How to create a simple ColdFire and Altera FPGA IOC

W. Eric Norum

September 22, 2008
### Contents

1 Introduction ........................................ 1

2 FPGA application ..................................... 3

3 EPICS application .................................... 11

4 Remote Reset ........................................ 17

5 Remote Reprogramming ............................... 19
   5.1 uCDIMM ColdFire 5282 external flash .......... 19
   5.2 Development kit configuration flash .......... 20

6 FPGA Programming Information Process Variable 27

A ledDriver.c ......................................... 29

B Alternate Pinouts .................................... 33
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 output 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.
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 CS1 and 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.

ICMS: APS_1189963  12
(b) The port attributes. This driver 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:

```
$ 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: $

Till Straumann, Stanford/SLAC/SSRL 2002
See LICENSE file for licensing info
This is RTEMS-NFS $Name: $

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("..../db/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 embedded IOC to be truly standalone, capable 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:
CHAPTER 5. REMOTE REPROGRAMMING

5.2. DEVELOPMENT KIT CONFIGURATION FLASH

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

<table>
<thead>
<tr>
<th>Use</th>
<th>Module Name</th>
<th>Description</th>
<th>Clock</th>
<th>Base</th>
<th>End</th>
<th>IRQ</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ColdFireBridge_0</td>
<td>Master port</td>
<td>clk</td>
<td>IRQ 0</td>
<td>Reader</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pio_0</td>
<td>Parallel I/O Master port</td>
<td>clk</td>
<td>0x00000000</td>
<td>0x000000FF</td>
<td></td>
</tr>
<tr>
<td></td>
<td>indirect Slave_0</td>
<td>IndirectMaster Slave port</td>
<td>clk</td>
<td>0x00000000</td>
<td>0x000000FF</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tri_state bridge_0</td>
<td>Avalon Tri-State Bridge</td>
<td>clk</td>
<td>0x00000000</td>
<td>0x000000FF</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tri_state slave</td>
<td>Slave port</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>tri_state master</td>
<td>Master port</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cfi flash_0</td>
<td>Flash Memory (Common Flash Interface)</td>
<td></td>
<td>0x00000000</td>
<td>0x000000FF</td>
<td></td>
</tr>
</tbody>
</table>

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.

```
cut_port_from_the_pio_0[7..0]
select_n_to_the_cfi_flash_0
tri_state_bridge_0_address[23..0]
tri_state_bridge_0_data[7..0]
tri_state_bridge_0_readn
write_n_to_the_cfi_flash_0
```

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.

<table>
<thead>
<tr>
<th>Signal</th>
<th>Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>flash_CEn</td>
<td>AA32</td>
</tr>
<tr>
<td>flash_OEn</td>
<td>AA31</td>
</tr>
<tr>
<td>flash_WEn</td>
<td>W32</td>
</tr>
<tr>
<td>flash_A[0]</td>
<td>AF30</td>
</tr>
<tr>
<td>flash_A[1]</td>
<td>AF29</td>
</tr>
<tr>
<td>flash_A[6]</td>
<td>AF32</td>
</tr>
<tr>
<td>flash_A[7]</td>
<td>AF31</td>
</tr>
</tbody>
</table>
### 5.2. DEVELOPMENT KIT CONFIGURATION FLASH

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>flash_A[8]</td>
<td>AE32</td>
</tr>
<tr>
<td>flash_A[9]</td>
<td>AE31</td>
</tr>
<tr>
<td>flash_A[10]</td>
<td>AD32</td>
</tr>
<tr>
<td>flash_A[12]</td>
<td>AB28</td>
</tr>
<tr>
<td>flash_A[14]</td>
<td>AC32</td>
</tr>
<tr>
<td>flash_A[15]</td>
<td>AC31</td>
</tr>
<tr>
<td>flash_A[16]</td>
<td>AB30</td>
</tr>
<tr>
<td>flash_A[17]</td>
<td>AB29</td>
</tr>
<tr>
<td>flash_A[18]</td>
<td>Y29</td>
</tr>
<tr>
<td>flash_A[19]</td>
<td>Y28</td>
</tr>
<tr>
<td>flash_A[20]</td>
<td>AA30</td>
</tr>
<tr>
<td>flash_A[21]</td>
<td>AA29</td>
</tr>
<tr>
<td>flash_A[22]</td>
<td>AB32</td>
</tr>
<tr>
<td>flash_A[23]</td>
<td>AB31</td>
</tr>
<tr>
<td>flash_D[0]</td>
<td>AH30</td>
</tr>
<tr>
<td>flash_D[1]</td>
<td>AH29</td>
</tr>
<tr>
<td>flash_D[3]</td>
<td>AJ31</td>
</tr>
<tr>
<td>flash_D[4]</td>
<td>AG30</td>
</tr>
<tr>
<td>flash_D[5]</td>
<td>AG29</td>
</tr>
<tr>
<td>flash_D[6]</td>
<td>AH32</td>
</tr>
<tr>
<td>flash_D[7]</td>
<td>AH31</td>
</tr>
</tbody>
</table>

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 resulting '.hexout' file is burned using 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"]
```

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 \texttt{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 \texttt{configure/RELEASE} to specify the location of the MCF5282 support:

   \begin{verbatim}
   MCF5282=\ldots\textbackslash modules\textbackslash instrument\textbackslash mcf5282
   \end{verbatim}

2. Edit \texttt{ledDriverApp/src/Makefile}:

   - Add FPGA programming information device support:
     \begin{verbatim}
     ledDriver_DBD += fpgaProgrammingInfo.dbd
     \end{verbatim}

   - Add the epicsMCF5282 library to \texttt{ledDriver_LIBS} (before the \texttt{EPICS_BASE_IOC_LIBS} and \texttt{asyn} lines):
     \begin{verbatim}
     ledDriver_LIBS += epicsMCF5282
     ledDriver_LIBS += asyn
     ledDriver_LIBS += $(EPICS_BASE_IOC_LIBS)
     \end{verbatim}

3. Edit \texttt{ledDriverApp/Db/Makefile} and add the line:

   \begin{verbatim}
   DB_INSTALLS += $(MCF5282)/db/fpgaProgrammingInfo.db
   \end{verbatim}

4. Edit \texttt{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):
     \begin{verbatim}
     devFpgaInfoConfigure("fpgaInfo",0x3800)
     \end{verbatim}

   - Add a line to load an FPGA programming information record. The \texttt{PORT} value must match the first argument to the configuration command:
     \begin{verbatim}
     dbLoadRecords("db/fpgaProgrammingInfo.db","P=$(P),R=,PORT=fpgaInfo")
     \end{verbatim}

     This will create a process variable named \texttt{led:FPGACompileTimeSI}.

5. Rebuild the application.
Appendix A

ledDriver.c

/*
* ASYN Int32 driver for simple FPGA application
*/
#include <epicsStdio.h>
#include <epicsExport.h>
#include <cantProceed.h>
#include <asynDriver.h>
#include <asynStandardInterfaces.h>
#include <devLib.h>

#include "SOPC.h" /* Symbolic link to SOPC-generated system header file */
#define PIO_BASE 0x000000
#define AVALON_BASE 0x30000000 /* Base of Avalon space in ColdFire space */
#define OPTR ((epicsUInt8 *)(AVALON_BASE+PIO_BASE+1))

/*
* Driver private storage
*/
typedef struct drvPvt {
    asynStandardInterfaces asynStandardInterfaces;
    const char *portName;
    volatile epicsUInt8 *optr;
} drvPvt;

/*
* asynCommon methods
*/
static void report(void *pvt, FILE *fp, int details)
{
    drvPvt *drvPvt = (drvPvt *)pvt;
if (details)
    fprintf(fp, "%s: %x@%p\n", pdrvPvt->portName, *pdrvPvt->optr, pdrvPvt->ostr);
}

static asynStatus
connect(void *pvt, asynUser *pasynUser)
{
    pasynManager->exceptionConnect(pasynUser);
    return asynSuccess;
}

static asynStatus
disconnect(void *pvt, asynUser *pasynUser)
{
    pasynManager->exceptionDisconnect(pasynUser);
    return asynSuccess;
}

static asynCommon commonMethods = { report, connect, disconnect };

/*
 * asynInt32 methods
 */
static asynStatus
int32Write(void *pvt, asynUser *pasynUser, epicsInt32 value)
{
    drvPvt *pdrvPvt = (drvPvt *)pvt;
    *pdrvPvt->ostr = value;
    return asynSuccess;
}

static asynStatus
int32Read(void *pvt, asynUser *pasynUser, epicsInt32 *value)
{
    drvPvt *pdrvPvt = (drvPvt *)pvt;
    *value = *pdrvPvt->ostr;
    return asynSuccess;
}

static asynInt32 int32Methods = { int32Write, int32Read };

/*
 * Register ourself with ASYN
 */
static void ledDriverRegistrar(void)
{
    drvPvt *pdrvPvt;
    asynStatus status;

const char *portName = "ledDriver";
asynUser *pasynUser = pasynManager->createAsynUser(0, 0);
epicsUInt8 dummy;

pdrvPvt = callocMustSucceed(sizeof(drvPvt), 1, portName);
if (devWriteProbe(sizeof(dummy), pdrvPvt->optr, &dummy) != 0) {
    printf("ledDriver: memory probe failed\n");
    return;
}
pdrvPvt->portName = portName;
pdrvPvt->optr = OPTR;
status = pasynManager->registerPort(portName, 0, 1, 0, 0);
if(status != asynSuccess) {
    printf("registerDriver failed\n");
    return;
}
pdrvPvt->asynStandardInterfaces.common.pinterface = &commonMethods;
pdrvPvt->asynStandardInterfaces.int32.pinterface = &int32Methods;
status = pasynStandardInterfacesBase->initialize(portName,
    &pdrvPvt->asynStandardInterfaces, pasynUser, pdrvPvt);
if (status != asynSuccess) {
    printf("Can't register interfaces: %s\n", pasynUser->errorMessage);
    return;
}
pasynManager->freeAsynUser(pasynUser);
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.

<table>
<thead>
<tr>
<th>Signal</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>BoardClock100</td>
<td>A16</td>
<td>A15</td>
<td>N2</td>
<td>B13</td>
</tr>
<tr>
<td>ColdFireA[8]</td>
<td>L32</td>
<td>H9</td>
<td>P18</td>
<td>F1</td>
</tr>
<tr>
<td>Device</td>
<td>Pin</td>
<td>Value</td>
<td>Symbol</td>
<td>Value</td>
</tr>
<tr>
<td>---------------</td>
<td>------</td>
<td>-------</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>ColdFireA[18]</td>
<td>Proto1_IO18</td>
<td>K30</td>
<td>C7</td>
<td>H26</td>
</tr>
<tr>
<td>ColdFireA[19]</td>
<td>Proto1_IO19</td>
<td>K29</td>
<td>D7</td>
<td>K18</td>
</tr>
<tr>
<td>ColdFireBS2n</td>
<td>Proto2_IO23</td>
<td>U27</td>
<td>A24</td>
<td>AD19</td>
</tr>
<tr>
<td>ColdFireBS3n</td>
<td>Proto2_IO24</td>
<td>U23</td>
<td>B24</td>
<td>AC19</td>
</tr>
<tr>
<td>ColdFireCLKOUT</td>
<td>Proto1_CLKOUT</td>
<td>T32</td>
<td>AC14</td>
<td>N26</td>
</tr>
<tr>
<td>ColdFireCS1n</td>
<td>Proto2_IO20</td>
<td>V29</td>
<td>D18</td>
<td>AC20</td>
</tr>
<tr>
<td>ColdFireCS2n</td>
<td>Proto2_IO19</td>
<td>V23</td>
<td>J18</td>
<td>AB20</td>
</tr>
<tr>
<td>ColdFireD[16]</td>
<td>Proto2_IO10</td>
<td>AC27</td>
<td>H15</td>
<td>AE241</td>
</tr>
<tr>
<td>ColdFireD[17]</td>
<td>Proto2_IO1</td>
<td>AC26</td>
<td>J15</td>
<td>T21</td>
</tr>
<tr>
<td>ColdFireD[18]</td>
<td>Proto2_IO2</td>
<td>AD27</td>
<td>C16</td>
<td>V22</td>
</tr>
<tr>
<td>ColdFireD[19]</td>
<td>Proto2_IO3</td>
<td>AD26</td>
<td>A17</td>
<td>AF23</td>
</tr>
<tr>
<td>ColdFireD[20]</td>
<td>Proto2_IO4</td>
<td>Y23</td>
<td>C17</td>
<td>AE23</td>
</tr>
<tr>
<td>ColdFireD[21]</td>
<td>Proto2_IO5</td>
<td>Y22</td>
<td>A18</td>
<td>AC22</td>
</tr>
<tr>
<td>ColdFireD[22]</td>
<td>Proto2_IO6</td>
<td>Y25</td>
<td>F17</td>
<td>AB21</td>
</tr>
<tr>
<td>ColdFireD[23]</td>
<td>Proto2_IO7</td>
<td>Y24</td>
<td>K16</td>
<td>AD23</td>
</tr>
<tr>
<td>ColdFireD[24]</td>
<td>Proto2_IO8</td>
<td>AA27</td>
<td>G17</td>
<td>AD22</td>
</tr>
<tr>
<td>ColdFireD[25]</td>
<td>Proto2_IO9</td>
<td>AA26</td>
<td>A19</td>
<td>AC21</td>
</tr>
<tr>
<td>ColdFireD[26]</td>
<td>Proto2_IO10</td>
<td>Y27</td>
<td>C18</td>
<td>AD21</td>
</tr>
<tr>
<td>ColdFireD[27]</td>
<td>Proto2_IO11</td>
<td>Y26</td>
<td>C19</td>
<td>AF22</td>
</tr>
<tr>
<td>ColdFireD[28]</td>
<td>Proto2_IO12</td>
<td>W25</td>
<td>A20</td>
<td>AE22</td>
</tr>
<tr>
<td>ColdFireD[29]</td>
<td>Proto2_IO13</td>
<td>W24</td>
<td>J17</td>
<td>V18</td>
</tr>
<tr>
<td>ColdFireD[31]</td>
<td>Proto2_IO15</td>
<td>W26</td>
<td>C20</td>
<td>U17</td>
</tr>
<tr>
<td>ColdFireIRQ1n</td>
<td>Proto2_IO17</td>
<td>W28</td>
<td>A22</td>
<td>AF21</td>
</tr>
<tr>
<td>ColdFireOEn</td>
<td>Proto2_IO25</td>
<td>U22</td>
<td>K17</td>
<td>AA17</td>
</tr>
<tr>
<td>ColdFireRSTIn</td>
<td>Proto2_IO28</td>
<td>M22</td>
<td>H18</td>
<td>V17</td>
</tr>
<tr>
<td>ColdFireRSTOn</td>
<td>Proto2_IO27</td>
<td>M23</td>
<td>J14</td>
<td>W17</td>
</tr>
<tr>
<td>ColdFireRpWn</td>
<td>Proto2_IO22</td>
<td>U28</td>
<td>B22</td>
<td>AE20</td>
</tr>
<tr>
<td>ColdFireTan</td>
<td>Proto2_IO16</td>
<td>W29</td>
<td>C21</td>
<td>U18</td>
</tr>
<tr>
<td>ColdFireTEAn</td>
<td>Proto2_IO26</td>
<td>L25</td>
<td>H17</td>
<td>AA18</td>
</tr>
<tr>
<td>ColdFireTSn</td>
<td>Proto2_IO21</td>
<td>V28</td>
<td>C22</td>
<td>AF20</td>
</tr>
<tr>
<td>LEDS[0]</td>
<td></td>
<td>B4</td>
<td>AD26</td>
<td>AC10</td>
</tr>
<tr>
<td>LEDS[1]</td>
<td></td>
<td>D5</td>
<td>AD25</td>
<td>W11</td>
</tr>
<tr>
<td>LEDS[2]</td>
<td></td>
<td>E5</td>
<td>AC25</td>
<td>W12</td>
</tr>
</tbody>
</table>
### APPENDIX B. ALTERNATE PINOUTS

<table>
<thead>
<tr>
<th>LEDs[3]</th>
<th>A4</th>
<th>AC24</th>
<th>AE8</th>
<th>AA17</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEDs[4]</td>
<td>A5</td>
<td>AB24</td>
<td>AF8</td>
<td>V16</td>
</tr>
<tr>
<td>LEDs[5]</td>
<td>D6</td>
<td>AB23</td>
<td>AE7</td>
<td>AB17</td>
</tr>
<tr>
<td>LEDs[6]</td>
<td>C6</td>
<td>AB26</td>
<td>AF7</td>
<td>AD18</td>
</tr>
<tr>
<td>SerialIn</td>
<td>L16</td>
<td>H7</td>
<td>AB15</td>
<td>AD26</td>
</tr>
</tbody>
</table>

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”.