Activity: New Target

In this activity, you will create a definition for a second target that results in a second cFS application executable. (Admittedly, cross-compiling on a slower platform adds unnecessary complexity. But, the point is to show how to build a cross-compiled application.)

# Create an Ubuntu Virtual Machine

|  |  |  |
| --- | --- | --- |
| 1 | Using Virtual Box, create an Ubuntu virtual machine. (I’m starting with 18.04.)  You can also create a docker container, if you prefer. |  |
| 2 | Install prerequisites. | jbrandenburg@cfs-cross-compile:~$ sudo apt-get update  <snip>  jbrandenburg@cfs-cross-compile:~$ sudo apt-get upgrade  <snip>  jbrandenburg@cfs-cross-compile:~$ sudo apt-get install build-essential  <snip>  jbrandenburg@cfs-cross-compile:~$ sudo apt-get install git bison flex texinfo python libz-dev libncurses5-dev python2.7-dev libpython2.7-dev  <snip> |

# Build RTEMS

|  |  |  |
| --- | --- | --- |
| 1 | Obtain the RTEMS source builder for RTEMS 4.11.3 (this version of cFS has configuration files for RTEMS 4.11). | debian@beaglebone:~$ sudo mkdir -p /opt/rtems-4.11  debian@beaglebone:~$ git clone git://git.rtems.org/rtems-source-builder.git  Cloning into 'rtems-source-builder'...  remote: Counting objects: 10909, done.  remote: Compressing objects: 100% (3950/3950), done.  remote: Total 10909 (delta 7630), reused 9886 (delta 6914)  Receiving objects: 100% (10909/10909), 3.15 MiB | 526.00 KiB/s, done.  Resolving deltas: 100% (7630/7630), done.  debian@beaglebone:~$ cd rtems-source-builder/rtems  debian@beaglebone:~/rtems-source-builder/rtems$ git checkout 4.11.3  Note: checking out '4.11.3'.  You are in 'detached HEAD' state. You can look around, make experimental  changes and commit them, and you can discard any commits you make in this  state without impacting any branches by performing another checkout.  If you want to create a new branch to retain commits you create, you may  do so (now or later) by using -b with the checkout command again. Example:  git checkout -b <new-branch-name>  HEAD is now at 4671017... sb: Option --source-only-download does not download the source. |
| 2 | Install the RTEMS toolchain for i386-rtems.411. This will take a LONG time. Plan ahead to have something else to research and/or work on. | debian@beaglebone:~/rtems-source-builder/rtems$ sudo ../source-builder/sb-set-builder --prefix=/opt/rtems-4.11 4.11/rtems-i386  RTEMS Source Builder - Set Builder, 4.11 (4671017f41d3 modified)  Build Set: 4.11/rtems-i386  Build Set: 4.11/rtems-autotools.bset  Build Set: 4.11/rtems-autotools-internal.bset  config: tools/rtems-autoconf-2.69-1.cf  <snip> |
| 3 | Bootstrap the RTEMS source tree. | jbrandenburg@cfs-cross-compile:~$ git clone -b 4.11.3 git://git.rtems.org/rtems.git  <snip>  jbrandenburg@cfs-cross-compile:~$ export PATH=/opt/rtems-4.11/bin:$PATH  <snip>  jbrandenburg@cfs-cross-compile:~$ (cd rtems && ./bootstrap)  <snip> |
| 4 | Install the RTEMS pc686 BSP. | jbrandenburg@cfs-cross-compile:~/rtems$ mkdir b-pc686  <snip>  jbrandenburg@cfs-cross-compile:~$ cd b-pc686  <snip>  jbrandenburg@cfs-cross-compile:~/b-pc686$ ../rtems/configure --target=i386-rtems4.11 --enable-rtemsbsp=pc686 --prefix=/opt/x-tools/rtems-4.11 --enable-networking --enable-cxx --disable-posix --disable-deprecated BSP\_ENABLE\_VGA=0 CLOCK\_DRIVER\_USE\_TSC=1 USE\_COM1\_AS\_CONSOLE=1 BSP\_PRESS\_KEY\_FOR\_RESET=0 BSP\_RESET\_BOARD\_AT\_EXIT=1  <snip>  jbrandenburg@cfs-cross-compile:~/b-pc686$ make  <snip>  jbrandenburg@cfs-cross-compile:~/b-pc686$ sudo su  root@cfs-cross-compile:/home/jbrandenburg/b-pc686# export PATH=/opt/rtems-4.11/bin:$PATH  root@cfs-cross-compile:/home/jbrandenburg/b-pc686# make install  <snip>  root@cfs-cross-compile:/home/jbrandenburg/b-pc686# exit  exit  jbrandenburg@cfs-cross-compile:~/b-pc686$ cd .. |
| 5 | Install cexp-2.2.x (dynamic module loader library). | jbrandenburg@cfs-cross-compile:~$ wget http://www.slac.stanford.edu/~strauman/rtems/cexp/cexp-2.2.3.tgz  --2020-10-06 14:45:39-- http://www.slac.stanford.edu/~strauman/rtems/cexp/cexp-2.2.3.tgz  Resolving www.slac.stanford.edu (www.slac.stanford.edu)... 134.79.138.26  Connecting to www.slac.stanford.edu (www.slac.stanford.edu)|134.79.138.26|:80... connected.  HTTP request sent, awaiting response... 302 Found  Location: https://www.slac.stanford.edu/~strauman/rtems/cexp/cexp-2.2.3.tgz [following]  --2020-10-06 14:45:40-- https://www.slac.stanford.edu/~strauman/rtems/cexp/cexp-2.2.3.tgz  Connecting to www.slac.stanford.edu (www.slac.stanford.edu)|134.79.138.26|:443... connected.  HTTP request sent, awaiting response... 200 OK  Length: 1174448 (1.1M) [application/x-tar]  Saving to: ‘cexp-2.2.3.tgz’  cexp-2.2.3.tgz 100%[=======================================================>] 1.12M 1.24MB/s in 0.9s  2020-10-06 14:45:41 (1.24 MB/s) - ‘cexp-2.2.3.tgz’ saved [1174448/1174448]  jbrandenburg@cfs-cross-compile:~$ tar xzvf cexp-2.2.3.tgz  <snip>  # Modify the cexplock.c file  jbrandenburg@cfs-cross-compile:~$ cd cexp-CEXP\_Release\_2\_2\_3/  jbrandenburg@cfs-cross-compile:~/cexp-CEXP\_Release\_2\_2\_3$ diff cexplock.c.0 cexplock.c  1,2d0  < #define \_XOPEN\_SOURCE 500  <  jbrandenburg@cfs-cross-compile:~/cexp-CEXP\_Release\_2\_2\_3$ cd ..  jbrandenburg@cfs-cross-compile:~$ mkdir b-cexp  jbrandenburg@cfs-cross-compile:~$ cd b-cexp  jbrandenburg@cfs-cross-compile:~/b-cexp$ ../cexp-CEXP\_Release\_2\_2\_3/configure --with-rtems-top=/opt/x-tools/rtems-4.11 --host=i386-rtems4.11 --enable-std-rtems-installdirs  <snip>  jbrandenburg@cfs-cross-compile:~/b-cexp$ make  <snip>  jbrandenburg@cfs-cross-compile:~/b-cexp$ sudo su  root@cfs-cross-compile:/home/jbrandenburg/b-cexp# export PATH=/opt/rtems-4.11/bin:$PATH  root@cfs-cross-compile:/home/jbrandenburg/b-cexp# make install  <snip>  root@cfs-cross-compile:/home/jbrandenburg/b-cexp# exit  exit  jbrandenburg@cfs-cross-compile:~/b-cexp$ cd .. |

# Obtain the cFS Application

|  |  |  |
| --- | --- | --- |
| 1 | Get the cFS application source tree from github. There is a tag, “Activity08”, that identifies the state of the repository after the previous activity. | jbrandenburg@cfs-cross-compile:~$ git clone <https://github.com/METECS/cFS.git>  Cloning into 'cFS'...  remote: Enumerating objects: 5, done.  remote: Counting objects: 100% (5/5), done.  remote: Compressing objects: 100% (4/4), done.  remote: Total 410 (delta 1), reused 3 (delta 1), pack-reused 405  Receiving objects: 100% (410/410), 132.79 KiB | 755.00 KiB/s, done.  Resolving deltas: 100% (193/193), done.  jbrandenburg@cfs-cross-compile:~$ cd cFS  jbrandenburg@cfs-cross-compile:~/cFS$ git checkout Activity08  Note: checking out 'Activity08'.  You are in 'detached HEAD' state. You can look around, make experimental  changes and commit them, and you can discard any commits you make in this  state without impacting any branches by performing another checkout.  If you want to create a new branch to retain commits you create, you may  do so (now or later) by using -b with the checkout command again. Example:  git checkout -b <new-branch-name>  HEAD is now at eeb48ce Update CCDD branch  jbrandenburg@cfs-cross-compile:~/cFS$ git submodule sync --recursive  jbrandenburg@cfs-cross-compile:~/cFS$ git submodule update --init --recursive  Submodule 'CCDD' (https://github.com/METECS/CCDD.git) registered for path 'CCDD'  Submodule 'apps/ci\_lab' (https://github.com/metecs/ci\_lab.git) registered for path 'apps/ci\_lab'  Submodule 'apps/sample\_app' (https://github.com/metecs/sample\_app.git) registered for path 'apps/sample\_app'  Submodule 'apps/sample\_lib' (https://github.com/metecs/sample\_lib.git) registered for path 'apps/sample\_lib'  Submodule 'apps/sch\_lab' (https://github.com/metecs/sch\_lab.git) registered for path 'apps/sch\_lab'  Submodule 'apps/to\_lab' (https://github.com/metecs/to\_lab.git) registered for path 'apps/to\_lab'  Submodule 'cfe' (https://github.com/metecs/cFE.git) registered for path 'cfe'  Submodule 'osal' (https://github.com/metecs/osal.git) registered for path 'osal'  Submodule 'psp' (https://github.com/metecs/PSP.git) registered for path 'psp'  Submodule 'tools/cFS-GroundSystem' (https://github.com/metecs/cFS-GroundSystem.git) registered for path 'tools/cFS-GroundSystem'  Submodule 'tools/elf2cfetbl' (https://github.com/metecs/elf2cfetbl.git) registered for path 'tools/elf2cfetbl'  Submodule 'tools/tblCRCTool' (https://github.com/metecs/tblCRCTool.git) registered for path 'tools/tblCRCTool'  Cloning into '/home/jbrandenburg/cFS/CCDD'...  Cloning into '/home/jbrandenburg/cFS/apps/ci\_lab'...  Cloning into '/home/jbrandenburg/cFS/apps/sample\_app'...  Cloning into '/home/jbrandenburg/cFS/apps/sample\_lib'...  Cloning into '/home/jbrandenburg/cFS/apps/sch\_lab'...  Cloning into '/home/jbrandenburg/cFS/apps/to\_lab'...  Cloning into '/home/jbrandenburg/cFS/cfe'...  Cloning into '/home/jbrandenburg/cFS/osal'...  Cloning into '/home/jbrandenburg/cFS/psp'...  Cloning into '/home/jbrandenburg/cFS/tools/cFS-GroundSystem'...  Cloning into '/home/jbrandenburg/cFS/tools/elf2cfetbl'...  Cloning into '/home/jbrandenburg/cFS/tools/tblCRCTool'...  Submodule path 'CCDD': checked out 'bf67db6822a41cf42890f56c629231d15a297435'  Submodule path 'apps/ci\_lab': checked out '84f7d07a8007b4299c75cda643335cb38d200926'  Submodule path 'apps/sample\_app': checked out '46db0fbdeb3a06db9bbb32187dd41a92a0e3cd13'  Submodule path 'apps/sample\_lib': checked out 'df3dd7e6ec4708c594471bf5f9a358010328767a'  Submodule path 'apps/sch\_lab': checked out '2797196dc46206caf09aecc799cc0d3deba5d36a'  Submodule path 'apps/to\_lab': checked out 'a307084eaff7abd4631db2c053ae66db453a251a'  Submodule path 'cfe': checked out '0c3e9e0b33c55c4aef6de359eb7939aacbe93458'  Submodule path 'osal': checked out '8239eb5b521fdac68a09973e8e5ce6eb4a251e90'  Submodule path 'psp': checked out '91e209def67fc8a8f2e78691cd841165ca782e0c'  Submodule path 'tools/cFS-GroundSystem': checked out 'b9d4de03df7e3075b928ceff320e1a3eee76852b'  Submodule path 'tools/elf2cfetbl': checked out '3dc2a61f7d44bade7e00e57afe1f4c80381395d7'  Submodule path 'tools/tblCRCTool': checked out 'd61efe6ab69626633786d7cee6b2b30089fe52d8' |
| 2 | Reproduce the sample build files. | jbrandenburg@cfs-cross-compile:~/cFS$ cp cfe/cmake/Makefile.sample Makefile  jbrandenburg@cfs-cross-compile:~/cFS$ cp -r cfe/cmake/sample\_defs sample\_defs  jbrandenburg@cfs-cross-compile:~/cFS$ sed -i 's/undef OSAL\_DEBUG\_PERMISSIVE\_MODE/define OSAL\_DEBUG\_PERMISSIVE\_MODE/g' sample\_defs/default\_osconfig.h |
| 3 | Install cmake. | jbrandenburg@cfs-cross-compile:~/cFS$ sudo apt-get install cmake  <snip> |
| 4 | Modify the targets.cmake to build the new apps. | jbrandenburg@cfs-cross-compile:~/cFS/sample\_defs$ cat targets.cmake  <snip>  # Each target board can have its own HW arch selection and set of included apps  SET(TGT1\_NAME cpu1)  SET(TGT1\_APPLIST sample\_app sample\_lib ci\_lab to\_lab sch\_lab bcamp\_io\_app bcamp\_mn\_app)  SET(TGT1\_FILELIST cfe\_es\_startup.scr)  # CPU2/3 are duplicates of CPU1. These are not built by default anymore but are  # commented out to serve as an example of how one would configure multiple cpus.  #SET(TGT2\_NAME cpu2)  #SET(TGT2\_APPLIST sample\_app ci\_lab to\_lab sch\_lab)  #SET(TGT2\_FILELIST cfe\_es\_startup.scr)  #SET(TGT3\_NAME cpu3)  #SET(TGT3\_APPLIST sample\_app ci\_lab to\_lab sch\_lab)  #SET(TGT3\_FILELIST cfe\_es\_startup.scr) |
| 5 | Modify the startup script to load the sample apps. | jbrandenburg@cfs-cross-compile:~/cFS/sample\_defs$ cat cpu1\_cfe\_es\_startup.scr  CFE\_LIB, /cf/sample\_lib.so, SAMPLE\_LibInit, SAMPLE\_LIB, 0, 0, 0x0, 0;  CFE\_APP, /cf/sample\_app.so, SAMPLE\_AppMain, SAMPLE\_APP, 50, 16384, 0x0, 0;  CFE\_APP, /cf/bcamp\_io\_app.so, BCAMP\_IO\_AppMain, BCAMP\_IO\_APP, 50, 16384, 0x0, 0;  CFE\_APP, /cf/bcamp\_mn\_app.so, BCAMP\_MN\_AppMain, BCAMP\_MN\_APP, 50, 16384, 0x0, 0;  CFE\_APP, /cf/ci\_lab.so, CI\_Lab\_AppMain, CI\_LAB\_APP, 60, 16384, 0x0, 0;  CFE\_APP, /cf/to\_lab.so, TO\_Lab\_AppMain, TO\_LAB\_APP, 70, 16384, 0x0, 0;  CFE\_APP, /cf/sch\_lab.so, SCH\_Lab\_AppMain, SCH\_LAB\_APP, 80, 16384, 0x0, 0;  !  ! Startup script fields:  ! 1. Object Type -- CFE\_APP for an Application, or CFE\_LIB for a library.  ! 2. Path/Filename -- This is a cFE Virtual filename, not a vxWorks device/pathname  ! 3. Entry Point -- This is the "main" function for Apps.  ! 4. CFE Name -- The cFE name for the the APP or Library  ! 5. Priority -- This is the Priority of the App, not used for Library  ! 6. Stack Size -- This is the Stack size for the App, not used for the Library  ! 7. Load Address -- This is the Optional Load Address for the App or Library. Currently not implemented  ! so keep it at 0x0.  ! 8. Exception Action -- This is the Action the cFE should take if the App has an exception.  ! 0 = Just restart the Application  ! Non-Zero = Do a cFE Processor Reset  !  ! Other Notes:  ! 1. The software will not try to parse anything after the first '!' character it sees. That  ! is the End of File marker.  ! 2. Common Application file extensions:  ! Linux = .so ( ci.so )  ! OS X = .bundle ( ci.bundle )  ! Cygwin = .dll ( ci.dll )  ! vxWorks = .o ( ci.o )  ! RTEMS with S-record Loader = .s3r ( ci.s3r )  ! RTEMS with CEXP Loader = .o ( ci.o ) |

# Prepare the RTEMS PSP

|  |  |  |
| --- | --- | --- |
| 1 | Modify the CMakeLists.txt to include module specific PSP instructions. | jbrandenburg@cfs-cross-compile:~/cFS$ cd psp  jbrandenburg@cfs-cross-compile:~/cFS/psp$ git diff CMakeLists.txt  diff --git a/CMakeLists.txt b/CMakeLists.txt  index 176a3ef..15912cc 100644  --- a/CMakeLists.txt  +++ b/CMakeLists.txt  @@ -1,3 +1,5 @@  +set(CMAKE\_MODULE\_PATH "${CMAKE\_SOURCE\_DIR}/../psp/cmake/Modules" ${CMAKE\_MODULE\_PATH})  +  project(CFEPSP C)    if (NOT CFE\_SYSTEM\_PSPNAME) |
| 2 | Hack the psp/cmake/Modules/Platform/RTEMS.cmake file to remove the -c option. If RTEMS\_BSP\_C\_FLAGS is not defined (which it isn’t in this activity), build fails. Is there a better way to do this? | jbrandenburg@cfs-cross-compile:~/cFS/psp$ git diff cmake/Modules/Platform/RTEMS.cmake  diff --git a/cmake/Modules/Platform/RTEMS.cmake b/cmake/Modules/Platform/RTEMS.cmake  index 50fc2c8..6f6d260 100644  --- a/cmake/Modules/Platform/RTEMS.cmake  +++ b/cmake/Modules/Platform/RTEMS.cmake  @@ -37,7 +37,7 @@ set(CMAKE\_CXX\_CREATE\_SHARED\_LIBRARY ${CMAKE\_C\_CREATE\_SHARED\_MODULE})  # Then run "rtems-syms" and re-link the output into a final executable  set(CMAKE\_C\_LINK\_EXECUTABLE  "<CMAKE\_C\_COMPILER> <FLAGS> -o <TARGET>-prelink <OBJECTS> <CMAKE\_C\_LINK\_FLAGS> <LINK\_FLAGS> <LINK\_LIBRARIES>"  - "${RTEMS\_TOOLS\_PREFIX}/bin/rtems-syms -v -e -c ${RTEMS\_BSP\_C\_FLAGS} -C <CMAKE\_C\_COMPILER> -o <TARGET>-dl-sym.o <TARGET>-prelink"  + "${RTEMS\_TOOLS\_PREFIX}/bin/rtems-syms -v -e -C <CMAKE\_C\_COMPILER> -o <TARGET>-dl-sym.o <TARGET>-prelink"  "<CMAKE\_C\_COMPILER> <FLAGS> -o <TARGET> <TARGET>-dl-sym.o <OBJECTS> <CMAKE\_C\_LINK\_FLAGS> <LINK\_FLAGS> <LINK\_LIBRARIES>")    set(RTEMS\_TARGET\_PATH |
| 3 | Modify psp/fsw/pc-rtems/make/build\_options.cmake to add a macro to identify the operating system. | jbrandenburg@cfs-cross-compile:~/cFS/psp$ git diff fsw/pc-rtems/make/build\_options.cmake  diff --git a/fsw/pc-rtems/make/build\_options.cmake b/fsw/pc-rtems/make/build\_options.cmake  index ebacf2b..99f716d 100644  --- a/fsw/pc-rtems/make/build\_options.cmake  +++ b/fsw/pc-rtems/make/build\_options.cmake  @@ -1,3 +1,4 @@  # This indicates where to install target binaries created during the build  set(INSTALL\_SUBDIR "eeprom")  +add\_definitions("-D\_RTEMS\_OS\_") |

# Create the RTEMS Target

|  |  |  |
| --- | --- | --- |
| 1 | Create the target header files and startup script. | jbrandenburg@cfs-cross-compile:~/cFS/sample\_defs$ cp cpu1\_cfe\_es\_startup.scr cpu2\_cfe\_es\_startup.scr  jbrandenburg@cfs-cross-compile:~/cFS/sample\_defs$ cp cpu1\_msgids.h cpu2\_msgids.h  jbrandenburg@cfs-cross-compile:~/cFS/sample\_defs$ cp cpu1\_platform\_cfg.h cpu2\_platform\_cfg.h |
| 2 | Enable the second target in targets.cmake. | jbrandenburg@cfs-cross-compile:~/cFS/sample\_defs$ diff targets.cmake.0 targets.cmake  93,95c93,95  < #SET(TGT2\_NAME cpu2)  < #SET(TGT2\_APPLIST sample\_app ci\_lab to\_lab sch\_lab)  < #SET(TGT2\_FILELIST cfe\_es\_startup.scr)  ---  > SET(TGT2\_NAME cpu2)  > SET(TGT2\_APPLIST sample\_app sample\_lib ci\_lab to\_lab sch\_lab bcamp\_io\_app bcamp\_mn\_app)  > SET(TGT2\_FILELIST cfe\_es\_startup.scr) |
| 3 | Build the RTEMS toolchain by setting the contents of toolchain-cpu.cmake. | jbrandenburg@cfs-cross-compile:~/cFS/sample\_defs$ cat toolchain-cpu2.cmake  set(CMAKE\_SYSTEM\_NAME RTEMS)  set(CMAKE\_SYSTEM\_PROCESSOR i386)  set(CMAKE\_SYSTEM\_VERSION 4.11)  set(RTEMS\_TOOLS\_PREFIX "/opt/rtems-${CMAKE\_SYSTEM\_VERSION}")  set(RTEMS\_BSP\_PREFIX "/opt/x-tools/rtems-${CMAKE\_SYSTEM\_VERSION}")  set(RTEMS\_BSP pc686)  # specify the cross compiler - adjust accord to compiler installation  # This uses the compiler-wrapper toolchain that buildroot produces  set(TARGET\_PREFIX "${CMAKE\_SYSTEM\_PROCESSOR}-rtems${CMAKE\_SYSTEM\_VERSION}-")  set(CPUTUNEFLAGS "-march=i686 -mtune=i686")  SET(CMAKE\_C\_COMPILER "${RTEMS\_TOOLS\_PREFIX}/bin/${TARGET\_PREFIX}gcc")  SET(CMAKE\_CXX\_COMPILER "${RTEMS\_TOOLS\_PREFIX}/bin/${TARGET\_PREFIX}g++")  SET(CMAKE\_LINKER "${RTEMS\_TOOLS\_PREFIX}/bin/${TARGET\_PREFIX}ld")  SET(CMAKE\_ASM\_COMPILER "${RTEMS\_TOOLS\_PREFIX}/bin/${TARGET\_PREFIX}as")  SET(CMAKE\_STRIP "${RTEMS\_TOOLS\_PREFIX}/bin/${TARGET\_PREFIX}strip")  SET(CMAKE\_NM "${RTEMS\_TOOLS\_PREFIX}/bin/${TARGET\_PREFIX}nm")  SET(CMAKE\_AR "${RTEMS\_TOOLS\_PREFIX}/bin/${TARGET\_PREFIX}ar")  SET(CMAKE\_OBJDUMP "${RTEMS\_TOOLS\_PREFIX}/bin/${TARGET\_PREFIX}objdump")  SET(CMAKE\_OBJCOPY "${RTEMS\_TOOLS\_PREFIX}/bin/${TARGET\_PREFIX}objcopy")  # search for programs in the build host directories  SET(CMAKE\_FIND\_ROOT\_PATH\_MODE\_PROGRAM NEVER)  # for libraries and headers in the target directories  SET(CMAKE\_FIND\_ROOT\_PATH\_MODE\_LIBRARY ONLY)  SET(CMAKE\_FIND\_ROOT\_PATH\_MODE\_INCLUDE ONLY)  SET(CMAKE\_PREFIX\_PATH /)  # these settings are specific to cFE/OSAL and determines which  # abstraction layers are built when using this toolchain  SET(CFE\_SYSTEM\_PSPNAME pc-rtems)  SET(OSAL\_SYSTEM\_BSPTYPE pc-rtems)  SET(OSAL\_SYSTEM\_OSTYPE rtems) |
|  |  |  |

# Build the RTEMS Target

|  |  |  |
| --- | --- | --- |
| 1 | Generate the build structure. | jbrandenburg@cfs-cross-compile:~/cFS$ make distclean  rm -rf "build"  jbrandenburg@cfs-cross-compile:~/cFS$ make prep  <snip> |
| 2 | Build the targets. It will fail. | jbrandenburg@cfs-cross-compile:~/cFS$ make  <snip>  [100%] Linking C executable core-cpu2.exe  i386-rtems4.11-gcc: error: unrecognized command line option '--whole-archive'  i386-rtems4.11-gcc: error: unrecognized command line option '--no-whole-archive'  cpu2/CMakeFiles/core-cpu2.dir/build.make:97: recipe for target 'cpu2/core-cpu2.exe' failed  make[7]: \*\*\* [cpu2/core-cpu2.exe] Error 1  CMakeFiles/Makefile2:781: recipe for target 'cpu2/CMakeFiles/core-cpu2.dir/all' failed  make[6]: \*\*\* [cpu2/CMakeFiles/core-cpu2.dir/all] Error 2  Makefile:140: recipe for target 'all' failed  make[5]: \*\*\* [all] Error 2  CMakeFiles/cpu2-all.dir/build.make:57: recipe for target 'CMakeFiles/cpu2-all' failed  make[4]: \*\*\* [CMakeFiles/cpu2-all] Error 2  CMakeFiles/Makefile2:99: recipe for target 'CMakeFiles/cpu2-all.dir/all' failed  make[3]: \*\*\* [CMakeFiles/cpu2-all.dir/all] Error 2  CMakeFiles/Makefile2:171: recipe for target 'CMakeFiles/mission-all.dir/rule' failed  make[2]: \*\*\* [CMakeFiles/mission-all.dir/rule] Error 2  Makefile:214: recipe for target 'mission-all' failed  make[1]: \*\*\* [mission-all] Error 2  Makefile:114: recipe for target 'all' failed  make: \*\*\* [all] Error 2 |
| 3 | You can hack the file build/cpu2/cpu2/CMakeFiles/core-cpu2.dir/link.txt to pass the --whole-archie and --no-whole-archive options to the linker.  But, this is a hack and must be reproduced each time make distclean is run. Can you find a better solution?  Hint: The file cfe/cmake/target/CMakeLists.txt contains code to determine if a prefix is needed to pass options to the linker. It’s not working properly with the RTEMS target as we’ve defined it. | jbrandenburg@cfs-cross-compile:~/cFS/build/cpu2/cpu2/CMakeFiles/core-cpu2.dir$ cat link.txt  /opt/rtems-4.11/bin/i386-rtems4.11-gcc -Wall -B/opt/x-tools/rtems-4.11/i386-rtems4.11/pc686/lib -specs bsp\_specs -qrtems -D\_RTEMS\_OS\_ -g -o core-cpu2.exe-prelink CMakeFiles/core-cpu2.dir/src/target\_config.c.o -u Init -Wl,--whole-archive ../cfe\_core\_default\_cpu2/libcfe\_core\_default\_cpu2.a ../psp/pc-rtems/libpsp-pc-rtems.a ../osal/libosal.a -Wl,--no-whole-archive -lrtemscpu  /opt/rtems-4.11/bin/rtems-syms -v -e -C /opt/rtems-4.11/bin/i386-rtems4.11-gcc -o core-cpu2.exe-dl-sym.o core-cpu2.exe-prelink  /opt/rtems-4.11/bin/i386-rtems4.11-gcc -Wall -B/opt/x-tools/rtems-4.11/i386-rtems4.11/pc686/lib -specs bsp\_specs -qrtems -D\_RTEMS\_OS\_ -g -o core-cpu2.exe core-cpu2.exe-dl-sym.o CMakeFiles/core-cpu2.dir/src/target\_config.c.o -u Init -Wl,--whole-archive ../cfe\_core\_default\_cpu2/libcfe\_core\_default\_cpu2.a ../psp/pc-rtems/libpsp-pc-rtems.a ../osal/libosal.a -Wl,--no-whole-archive -lrtemscpu |
| 4 | Build the target. It will succeed!  Note: “make clean” will retain the changes we made to link.txt. | jbrandenburg@cfs-cross-compile:~/cFS$ make clean  make --no-print-directory -C "build" mission-clean  Built target cpu2-clean  ^[[A^[[ABuilt target cpu1-clean  Built target mission-clean  jbrandenburg@cfs-cross-compile:~/cFS$ make  <snip>  [100%] Linking C executable core-cpu2.exe  RTEMS Kernel Symbols 4.11.74768d25ac76  kernel: core-cpu2.exe-prelink  cache:load-sym: object files: 1  cache:load-sym: symbols: 4673  symbol C file: /tmp/rld--qJaaaa.c  symbol O file: core-cpu2.exe-dl-sym.o  [100%] Built target core-cpu2  [100%] Built target cpu2-all  [ 14%] Building C object tools/cFS-GroundSystem/Subsystems/cmdUtil/CMakeFiles/cmdUtil.dir/cmdUtil.c.o  [ 28%] Building C object tools/cFS-GroundSystem/Subsystems/cmdUtil/CMakeFiles/cmdUtil.dir/SendUdp.c.o  [ 42%] Linking C executable cmdUtil  [ 42%] Built target cmdUtil  [ 71%] Built target elf2cfetbl  [ 85%] Building C object tools/tblCRCTool/CMakeFiles/cfe\_ts\_crc.dir/cfe\_ts\_crc.c.o  [100%] Linking C executable cfe\_ts\_crc  [100%] Built target cfe\_ts\_crc  Built target mission-all |
| 5 | FYI: While these are the steps to cross-compile for a different platform, the result is not complete. |  |

# 