FPGA IOC

W. Eric Norum

September 22, 2008

Contents

Chapter 1

Introduction

This tutorial presents a step-by-step series of operations for creating a simple EPICS application for an Arcturus uCDIMM ColdFire 5282 module attached to an Altera FPGA development kit. The following software and hardware components are assumed to be in place:

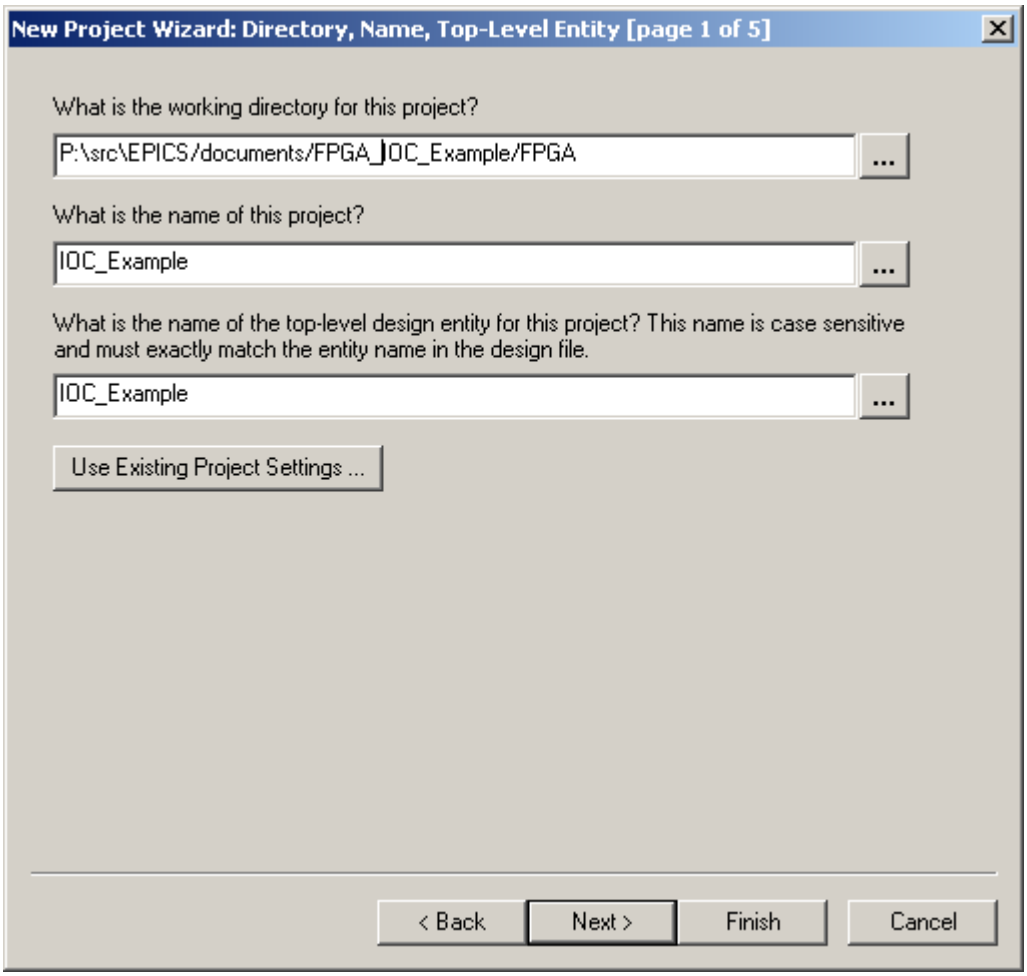
- Arcturus uCDIMM ColdFire 5282
- RTEMS with m68k tool chain and uC5282 board-support package
- EPICS version R3.14.7 or greater with RTEMS-uC5282 target
- Altera FPGA development kit with at least two sets of expansion prototype connectors.
- ColdFire/FPGA adapter card
- Quartus version 6.0 or greater and Altera SOPC builder (this tutorial shows screen captures from Quartus version 7.1)
- ColdFire bridge SOPC component
- Console reset detect Quartus component (optional)

This tutorial is written for use with an Altera Stratix II DSP development kit. The changes required for use with other development kits are summarized in appendix B.

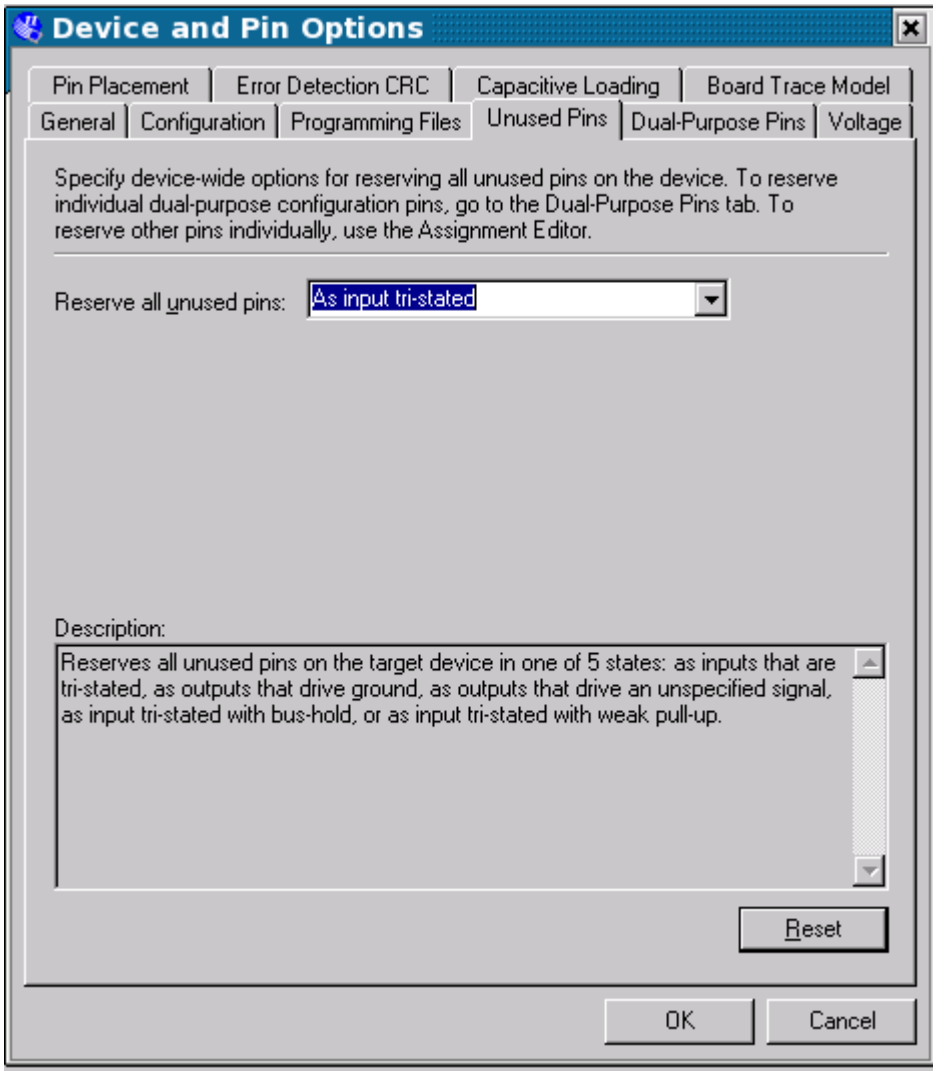
Chapter 2

FPGA application

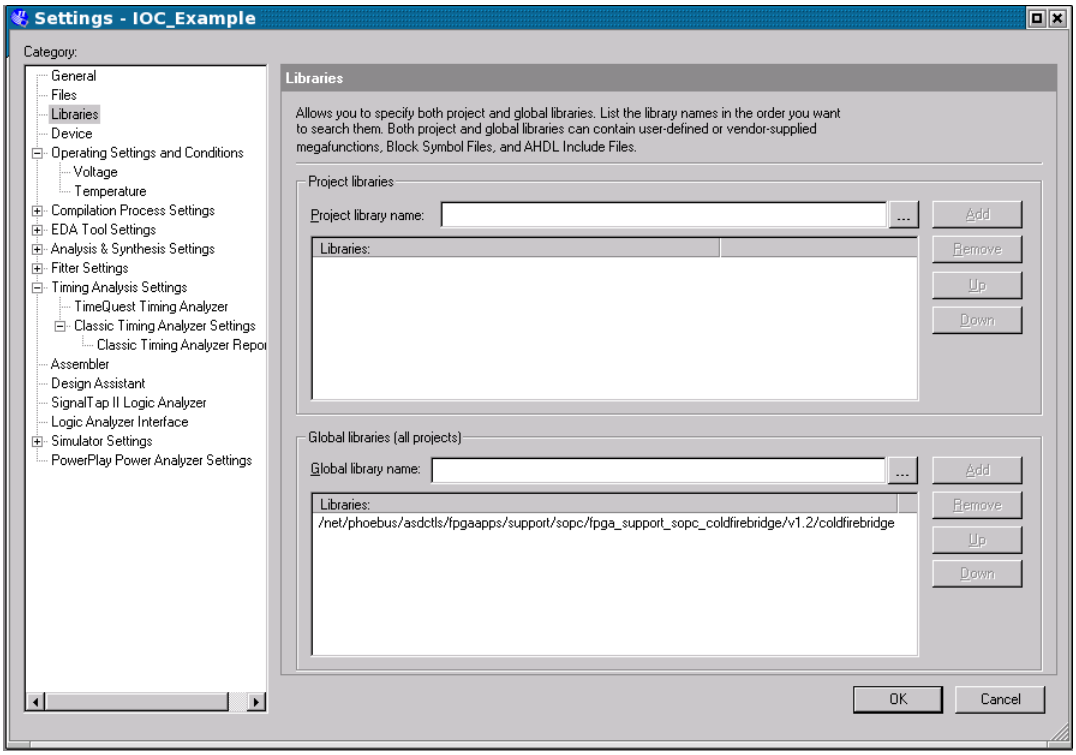
1. Use the Quartus ”New Project Wizard” to create a new project. I called the project `IOC_Example` with same name for the top-level entity:



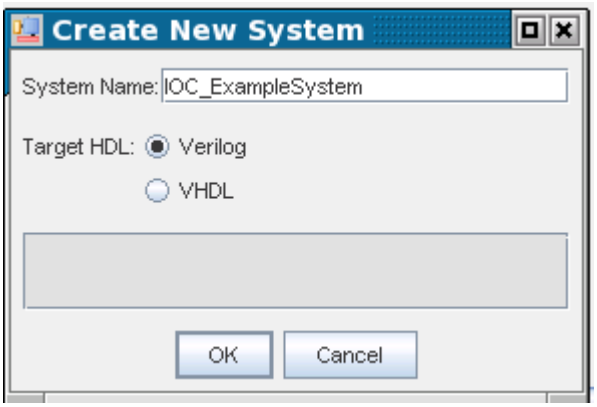
2. Ensure that unused pins are treated as inputs (Assignments→Device..., Device & Pin Options, Unused Pins tab). Not all the ColdFire signals are used by this example and Quartus helpfully grounds all the corresponding pins if this step is omitted!



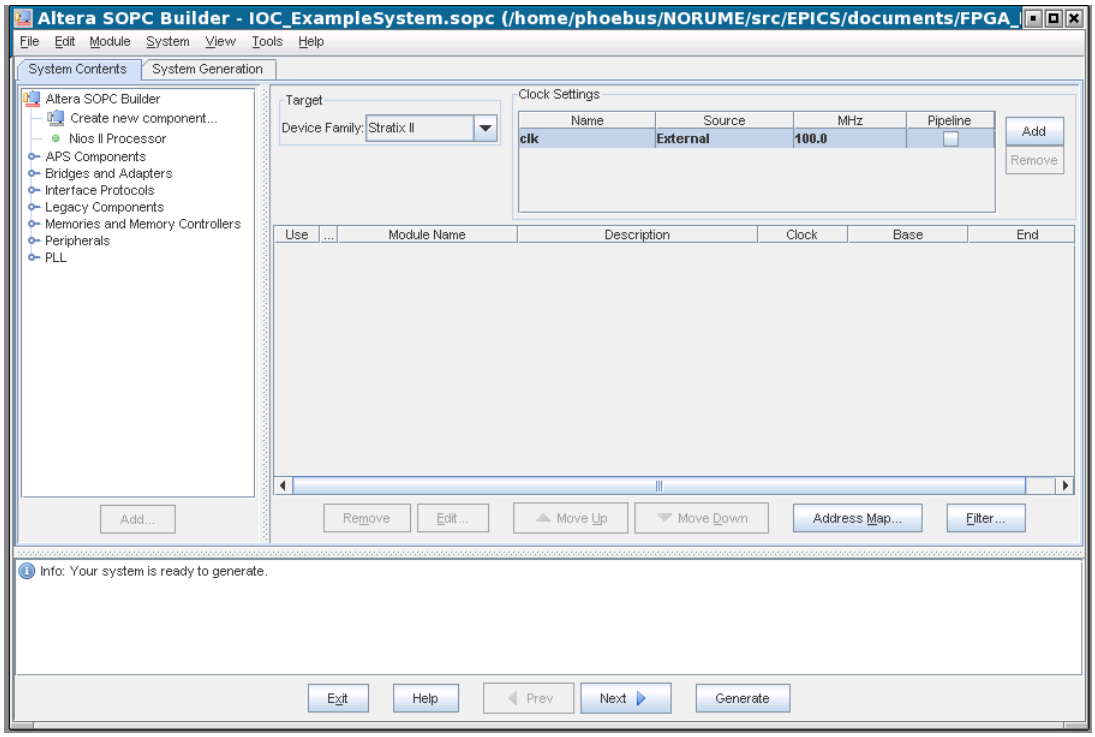
3. If you have not done so in a previous SOPC project, add the directory containing the ColdFire Bridge SOPC component to the list of global libraries (Assignments→Settings→Libraries). Note that this list of directories applies to SOPC Builder sessions in **all** projects. When the SOPC Builder is started and searches the directory containing the ColdFire Bridge SOPC component it will add the ColdFire Bridge component to the "APS Component" list.



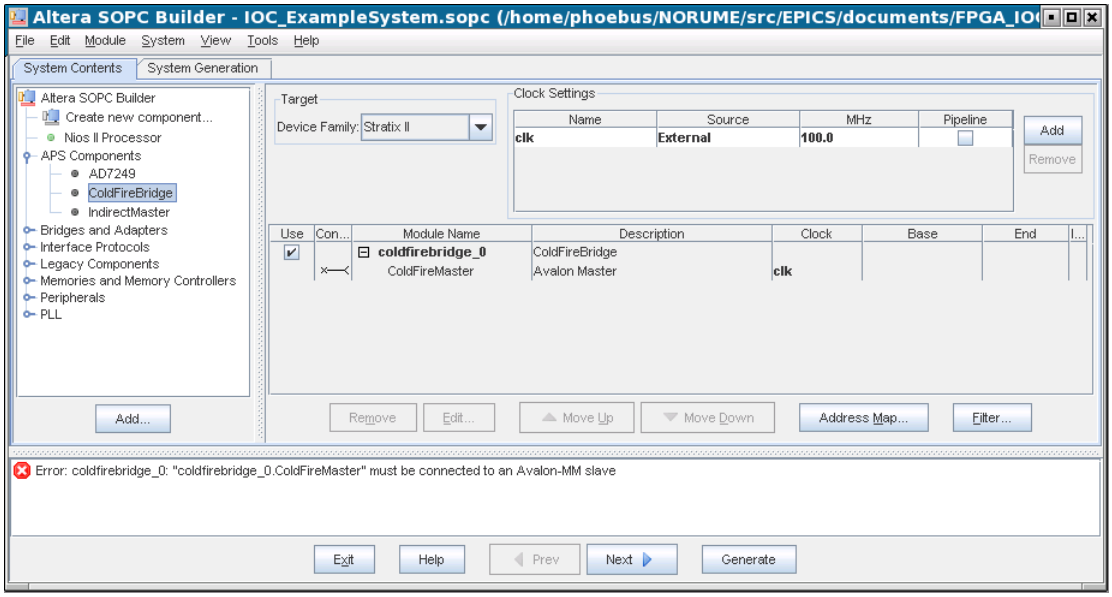
4. Start the SOPC Builder (Tools→SOPC Builder...) and create a new SOPC system. I called the SOPC system `IOC_ExampleSystem`:



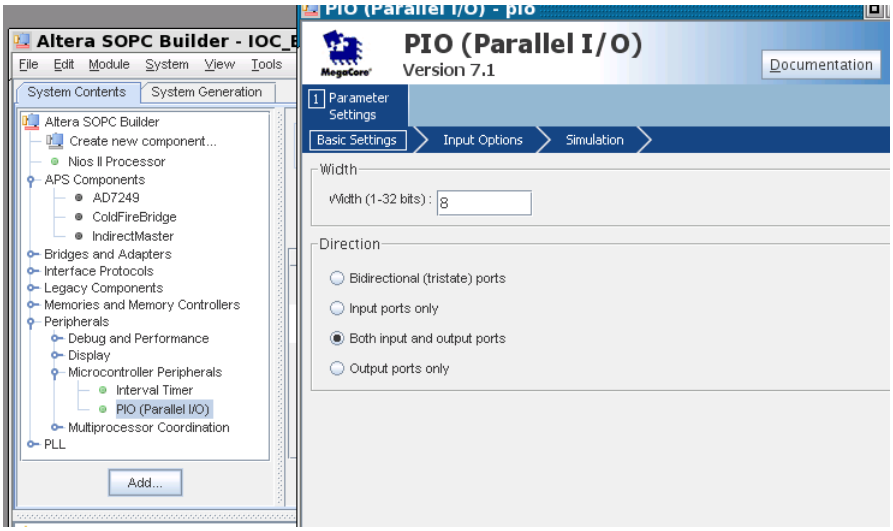
5. Set the SOPC Device and Clock parameters:



6. Double-click the ColdFire Bridge component to add it to the SOPC design:



7. Add an 8-bit port to the SOPC design:



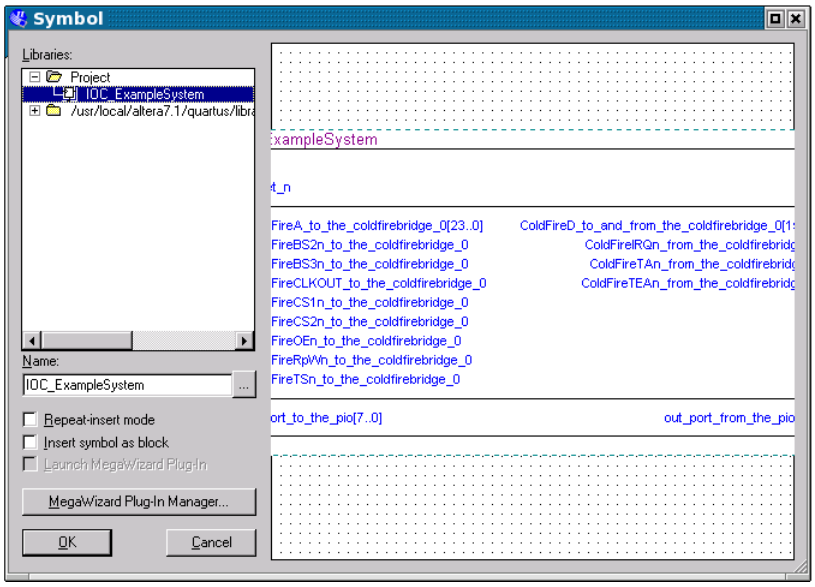
Although the port will be used as an output port to drive the LEDs it is a good idea to provide an input port as well so that the IOC can read back the value currently written to the output. This makes it "bumpless" IOC reboot possible.

Connect the I/O port to the ColdFire bridge and give the port an address. The SOPC components and connections should then be:

8. Click the `Generate` button to create the SOPC block.

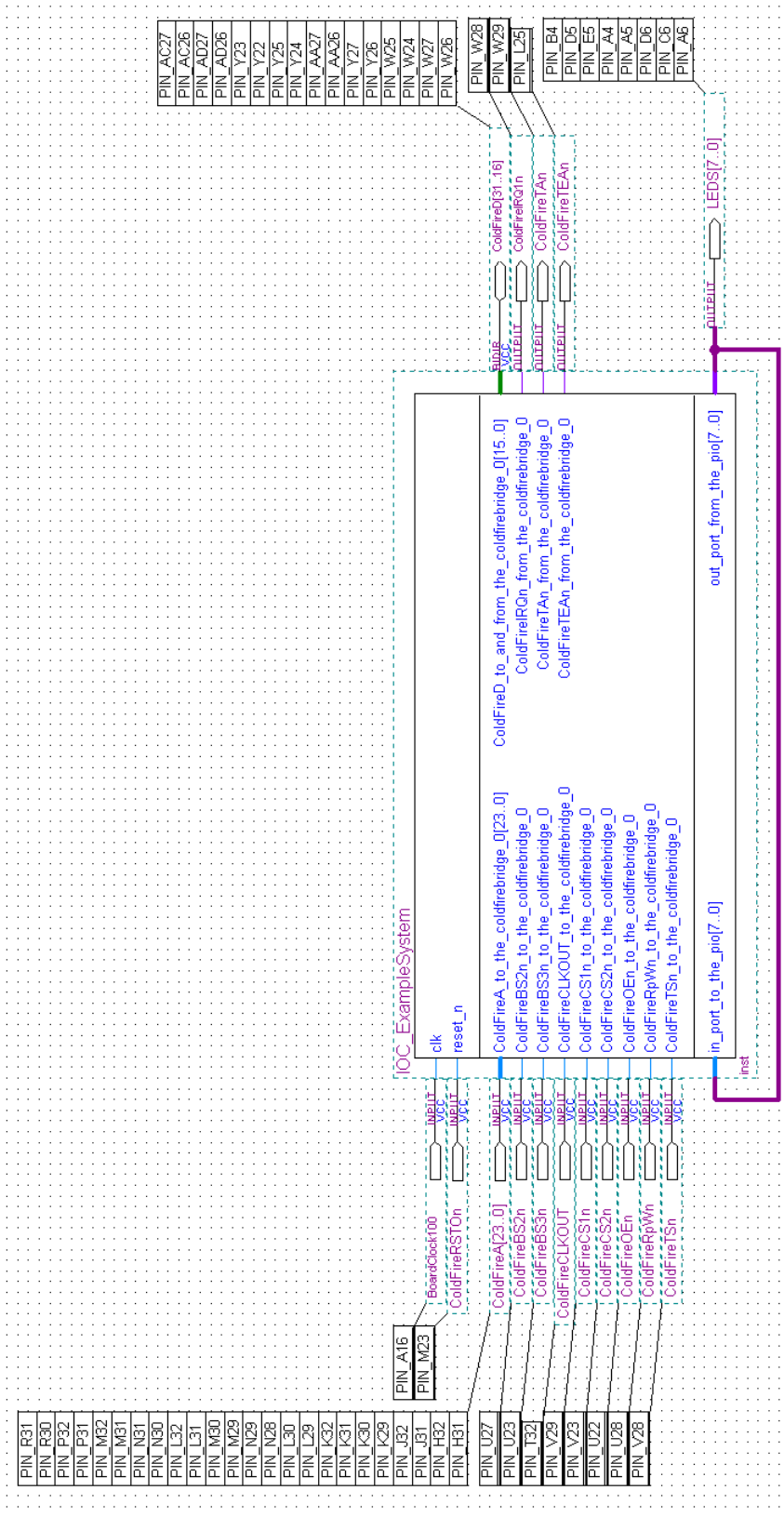
9. Create the top-level entity design file. In the Quartus window select File→New, create a new Block Diagram/Schematic File and save it as `IOC_Example`.

10. Double-click in the `IOC_Example.bdf` design window and add the `IOC_ExampleSystem` SOPC block to the design:



11. Add I/O pins and assign them to the appropriate FPGA pins using the Assignment Editor. A list of the pin assignments for several different development kits is included in appendix B. The complete system should then appear as shown on the following page. Notice the the ouput port is connected back to its input as well as to the LED pins.

12. Compile the project (you'll see lots of warnings...) and load it into the FPGA.


PIN_AC27
PIN_AC26
PIN_AD27
PIN_AD26
PIN_Y23
PIN_Y22
PIN_Y25
PIN_Y24
PIN_AA27
PIN_AA26
PIN_Y27
PIN_Y26
PIN_W25
PIN_W24
PIN_W27
PIN_W26
PIN_W28
PIN_W29
PIN_L25
PIN_B4
PIN_D5
PIN_E5
PIN_A4
PIN_A5
PIN_D6
PIN_C6
PIN_A6
ColdFireD[31..16]
ColdFireIRQn
ColdFireTAn
ColdFireTEAn
LEDS[7..0]
OUTPUT
VCC
GND
IOC_ExampleSystem
clk
reset_n
ColdFireA_to_the_coldfirebridge_0[23..0]
ColdFireBS2n_to_the_coldfirebridge_0
ColdFireBS3n_to_the_coldfirebridge_0
ColdFireCLKOUT_to_the_coldfirebridge_0
ColdFireCS1n_to_the_coldfirebridge_0
ColdFireCS2n_to_the_coldfirebridge_0
ColdFireOEn_to_the_coldfirebridge_0
ColdFireRpWn_to_the_coldfirebridge_0
ColdFireTSn_to_the_coldfirebridge_0
ColdFireD_to_and_from_the_coldfirebridge_0[15..0]
ColdFireIRQn_from_the_coldfirebridge_0
ColdFireTAn_from_the_coldfirebridge_0
ColdFireTEAn_from_the_coldfirebridge_0
in_port_to_the_pio[7..0]
out_port_from_the_pio[7..0]
inst
PIN_A16
PIN_M23
BoardClock100
ColdFireRSTOn
INPUT
PIN_R31
PIN_R30
PIN_P32
PIN_P31
PIN_M32
PIN_M31
PIN_N31
PIN_N30
PIN_L32
PIN_L31
PIN_M30
PIN_M29
PIN_N29
PIN_N28
PIN_L30
PIN_L29
PIN_K32
PIN_K31
PIN_K30
PIN_K29
PIN_J32
PIN_J31
PIN_H32
PIN_H31
PIN_U27
PIN_U23
PIN_T32
PIN_V29
PIN_V23
PIN_U22
PIN_U28
PIN_V28

Chapter 3

EPICS application

The steps listed below show how to create an example EPICS IOC application which uses the ASYN I/O environment to control the LEDs on the FPGA card.

1. Create a new EPICS <TOP> directory and make a new application in it. You must specify the full path to the makeBaseApp.pl script in your EPICS installation:

   ```
   /···/makeBaseApp.pl -t ioc ledDriver
   /···/makeBaseApp.pl -t ioc -i -a RTEMS-uC5282 ledDriver
   ```

2. Edit configure/CONFIG_SITE to enable only the RTEMS-uC5282 target:

   ```
   CROSS_COMPILER_TARGET_ARCHS = RTEMS-uC5282
   ```

3. Edit configure/RELEASE to specify the location of ASYN support:

   ```
   ASYN=/···/modules/soft/asyn
   ```

4. Edit ledDriverApp/src/Makefile:

   - Build only for RTEMS IOC targets (change the PROD_IOC line to PROD_RTEMS):

     ```
     PROD_RTEMS = ledDriver
     ```

   - Add asyn support:

     ```
     ledDriver_DBD += asyn.dbd
     ```

   - Add the asyn library to ledDriver_LIBS (before the EPICS_BASE_IOC_LIBS line which is already there):

     ```
     ledDriver_LIBS += asyn
     ledDriver_LIBS += $(EPICS_BASE_IOC_LIBS)
     ```

   - Add the FPGA device support dbd file (to be written in a following step):

     ```
     ledDriver_DBD += ledDriver.dbd
     ```

   - Add the FPGA device support source file (to be written in a following step):

```
ledDriver_SRCS += ledDriver.c
```

5. Edit ledDriverApp/Db/Makefile and add the line:

```
DB += ledDriver.db
```

6. Create the ledDriverApp/Db/ledDriver.db file referred to in the previous step. The file should contain:

```
record(longout,"leds") {
    field(DTYP,"asynInt32")
    field(OUT,"@asyn(ledDriver 0 0)")
    field(PINI,"YES")
}
```

   The three values in the `OUT` field are:

   (a) The port name.

   (b) The address (unused by this driver, but still needed)

   (c) The timeout value, in seconds (unused by this driver, but still needed).

7. Create ledDriverApp/src/ledDriver.dbd with the contents:

```
registrar(ledDriverRegistrar)
```

8. Create ledDriverApp/src/ledDriver.c. A complete listing of is included in appendix A. Much of this file is common to all ASYN drivers. The following points describe the lines of particular interest to this application.

   12 When the SOPC Builder generates a system it can create a C header file describing the system components. This header file can be included as shown in this comment. Unfortunately the version of Quartus current at the time of writing does not make this process easy so I've just put in my own definition.

   14 The SOPC address space appears in the ColdFire address space at the locations mapped to $\overline{\text{CS1}}$ and $\overline{\text{CS2}}$. The RTEMS board-support package sets up these chip selects at locations 30000000_{16} and 31000000_{16}, respectively.

   15 The '+1' is required because the ColdFire bus is 16-bit big-endian so the least-significant byte on the data bus appears at an odd address.

   21 Catch-all for standard ASYN interfaces.

   61 The line of code that actually performs the output operation.

   70 The line of code that reads back the current setting of the output port. By providing this method, and setting the record PINI field to YES the IOC can reboot and initialize the record to the value currently in the FPGA. This "bumpless' reboot capability makes changes to the IOC code or databases much less intrusive.

   78 More complete device support would probably do nothing more in this routine than register an iocsh command. The actual ASYN registration calls would then be called from the iocsh when invoked by the st.cmd script. This would allow the port name and other parameters to be set from the st.cmd script rather than being burned into the program.

   87 The check for the existence of the I/O port.

   93 The arguments to the registerPort method are:

   (a) The port name.

(b) The port attributes. This driver is is not 'multi device' and does not block.

(c) The autoconnect flag. This driver wants to be automatically reconnected.

(d) The priority of the I/O thread (unused for this driver).

(e) The stack size of the I/O thread (unused for this driver).

98-100 Associate our methods with the appropriate interfaces.

98-100 Associate our methods with the appropriate interfaces.

106 The asynUser structure was used only to provide a place for the initialize() method to return an error message so it can now be freed.

9. Run `make` to compile the application.

10. Use the uCDIMM ColdFire 5282 setenv command to set environment variables as shown below. The exact values will differ as appropriate for your network numbers and NFS server:

```
B$ setenv IPADDR0 www.xxx.yyy.56
B$ setenv HOSTNAME ioccoldfire2
B$ setenv BOOTFILE ucdimm.boot
B$ setenv NAMESERVER www.xxx.yyy.167
B$ setenv NETMASK 255.255.252.0
B$ setenv SERVER www.xxx.yyy.167
B$ setenv NFSMOUNT nfsserver:/export/homes:/home
B$ setenv CMDLINE /···/FPGA_IOC_Example/EPICS/iocBoot/iocledDriver/st.cmd
B$ printenv
FACTORY=Arcturus Networks Inc.
REVISION=uC5282 Rev 1.0 4MB External Flash
SERIAL=X42B20ADC-0130C
CONSOLE=ttyS0
KERNEL=0:linux.bin
KERNEL_ARGS=root=/dev/rom0
HWADDR0=00:06:3B:00:53:0C
FW_VERSION=180001
_0=10000000:400000:RW
RAMIMAGE=yes
IPADDR0=www.xxx.yyy.56
CACHE=on
HOSTNAME=ioccoldfire2
BOOTFILE=ucdimm.boot
NAMESERVER=www.xxx.yyy.167
NETMASK=255.255.252.0
SERVER=www.xxx.yyy.167
NFSMOUNT=nfssrv:/export/homes:/home
CMDLINE=.../FPGA_IOC_Example/EPICS/iocBoot/iocledDriver/st.cmd
```

11. Download and execute the application:

- Start the TFTP server on the ColdFire:

```
B$ tftp
uCTFTP Console 1.0 is running ...
```

- Use the tftp program on your workstation to transfer the executable image to the ColdFire:

```
tftp> binary
tftp> connect www.xxx.yyy.56
tftp> put ledDriver.boot
```

If you have a version of curl which supports the TFTP protocol you can use ie instead:

```
curl -T ledDriver.boot tftp://www.xxx.yyy.56
```

- When the executable image has been transferred press the <ESC> key to the ColdFire to terminate the TFTP server. Use the goram command to start the IOC:

```
Downloading.........................................
B$ goram
Go from RAM!
Go from 0x40000
NTPSERVER environment variable missing -- using www.xxx.yyy.167

***** Initializing network *****
Startup after External reset.
fs1: Ethernet address: 00:06:3b:00:53:0c
***** Initializing NFS *****
This is RTEMS-RPCIOD Release $Name:  $
($Id: tutorial.tex,v 1.30 2008/09/22 13:50:23 norume Exp $)

Till Straumann, Stanford/SLAC/SSRL 2002
See LICENSE file for licensing info
This is RTEMS-NFS $Name:  $
($Id: tutorial.tex,v 1.30 2008/09/22 13:50:23 norume Exp $)

Till Straumann, Stanford/SLAC/SSRL 2002
See LICENSE file for licensing info
Trying to mount www.xxx.yyy.167:/export/homes on /home
***** Initializing NTP *****
***** Starting EPICS application *****
## Example RTEMS startup script
## You may have to change ledDriver to something else
## everywhere it appears in this file
#< envPaths
## Register all support components
dbLoadDatabase("../../dbd/ledDriver.dbd",0,0)
ledDriver_registerRecordDeviceDriver(pdbbase)
## Load record instances
dbLoadRecords("../../db/ledDriver.db","user=norume")
iocInit()
Starting iocInit
############################################################################
###  EPICS IOC CORE built on Jul 25 2005
###  EPICS R3.14.7 $$Name:  $$ $$Date: 2008/09/22 13:50:23 $$
############################################################################
```

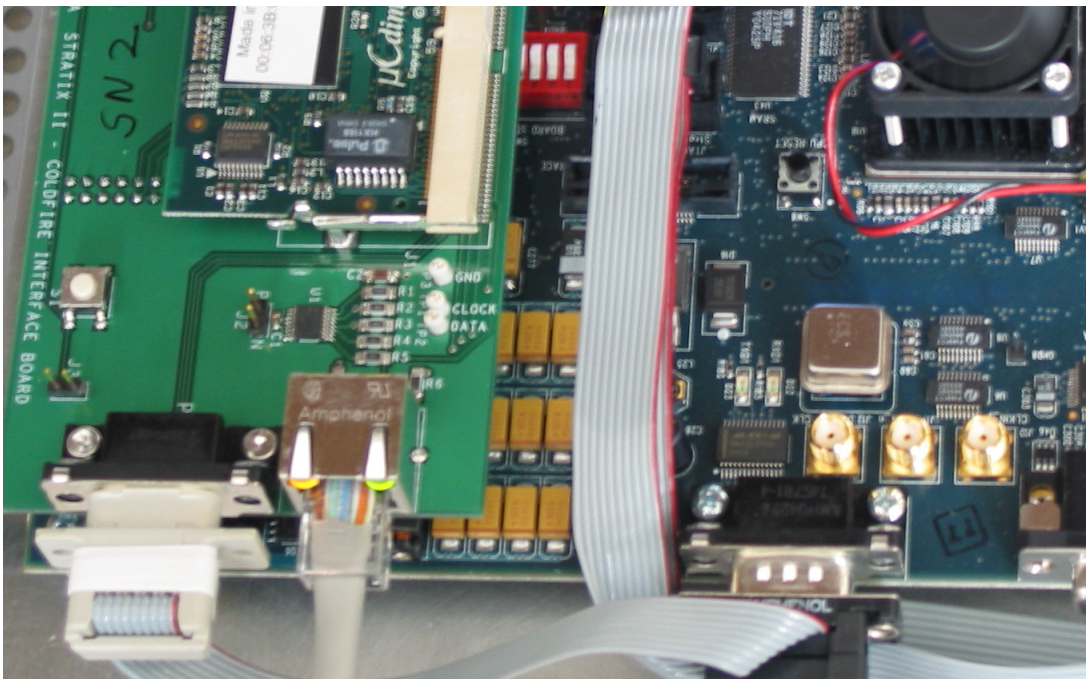
```
iocInit: All initialization complete
## Start any sequence programs
#seq sncledDriver,"user=norume"
ioccoldfire2>
```

Chapter 4

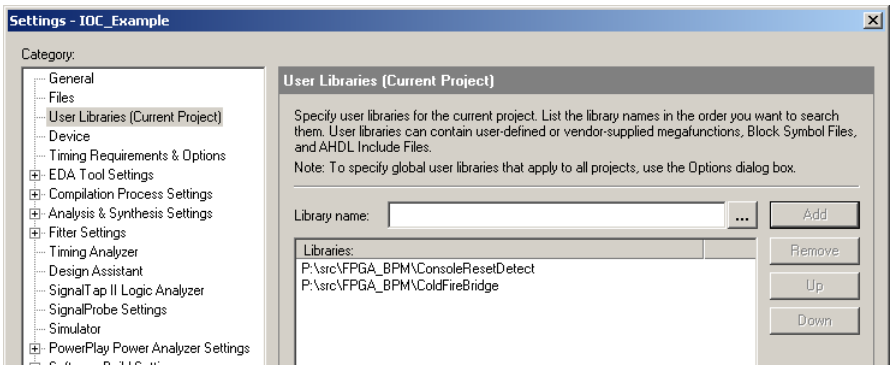
Remote Reset

All VME IOCs at the APS have a card which monitors the console received-data line and generates a system reset when a particular sequence (three consecutive control-X, control-Y or control-Z characters or any consecutive combinations of these characters) of characters is detected. It is quite easy to add this capability to the ColdFire/FPGA IOC.

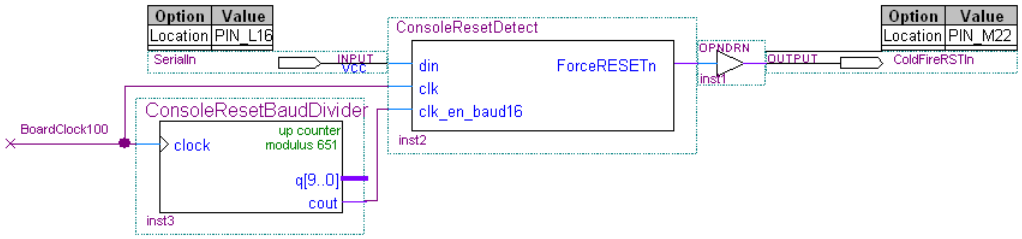
1. Make a special serial line cable which will connect the serial received-data and ground lines of the ColdFire console port to the 9-pin connector on the FPGA development kit. The following picture shows an example. The 9-pin connector plugged in to the FPGA development kit has only two pins (received-data and ground).



2. Add the path to the directory containing the Console Reset Detect component to the list of application user libraries:



3. Add the following components to the application top-level entity (`IOC_Example.bdf`). The ConsoleReset-BaudDivider is a simple modulus-651 counter which sets the serial line speed at 9600 baud ($100000000 \div (9600 \times 16) = 651$). If you're using a different system clock or a different serial line speed you'll have to replace this component with your own counter.



4. Recompile the project and load it into the FPGA. You should now have the ability to remotely reset the ColdFire.

Chapter 5

Remote Reprogramming

This chapter describes the steps which add the ability to reprogram the flash memories on the the ucDIMM ColdFire 5282 module and the Altera FPGA development kits through EPICS channel access. This makes firmware updates of installed systems much easier.

5.1 uCDIMM ColdFire 5282 external flash

The uCDIMM ColdFire 5282 module provides 4 Mbytes of flash memory which can be used to store the RTEMS/EPICS executable and an in-memory file system providing the startup scripts and EPICS '.db' and '.dbd' files. Placing all these files in the flash memory allows the embeded IOC to be truly standalone, cabable of starting and running even if there is no connection to the network. The flash memory can be programmed with the bootstrap 'program' command or, once the steps in this section have been applied, through EPICS channel access. Programming through channel access is convenient since it allows remote updates of multiple IOCs without the need to connect to each IOC console individually.

1. Edit configure/RELEASE to specify the location of the MCF5282 support:

   ```
   MCF5282=/···/modules/instrument/mcf5282
   ```

2. Edit ledDriverApp/src/Makefile:

   - Add mcf5282 support:

     ```
     ledDriver_DBD += drvCFIFlashBurner.dbd
     ```

   - Add the epicsMCF5282 library to `ledDriver_LIBS` (before the `EPICS_BASE_IOC_LIBS` and asyn lines):

     ```
     ledDriver_LIBS += epicsMCF5282
     ledDriver_LIBS += asyn
     ledDriver_LIBS += $(EPICS_BASE_IOC_LIBS)
     ```

3. Edit ledDriverApp/Db/Makefile and add the line:

   ```
   DB_INSTALLS += $(MCF5282)/db/xxdevFlashBurner.db
   ```

4. Edit iocBoot/iocledDriver/st.cmd:

   - Add a line to configure the uCDIMM ColdFire 5282 module flash memory device support. The arguments to this command are the input device name, the base address of the flash memory, a boolean value where 0 means that the flash memory appears directly in the ColdFire address space, and the bit width of the flash memory:

     ```
     drvCFIFlashBurnerConfigure("bootFlash",0x10000000,0,16)
     ```

   - Add a line to load a flash burner record. The `INP` value must match the first argument to the configuration command:

     ```
     dbLoadRecords("db/xxdevFlashBurner.db","P=led:,R=boot:,INP=bootFlash")
     ```

     This will create a process variable named `led:boot:FlashBurner`.

5. Rebuild the application.

When the IOC starts up you should see something like the following when the flash memory burner is configured:

```
drvCFIFlashBurnerConfigure("bootFlash",0x10000000,0,16)
bootFlash: capacity 0x400000
bootFlash: erase block 0 size 0x2000 count 8 base 0
bootFlash: erase block 1 size 0x10000 count 63 base 0x10000
bootFlash: typical sector erase time 1.02 seconds (max 4.1 seconds)
```

To burn a new image into the uCDIMM ColdFire 5282 flash memory module use the `flashBurner` command from the MCF5282 support area. The first argument to the command is the name of the flash burner PV in the ColdFire IOC. The second argument is the name of the file who's contents will be burned into the flash memory: a command like:

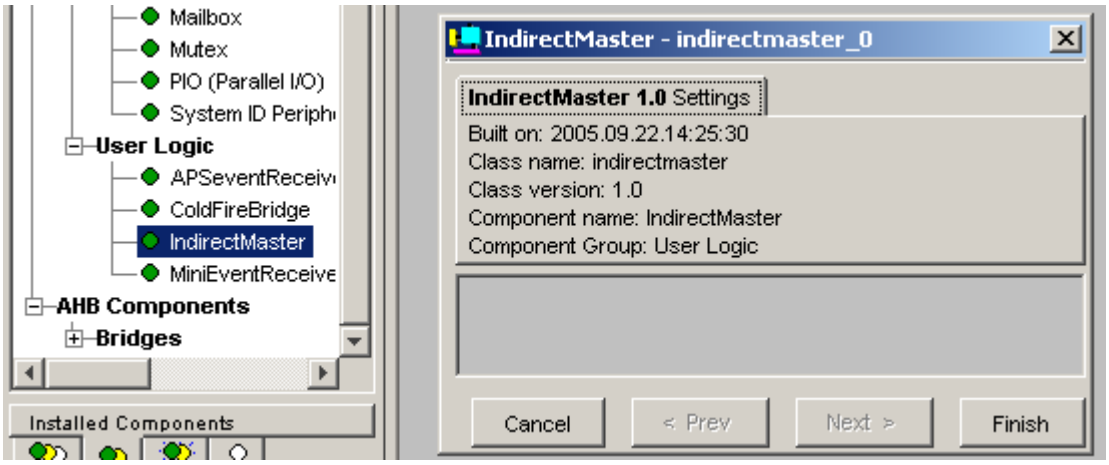
```
/···/modules/instrument/mcf5282/bin/hostArch/flashBurner \
                  led:boot:FlashBurner bin/RTEMS-uC5282/ledDriver.boot
```

5.2 Development kit configuration flash

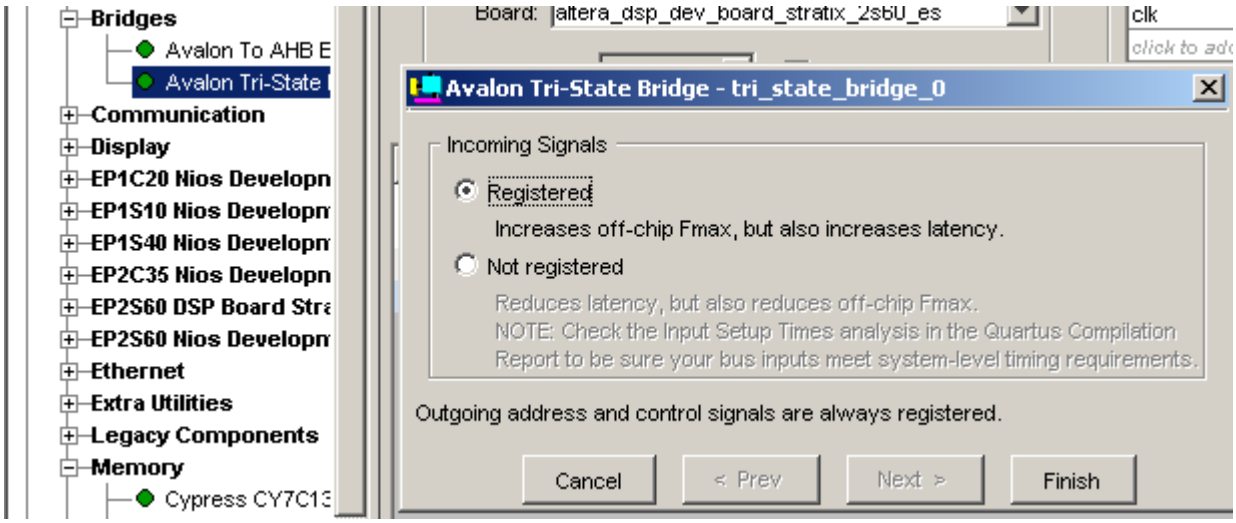
Adding the capability to remotely reprogram the FPGA configuration flash memory requires the following steps. The configuration flash memory can be connected directly to the ColdFire master but that eats up a large portion of the available address space. By using the IndirectMaster component the configuration flash can connected to the ColdFire master without consuming 16 MB of address space.

1. If you have not done so in a previous SOPC project, add the directory containing the Indirect Master SOPC component to the list of directories which the SOPC Builder will search.

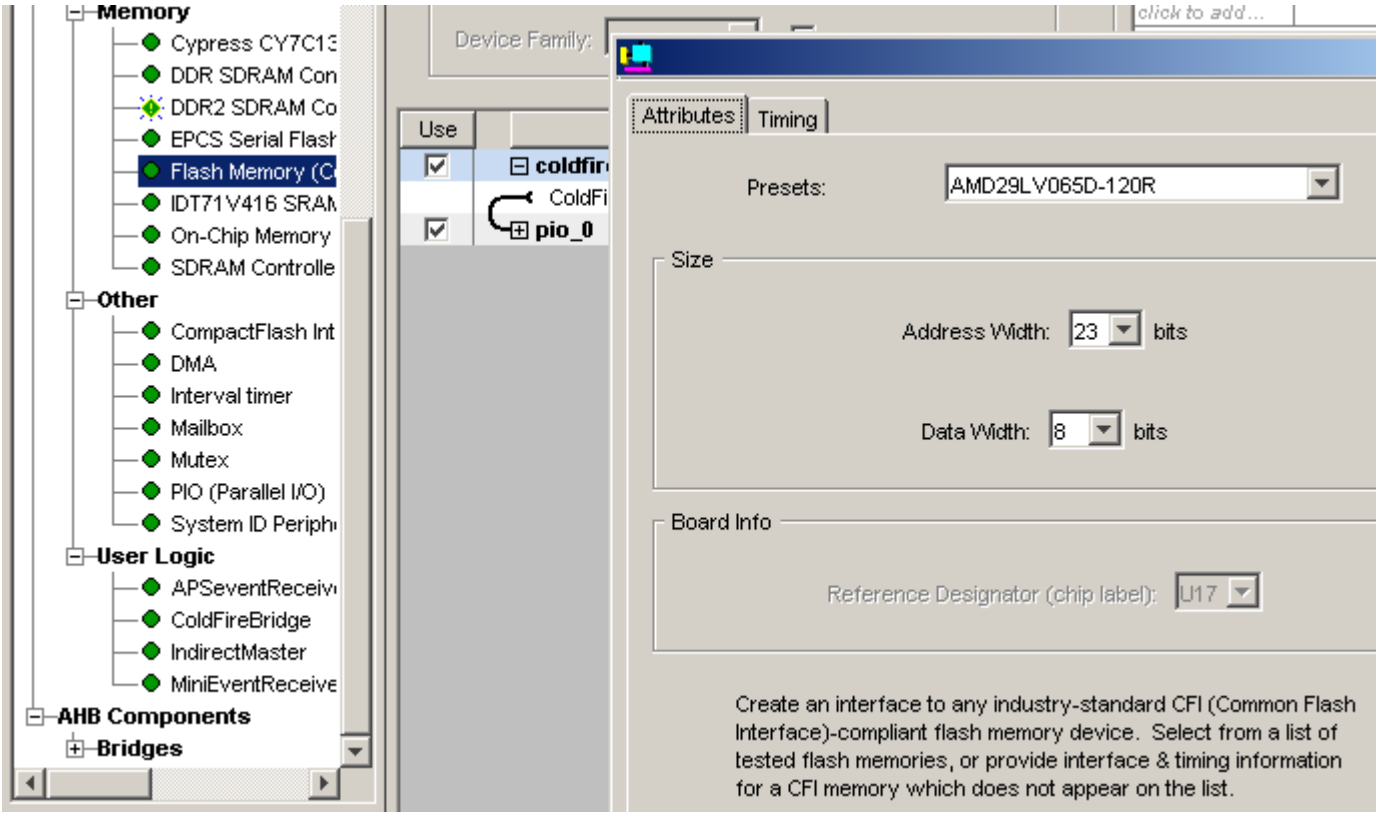
2. Add an IndirectMaster component to the SOPC system:



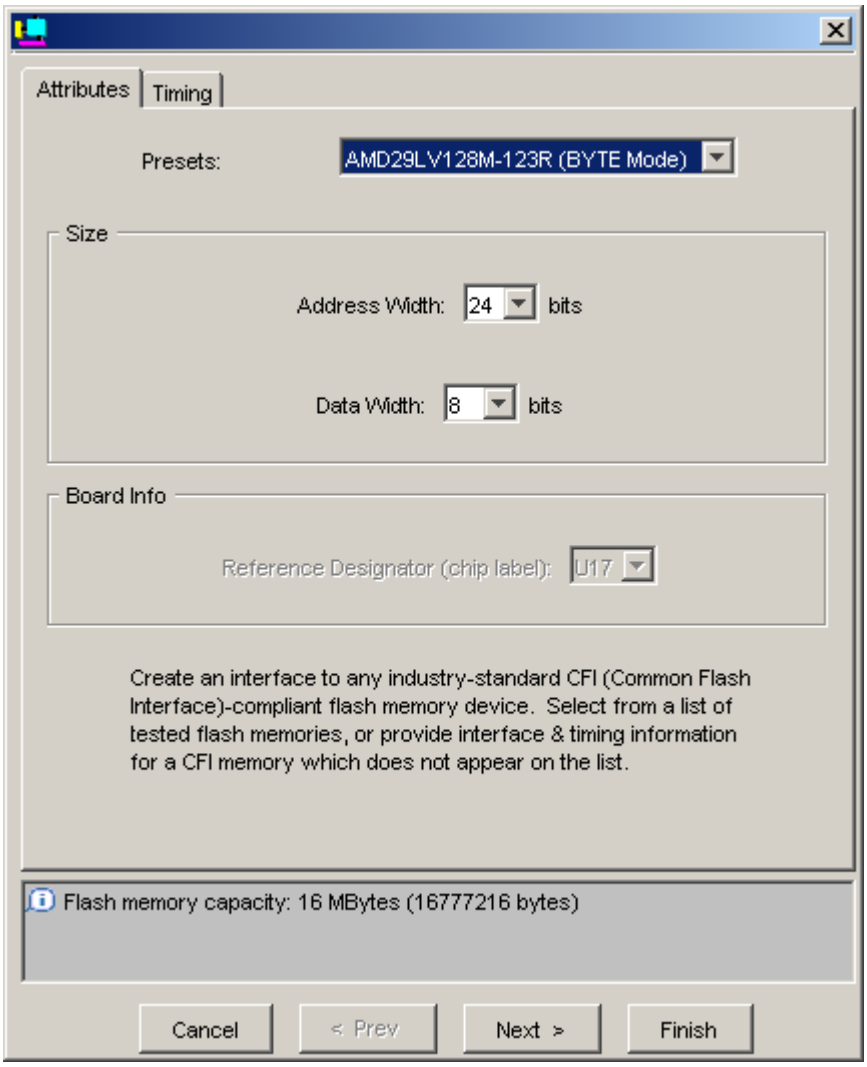
3. Add an Avalon Tri-State Bridge to the system



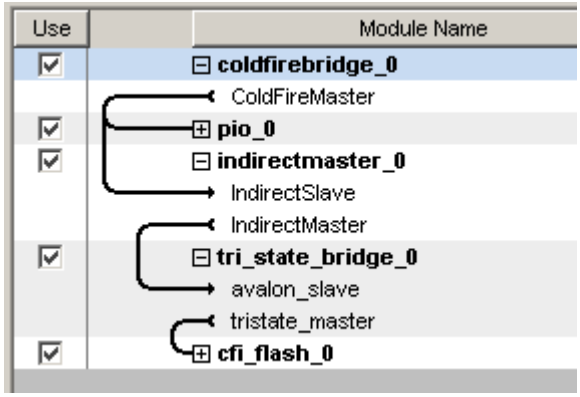
4. Add an Flash Memory (Common Flash Interface) component to the system



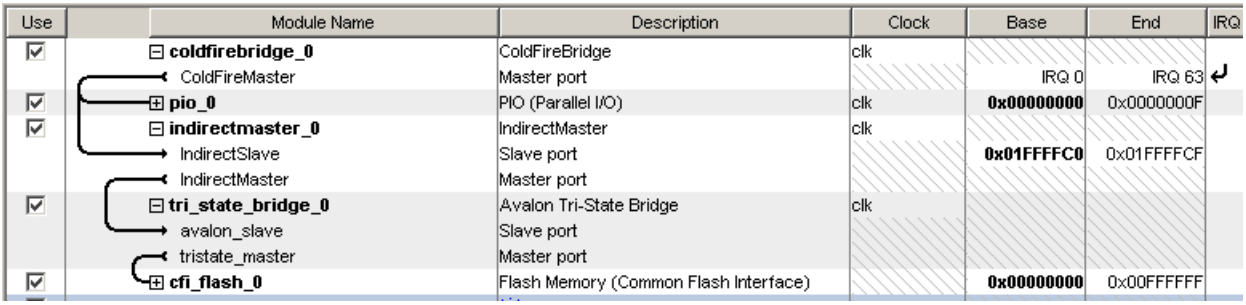
5. Select the Flash Memory configuration that matches the development kit hardware



6. Connect the Avalon Tri-State bridge to the Indirect Master

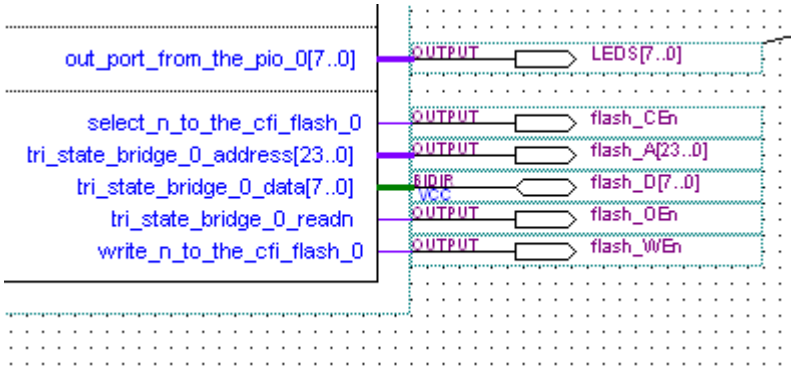


7. Set the IndirectMaster and Flash Memory addresses



Use	Module Name	Description	Clock	Base	End	IRQ
☑	**coldfirebridge_0**	ColdFireBridge	clk			
	ColdFireMaster	Master port		IRQ 0	IRQ 63	
☑	**pio_0**	PIO (Parallel I/O)	clk	**0x00000000**	0x0000000F	
☑	**indirectmaster_0**	IndirectMaster	clk			
	IndirectSlave	Slave port		**0x01FFFFC0**	0x01FFFFCF	
	IndirectMaster	Master port				
☑	**tri_state_bridge_0**	Avalon Tri-State Bridge	clk			
	avalon_slave	Slave port				
	tristate_master	Master port				
☑	**cfi_flash_0**	Flash Memory (Common Flash Interface)		**0x00000000**	0x00FFFFFF	

8. Generate the SOPC system.
9. Update the `IOC_Example` block in the top-level design file. The signals to the flash memory should appear.
10. Add bidir and output pins to the `IOC_Example` flash memory signals.



11. Assign the pin numbers to the flash memory pins. The following table shows the flash memory pin assignments for the Stratix II DSP development kit.

Signal	Pin
flash_CEn	AA32
flash_OEn	AA31
flash_WEn	W32
flash_A[0]	AF30
flash_A[1]	AF29
flash_A[2]	AE30
flash_A[3]	AE29
flash_A[4]	AG32
flash_A[5]	AG31
flash_A[6]	AF32
flash_A[7]	AF31

flash_A[8]	AE32
flash_A[9]	AE31
flash_A[10]	AD32
flash_A[11]	AD31
flash_A[12]	AB28
flash_A[13]	AB27
flash_A[14]	AC32
flash_A[15]	AC31
flash_A[16]	AB30
flash_A[17]	AB29
flash_A[18]	Y29
flash_A[19]	Y28
flash_A[20]	AA30
flash_A[21]	AA29
flash_A[22]	AB32
flash_A[23]	AB31
flash_D[0]	AH30
flash_D[1]	AH29
flash_D[2]	AJ32
flash_D[3]	AJ31
flash_D[4]	AG30
flash_D[5]	AG29
flash_D[6]	AH32
flash_D[7]	AH31

12. Compile the system.
13. Edit iocBoot/iocledDriver/st.cmd:
    - Add a line to configure the FPGA configuration module flash memory device support. The address is that of the IndirectMaster address assigned in the SOPC system plus the offset of the FPGA addresses in the ColdFire memory space. The width must match the development kit hardware flash memory:

      ```
      drvCFIFlashBurnerConfigure("fpgaFlash",0x31ffffc0,1,8)
      ```

    - Add a line to load a flash burner record. The `INP` value must match the first argument to the configuration command:

      ```
      dbLoadRecords("db/xxdevFlashBurner.db","P=led:,R=fpga:,INP=fpgaFlash")
      ```

      This will create a process variable named `led:fpga:FlashBurner`.
14. Rebuild the application.

When the IOC starts up you should see something like the following when the flash memory burner is configured:

```
drvCFIFlashBurnerConfigure("fpgaFlash",0X31FFFFC0,1,8)
fpgaFlash: capacity 0x1000000
fpgaFlash: erase block 0 size 0x10000 count 256 base 0
fpgaFlash: typical sector erase time 1.02 seconds (max 16.4 seconds)
```

To burn a new image into the FPGA configuration flash memory the Quartus '.sof' file must be converted to a '.hexout' file with the `quartus_cpf` program:

```
quartus_cpf -c -a 0x00500000 IOC_Example.sof IOC_Example.hexout
```

The offset value shown above is appropriate for the configuration flash memory on the Stratix II DSP development kit. A value of 0x00800000 should be used for the Stratix II NIOS development kit. The resuling '.hexout' file is burned usingg the `flashBurner` command:

```
/···/modules/instrument/mcf5282/bin/hostArch/flashBurner \
                  led:fpga:FlashBurner IOC_Example.hexout
```

Chapter 6

FPGA Programming Information Process Variable

This chapter describes how to add an EPICS stringin record whose contents contain information about the date and time that the FPGA firmware was compiled.

In SOPC builder create an 64-element 8-bit read-only on-chip memory component. Set the memory to be initialized by `VersionRom.mif`. Connect the component to the ColdFire Master at an address in the VME A16/D16 space (this space begins at SOPC address 0x1ff0000).

Edit the FPGA project configuration script (`IOC_Example.qsf`) and add a precompile script:

```
set_global_assignment -name PRE_FLOW_SCRIPT_FILE "quartus_sh:createVersionRomInitializer.tcl"
```

Place the following into `createVersionRomInitializer.tcl`:

```
proc createFile { name size contents } {
    set fd [open "$name" w 0644]
    puts $fd "-- $name -- $contents --"
    puts $fd "DEPTH = $size;"
    puts $fd "WIDTH = 8;"
    puts $fd "ADDRESS_RADIX = DEC;"
    puts $fd "DATA_RADIX = HEX;"
    puts $fd "CONTENT"
    puts $fd "  BEGIN"
    for {set i 0} {$i < $size} {incr i} {
        if {$i < [string length $contents]} {
            binary scan [string index $contents $i] H* hex
            scan $hex "%x" hex
        } else {
            set hex 0
        }
        puts $fd [format "%6d : %2.2X;" $i $hex]
    }
    puts $fd "  END;"
    close $fd
}

set msg [clock format [clock seconds] -format "%Y-%m-%d %H:%M:%S"]
```

```
createFile "VersionRom.mif" 64 "$msg"
catch { post_message "Created FPGA ROM initialization file." }
```

Of course you can change the `msg` to anything you want. Remember though, that an EPICS stringin record can hold only 40 characters.

Once you've completed the above changes recompile the FPGA project. You should see an additional pass in the progress window and some additional messages in the system status display window.

On the IOC end of things, perform the following steps. Some of these may have already been done earlier, so simply ignore steps that you've already performed.

1. Edit configure/RELEASE to specify the location of the MCF5282 support:

   ```
   MCF5282=/···/modules/instrument/mcf5282
   ```

2. Edit ledDriverApp/src/Makefile:

   - Add FPGA programming information device support:

     ```
     ledDriver_DBD += fpgaProgrammingInfo.dbd
     ```

   - Add the epicsMCF5282 library to `ledDriver_LIBS` (before the `EPICS_BASE_IOC_LIBS` and asyn lines):

     ```
     ledDriver_LIBS += epicsMCF5282
     ledDriver_LIBS += asyn
     ledDriver_LIBS += $(EPICS_BASE_IOC_LIBS)
     ```

3. Edit ledDriverApp/Db/Makefile and add the line:

   ```
   DB_INSTALLS += $(MCF5282)/db/fpgaProgrammingInfo.db
   ```

4. Edit iocBoot/iocledDriver/st.cmd:

   - Add a line to configure the uCDIMM FPGA programming information device support. The arguments to this command are the input device name, the base address of the information memory and the length of the information memory (by default 64 bytes):

     ```
     devFpgaInfoConfigure("fpgaInfo",0x3800)
     ```

   - Add a line to load an FPGA programming information record. The `PORT` value must match the first argument to the configuration command:

     ```
     dbLoadRecords("db/fpgaProgrammingInfo.db","P=$(P),R=,PORT=fpgaInfo")
     ```

     This will create a process variable named `led:FPGACompileTimeSI`.

5. Rebuild the application.

Appendix A

ledDriver.c

```
1  /*
2   * ASYN Int32 driver for simple FPGA application
3   */
4  #include <epicsStdio.h>
5  #include <epicsExport.h>
6  #include <cantProceed.h>
7
8  #include <asynDriver.h>
9  #include <asynStandardInterfaces.h>
10 #include <devLib.h>
11
12 /*#include "SOPC.h"   /* Symbolic link to SOPC-generated system header file */
13 #define PIO_BASE 0x000000
14 #define AVALON_BASE  0x30000000    /* Base of Avalon space in ColdFire space */
15 #define OPTR ((epicsUInt8 *)(AVALON_BASE+PIO_BASE+1))
16
17 /*
18  * Driver private storage
19  */
20 typedef struct drvPvt {
21     asynStandardInterfaces asynStandardInterfaces;
22     const char          *portName;
23     volatile epicsUInt8 *optr;
24 } drvPvt;
25
26 /*
27  * asynCommon methods
28  */
29 static void
30 report(void *pvt, FILE *fp, int details)
31 {
32     drvPvt *pdrvPvt = (drvPvt *)pvt;
33
```

```
34      if (details)
35          fprintf(fp, "%s: %#x@%p\n", pdrvPvt->portName, *pdrvPvt->optr, pdrvPvt->optr);
36  }
37
38  static asynStatus
39  connect(void *pvt, asynUser *pasynUser)
40  {
41      pasynManager->exceptionConnect(pasynUser);
42      return asynSuccess;
43  }
44
45  static asynStatus
46  disconnect(void *pvt, asynUser *pasynUser)
47  {
48      pasynManager->exceptionDisconnect(pasynUser);
49      return asynSuccess;
50  }
51  static asynCommon commonMethods = { report, connect, disconnect };
52
53  /*
54   * asynInt32 methods
55   */
56  static asynStatus
57  int32Write(void *pvt, asynUser *pasynUser, epicsInt32 value)
58  {
59      drvPvt *pdrvPvt = (drvPvt *)pvt;
60
61      *pdrvPvt->optr = value;
62      return asynSuccess;
63  }
64
65  static asynStatus
66  int32Read(void *pvt, asynUser *pasynUser, epicsInt32 *value)
67  {
68      drvPvt *pdrvPvt = (drvPvt *)pvt;
69
70      *value = *pdrvPvt->optr;
71      return asynSuccess;
72  }
73  static asynInt32 int32Methods = { int32Write, int32Read };
74
75  /*
76   * Register ourself with ASYN
77   */
78  static void ledDriverRegistrar(void)
79  {
80      drvPvt *pdrvPvt;
81      asynStatus status;
```

```
82      const char *portName = "ledDriver";
83      asynUser *pasynUser = pasynManager->createAsynUser(0, 0);
84      epicsUInt8 dummy;
85
86      pdrvPvt = callocMustSucceed(sizeof(drvPvt), 1, portName);
87      if (devWriteProbe(sizeof(dummy), pdrvPvt->optr, &dummy) != 0) {
88          printf("ledDriver: memory probe failed\n");
89          return;
90      }
91      pdrvPvt->portName = portName;
92      pdrvPvt->optr = OPTR;
93      status = pasynManager->registerPort(portName, 0, 1, 0, 0);
94      if(status != asynSuccess) {
95          printf("registerDriver failed\n");
96          return;
97      }
98      pdrvPvt->asynStandardInterfaces.common.pinterface  = &commonMethods;
99      pdrvPvt->asynStandardInterfaces.int32.pinterface  = &int32Methods;
100     status = pasynStandardInterfacesBase->initialize(portName,
101                         &pdrvPvt->asynStandardInterfaces, pasynUser, pdrvPvt);
102     if (status != asynSuccess) {
103         printf("Can't register interfaces: %s\n", pasynUser->errorMessage);
104         return;
105     }
106     pasynManager->freeAsynUser(pasynUser);
107 }
108 epicsExportRegistrar(ledDriverRegistrar);
```

Appendix B

Alternate Pinouts

The information in this tutorial applies to several Altera development kits, including:

1. Stratix II DSP development kit(EP2S60), 100 MHz clock.
2. "Old" (Non-PMC) Stratix II NIOS development kit (EP2S60), "BoardClock100" input is a 50 MHz clock. The Compact Flash socket on this kit shares many signals with the expansion headers to which the ColdFire module is connected. Remove the Compact Flash card from the socket before connecting the ColdFire module to the development kit.
3. Cyclone II NIOS development kit (EP2C35), "BoardClock100" input is a 50 MHz clock, expansion prototype connectors require extensions to clear tall components on development kit.
4. "New" (PMC) Stratix II NIOS development kit (EP2S60), "BoardClock100" input is a 50 MHz clock. The Compact Flash socket on this kit shares many signals with the expansion headers to which the ColdFire module is connected. Remove the Compact Flash card from the socket before connecting the ColdFire module to the development kit.

The pin assignments for each of these kits are presented in the following table.

Signal		1	2	3	4
BoardClock100		A16	AF15	N2	B13
ColdFireA[1]	Proto1_IO1	R30	J9	F24	C2
ColdFireA[2]	Proto1_IO2	P32	F8	F23	D2
ColdFireA[3]	Proto1_IO3	P31	A3	J21	D3
ColdFireA[4]	Proto1_IO4	M32	C4	J20	E1
ColdFireA[5]	Proto1_IO5	M31	C3	F25	E2
ColdFireA[6]	Proto1_IO6	N31	C5	F26	E3
ColdFireA[7]	Proto1_IO7	N30	K10	N18	E4
ColdFireA[8]	Proto1_IO8	L32	H9	P18	F1
ColdFireA[9]	Proto1_IO9	L31	G9	G23	F2
ColdFireA[10]	Proto1_IO10	M30	A5	G24	F3
ColdFireA[11]	Proto1_IO11	M29	B5	G25	F4
ColdFireA[12]	Proto1_IO12	N29	D6	G26	G3

ColdFireA[13]	Proto1_IO13	N28	A6	H23	G4
ColdFireA[14]	Proto1_IO14	L30	H10	H24	H3
ColdFireA[15]	Proto1_IO15	L29	K11	J23	H4
ColdFireA[16]	Proto1_IO16	K32	F10	J24	J3
ColdFireA[17]	Proto1_IO17	K31	A7	H25	J4
ColdFireA[18]	Proto1_IO18	K30	C7	H26	G1
ColdFireA[19]	Proto1_IO19	K29	D7	K18	G2
ColdFireA[20]	Proto1_IO20	J32	A8	K19	H1
ColdFireA[21]	Proto1_IO21	J31	G10	K23	H2
ColdFireA[22]	Proto1_IO22	H32	J11	K24	K3
ColdFireA[23]	Proto1_IO23	H31	F11	J25	K4
ColdFireBS2n	Proto2_IO23	U27	A24	AD19	AC3
ColdFireBS3n	Proto2_IO24	U23	B24	AC19	AD1
ColdFireCLKOUT	Proto1_CLKOUT	T32	AC14	N26	R26
ColdFireCS1n	Proto2_IO20	V29	D18	AC20	AB3
ColdFireCS2n	Proto2_IO19	V23	J18	AB20	AA4
ColdFireD[16]	Proto2_IO0	AC27	H15	AE24[1]	T2
ColdFireD[17]	Proto2_IO1	AC26	J15	T21	T3
ColdFireD[18]	Proto2_IO2	AD27	C16	V22	U1
ColdFireD[19]	Proto2_IO3	AD26	A17	AF23	U2
ColdFireD[20]	Proto2_IO4	Y23	C17	AE23	V1
ColdFireD[21]	Proto2_IO5	Y22	A18	AC22	V2
ColdFireD[22]	Proto2_IO6	Y25	F17	AB21	W1
ColdFireD[23]	Proto2_IO7	Y24	K16	AD23	W2
ColdFireD[24]	Proto2_IO8	AA27	G17	AD22	Y1
ColdFireD[25]	Proto2_IO9	AA26	A19	AC21	Y2
ColdFireD[26]	Proto2_IO10	Y27	C18	AD21	AA1
ColdFireD[27]	Proto2_IO11	Y26	C19	AF22	AA2
ColdFireD[28]	Proto2_IO12	W25	A20	AE22	AB1
ColdFireD[29]	Proto2_IO13	W24	J17	V18	AB2
ColdFireD[30]	Proto2_IO14	W27	A21	W19	W3
ColdFireD[31]	Proto2_IO15	W26	C20	U17	W4
ColdFireIRQ1n	Proto2_IO17	W28	A22	AF21	Y4
ColdFireOEn	Proto2_IO25	U22	K17	AA17	AD2
ColdFireRSTIn	Proto2_IO28	M22	H18	V17	W10
ColdFireRSTOn	Proto2_IO27	M23	J14	W17	W9
ColdFireRpWn	Proto2_IO22	U28	B22	AE20	AC2
ColdFireTAn	Proto2_IO16	W29	C21	U18	Y3
ColdFireTEAn	Proto2_IO26	L25	H17	AA18	Y7
ColdFireTSn	Proto2_IO21	V28	C22	AF20	AB4
LEDS[0]		B4	AD26	AC10	W15
LEDS[1]		D5	AD25	W11	V14
LEDS[2]		E5	AC25	W12	AD17

LEDS[3]		A4	AC24	AE8	AA17
LEDS[4]		A5	AB24	AF8	V16
LEDS[5]		D6	AB23	AE7	AB17
LEDS[6]		C6	AB26	AF7	AD18
LEDS[7]		A6	AB25	AA11	V17
SerialIn		L16	H7	AB15	AD26

1. Pin AE24 on the Cyclone II kit is a dual-purpose pin and needs to be configured as Assignments→Device..., Device & Pin Options, Dual-Purpose Pins tab): Change nCEO from "Use as programming pin" to "Use as regular IO".