SSF-OSリポジトリのRead Me文書

# 1. SSF-OSとは何か

このリポジトリに登録されたファイル群は、ARM Cortex-A(Raspberry Pi 3 B+)向けのリアルタイムOS(名称SSF-OS)開発ディレクトリです。

# 2. 何ができるか

ARM Cortex-Aに組み込むためのカーネル・ライブラリのビルド、WindowsマシンとRaspberry Piをつないでデモンストレーションを行うサンプル・プログラムのビルドができます。

# 3. カーネル・ライブラリのビルド方法

## 3.1 Cygwinのインストール

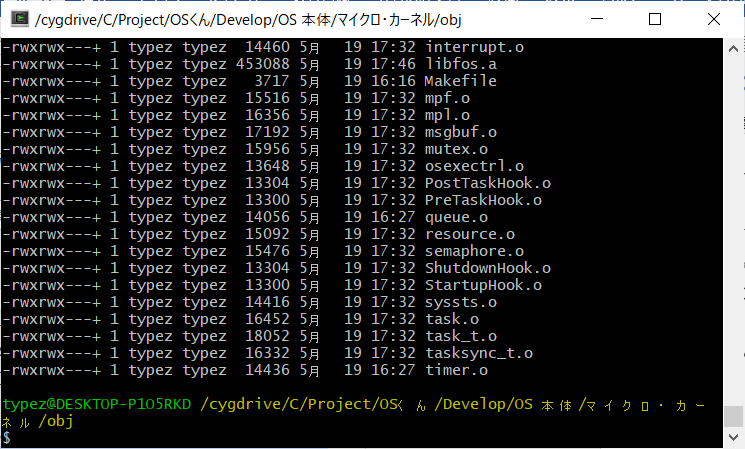
Cygwinのサイト(<https://www.cygwin.com/>)にアクセスし、**[setup-x86\_64.exe](https://www.cygwin.com/setup-x86_64.exe)**をダウンロードする。

ダウンロードが完了したらファイルをダブルクリックし、セットアップを行ってください。makeは必須ですので、必ずインストールしてください。

## 3.2 カーネル・ライブラリのビルド

リポジトリが”C:\Project\OSくん”というディレクトリにチェックアウトされているものとして説明します。

まず、Cygwinのコンソールから”/cygdrive/C/Project/OSくん/Develop/OS 本体/マイクロ・カーネル/obj/”に移動します。そこで、コマンド”make”を実行してください。このディレクトリにカーネル・ライブラリ”libfos.a”が生成されます。



## 3.3 ユーザ・オウン・コーディング部のオブジェクトをビルド

Cygwinのコンソールから” /cygdrive/C/Project/OSくん/Develop/OS 本体/ユーザ・オウン・コーディング部/obj/”に移動します。このディレクトリで”make”を実行してください。ユーザ・オウン・コーディング部のオブジェクト群”boot.o”, “interrupt.o”, “stack.o”が生成されます。

# 4. サンプル・プログラムのビルド方法

## 4.1 ファイル群のコピー

“C:\Project\OSくん\Develop\OS 本体\マイクロ・カーネル\inc”以下にあるファイル群を、kernel.incを**除いて**、” C:\Project\OSくん\Develop\OS 本体\サンプル・プログラム\inc”以下にコピーします。

次に、” C:\Project\OSくん\Develop\OS 本体\マイクロ・カーネル\obj”以下にある”libfos.a”を” C:\Project\OSくん\Develop\OS 本体\サンプル・プログラム\obj”にコピーします。

最後に、” C:\Project\OSくん\Develop\OS 本体\ユーザ・オウン・コーディング部\obj”以下にある”boot.o”, “interrupt.o”, “stack.o”を” C:\Project\OSくん\Develop\OS 本体\サンプル・プログラム\obj”にコピーします。

## 4.2 サンプル用オブジェクト群のビルド

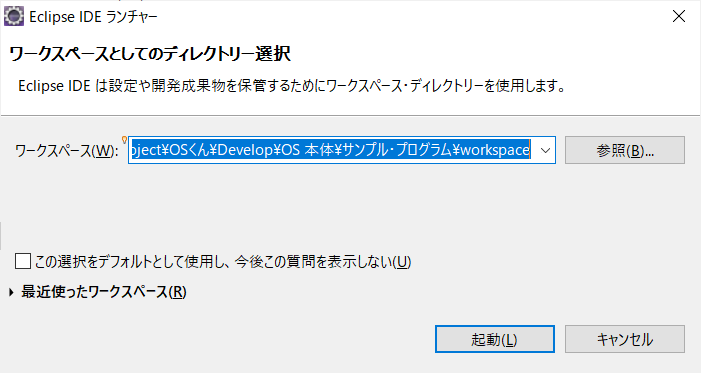
Cygwinのコンソールから” /cygdrive/C/Project/OSくん/Develop/OS 本体/サンプル・プログラム/conf/”に移動します。このディレクトリで”make”を実行します。これで、必要なファイルが生成されます。

## 4.3 Pleiadesの設定

サンプル・プログラムのビルドはPleiades(eclipse)で行います。ARM Cortex-A(Raspberry Pi)用にPleiadesでビルドを行うためには設定が必要です。

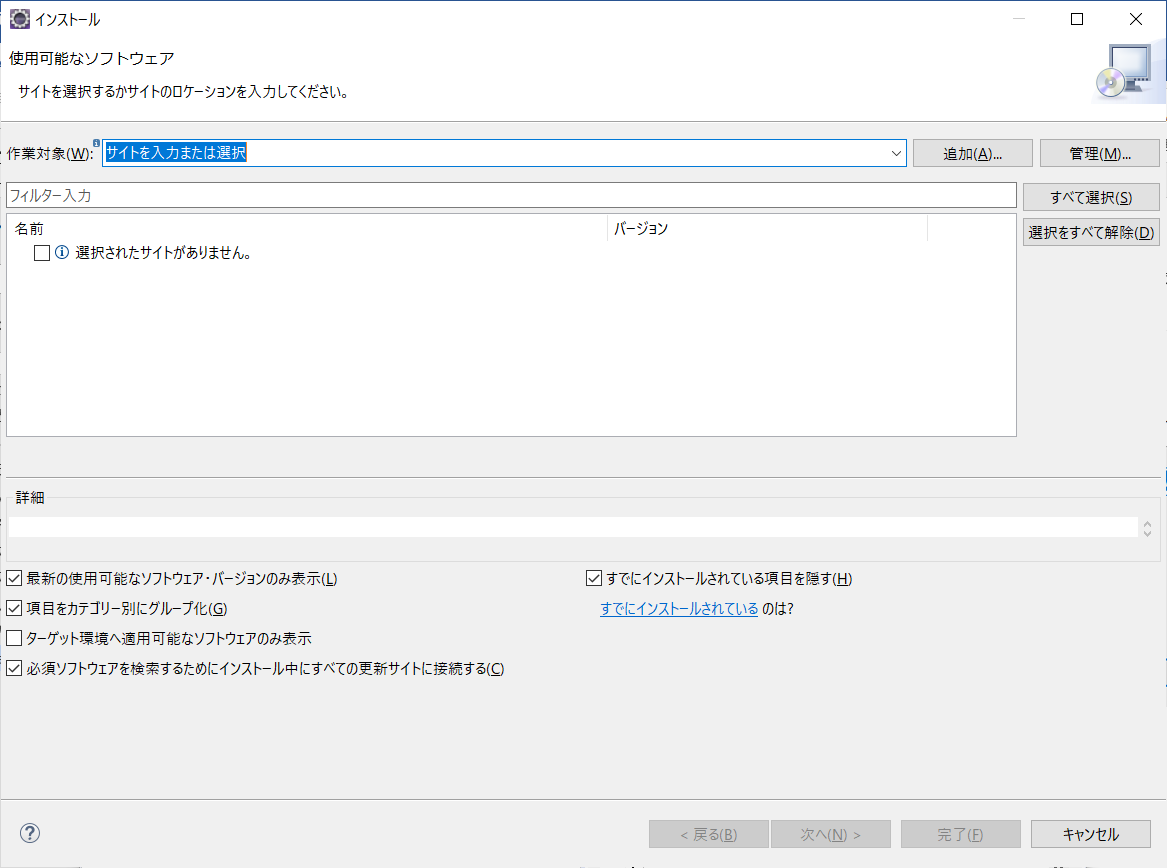
1. 起動時のワークスペース選択

“C:\Project\OSくん\Develop\OS 本体\サンプル・プログラム\workspace”を選択する。



2. クロス・コンパイラー・サポートのインストール

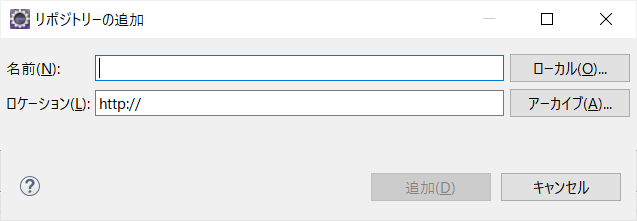
「ヘルプ」→「新規ソフトウェアのインストール」をクリックする。



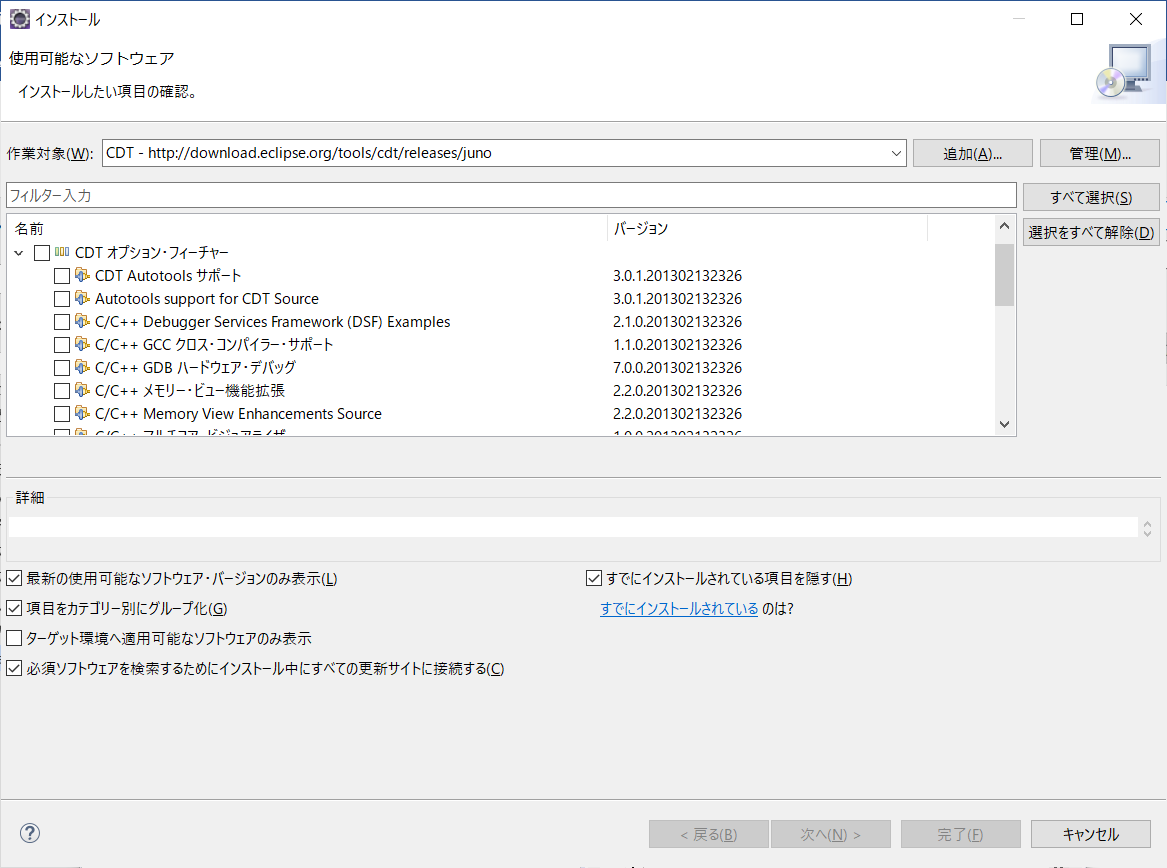
「追加(A)」をクリックして以下の情報を入力。

名前：CDT

ロケーション：http://download.eclipse.org/tools/cdt/releases/juno

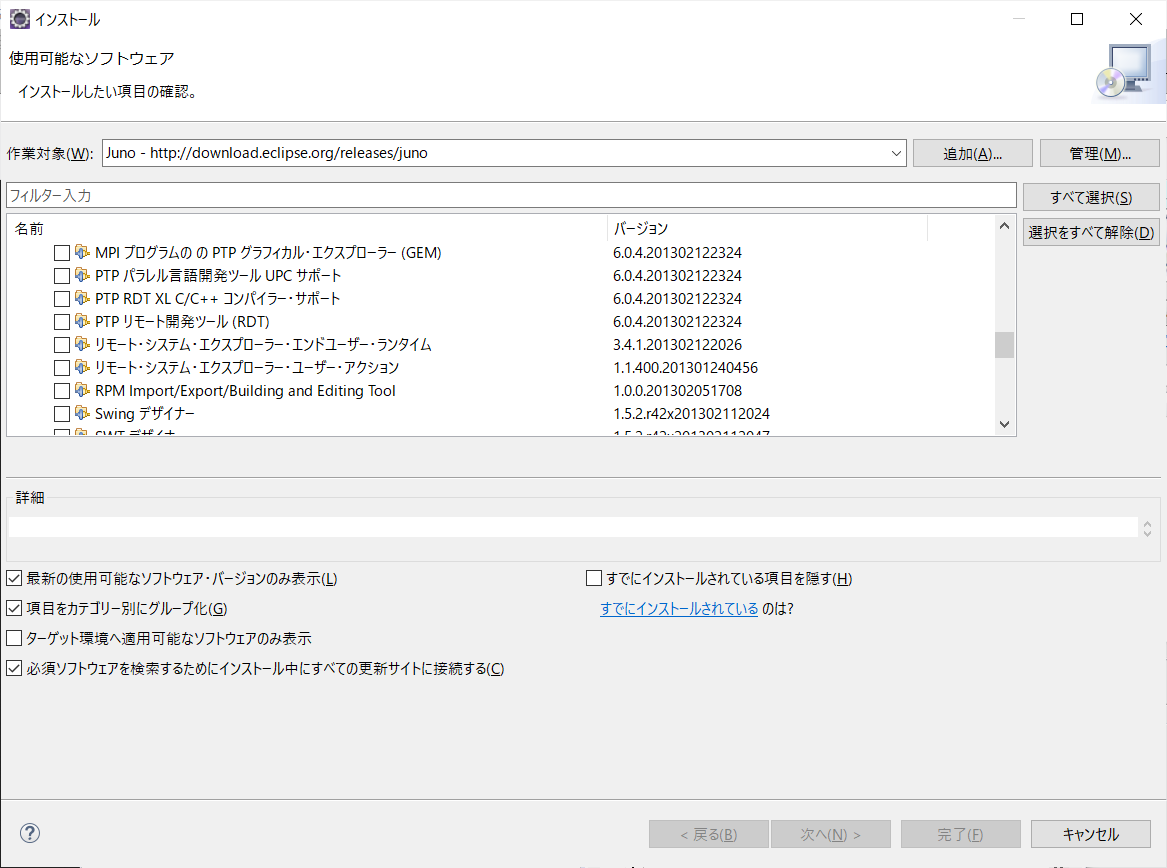


「CDTオプション・フィーチャー」の「C/C++ GCCクロス・コンパイラー・サポート」をインストールする。



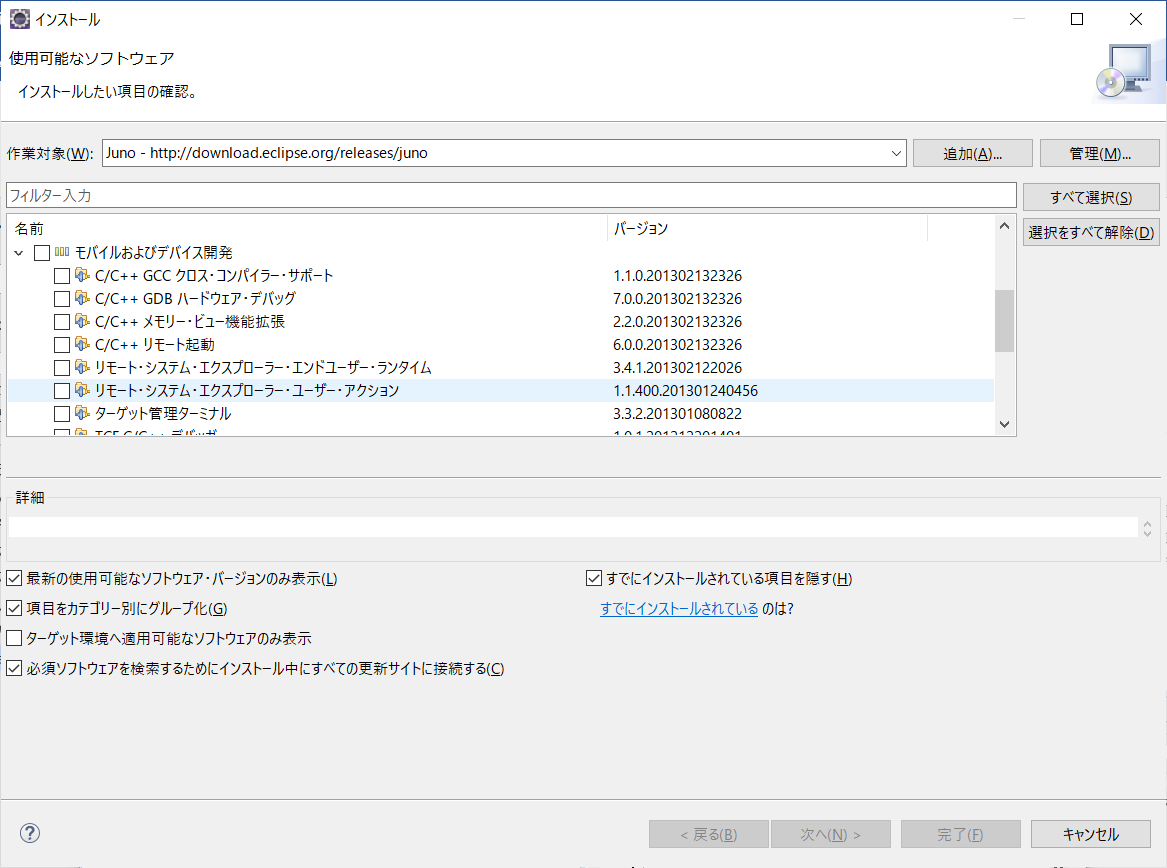
3. リモート・システム・エクスプローラーのインストール

「一般用ツール」の「リモート・システム・エクスプローラー・エンドユーザー・ランタイム」と、「リモート・システム・エクスプローラー・ユーザー・アクション」をインストールする。



4. C/C++ リモート起動のインストール

「モバイルおよびデバイス開発」の「C/C++ リモート起動」をインストールする。



5. デバッグの設定

「実行」→「デバッグの構成」をクリックする。

ですが、残念ながらPleiadesでデバッグを行うことはできません。

## 4.4 サンプル・プログラムをビルドするもう一つの方法

Cygwinで、”/cygdrive/C/Project/OSくん/Develop/OS 本体/サンプル・プログラム/workspace/SSF-OS\_Sample\_Project/Debug”に移動します。

このディレクトリでmakeコマンドを実行することにより、サンプル・プログラムをビルドすることができます。

# 5. サンプル・プログラムのデバッグ方法

Raspberry Piの実機と接続する方法は後述します。

まずは、QEMUと接続する方法を記します。

QEMUのインストール・フォルダに移動し、下記のコマンドを実行します。

|  |
| --- |
| qemu-system-arm -kernel C:\SysGCC\raspberry\bin\SSF-OS\_Sample\_Project.elf -cpu cortex-a8 -M raspi2 -m 256 -no-reboot -nographic -monitor null -serial stdio -s -S |

“C:\SysGCC\raspberry\bin\SSF-OS\_Sample\_Project.elf”の部分は、サンプル・プログラムのオブジェクト・ファイルを置いたパスを指定します。

Raspberry Piの実機を使う場合は、下記のコマンドを指定します。

Raspberry Pi 3 B+の場合

|  |
| --- |
| openocd.exe --file C:\Project\scripts\interface\ftdi\olimex-arm-usb-tiny-h.cfg -c "transport select jtag" --file C:\Project\scripts\target\rpi3.cfg |

rpi3.cfgの中身

|  |
| --- |
| # we need to enable srst even though we don't connect it  reset\_config trst\_and\_srst  adapter speed 1000  jtag\_ntrst\_delay 500  if { [info exists CHIPNAME] } {  set \_CHIPNAME $CHIPNAME  } else {  set \_CHIPNAME rpi3  }  #  # Main DAP  #  if { [info exists DAP\_TAPID] } {  set \_DAP\_TAPID $DAP\_TAPID  } else {  set \_DAP\_TAPID 0x4ba00477  }  jtag newtap $\_CHIPNAME cpu -irlen 32 -ircapture 0x1 -irmask 0xf -expected-id $\_DAP\_TAPID -enable -ignore-syspwrupack  set \_TARGETNAME $\_CHIPNAME.cpu  set \_TARGETNAME\_1 $\_CHIPNAME.cpu1  set \_TARGETNAME\_2 $\_CHIPNAME.cpu2  set \_TARGETNAME\_3 $\_CHIPNAME.cpu3  dap create $\_TARGETNAME.dap -chain-position $\_TARGETNAME -ignore-syspwrupack  dap create $\_TARGETNAME\_1.dap -chain-position $\_TARGETNAME\_1 -ignore-syspwrupack  dap create $\_TARGETNAME\_2.dap -chain-position $\_TARGETNAME\_2 -ignore-syspwrupack  dap create $\_TARGETNAME\_3.dap -chain-position $\_TARGETNAME\_3 -ignore-syspwrupack  #cti create $\_TARGETNAME.cti -dap $\_CHIPNAME.dap -ap-num 0 -ctibase 0x80018000  #cti create $\_TARGETNAME\_1.cti -dap $\_CHIPNAME.dap -ap-num 1 -ctibase 0x80019000  #cti create $\_TARGETNAME\_2.cti -dap $\_CHIPNAME.dap -ap-num 2 -ctibase 0x8001a000  #cti create $\_TARGETNAME\_3.cti -dap $\_CHIPNAME.dap -ap-num 3 -ctibase 0x8001b000  #$\_TARGETNAME.cti enable on  #$\_TARGETNAME\_1.cti enable on  #$\_TARGETNAME\_2.cti enable on  #$\_TARGETNAME\_3.cti enable on  #target create $\_TARGETNAME aarch64 -dap $\_TARGETNAME.dap -coreid 0 -dbgbase 0x80010000 -cti $\_TARGETNAME.cti  #target create $\_TARGETNAME\_1 aarch64 -dap $\_TARGETNAME\_1.dap -coreid 1 -dbgbase 0x80012000 -cti $\_TARGETNAME\_1.cti  #target create $\_TARGETNAME\_2 aarch64 -dap $\_TARGETNAME\_2.dap -coreid 2 -dbgbase 0x80014000 -cti $\_TARGETNAME\_2.cti  #target create $\_TARGETNAME\_3 aarch64 -dap $\_TARGETNAME\_3.dap -coreid 3 -dbgbase 0x80016000 -cti $\_TARGETNAME\_3.cti  target create $\_TARGETNAME cortex\_a -dap $\_TARGETNAME.dap -coreid 0 -dbgbase 0x80010000 -cti $\_TARGETNAME.cti  target create $\_TARGETNAME\_1 cortex\_a -dap $\_TARGETNAME\_1.dap -coreid 1 -dbgbase 0x80012000 -cti $\_TARGETNAME\_1.cti  target create $\_TARGETNAME\_2 cortex\_a -dap $\_TARGETNAME\_2.dap -coreid 2 -dbgbase 0x80014000 -cti $\_TARGETNAME\_2.cti  target create $\_TARGETNAME\_3 cortex\_a -dap $\_TARGETNAME\_3.dap -coreid 3 -dbgbase 0x80016000 -cti $\_TARGETNAME\_3.cti  target smp $\_TARGETNAME $\_TARGETNAME\_1 $\_TARGETNAME\_2 $\_TARGETNAME\_3  cortex\_a smp on  jtag configure $\_CHIPNAME.jrc -event setup "jtag tapenable $\_CHIPNAME.cpu"  $\_TARGETNAME configure -work-area-phys 0x640000 -work-area-size 8096  $\_TARGETNAME configure -event reset-assert-pre "cortex\_a dbginit"  $\_TARGETNAME configure -event gdb-attach { halt } |

Raspberry Pi 4 Model Bの場合

|  |
| --- |
| openocd.exe --file C:\Project\scripts\interface\ftdi\olimex-arm-usb-tiny-h.cfg -c "transport select jtag" --file C:\Project\scripts\target\raspi4.cfg |

raspi４.cfgの中身

|  |
| --- |
| set \_CHIPNAME bcm2711  set \_DAP\_TAPID 0x4ba00477  adapter\_khz 1000  transport select jtag  reset\_config trst\_and\_srst  telnet\_port 4444  # create tap  jtag newtap auto0 tap -irlen 4 -expected-id $\_DAP\_TAPID  # create dap  dap create auto0.dap -chain-position auto0.tap  set CTIBASE {0x80420000 0x80520000 0x80620000 0x80720000}  set DBGBASE {0x80410000 0x80510000 0x80610000 0x80710000}  set \_cores 4  set \_TARGETNAME $\_CHIPNAME.a72  set \_CTINAME $\_CHIPNAME.cti  set \_smp\_command ""  for {set \_core 0} {$\_core < $\_cores} { incr \_core} {  cti create $\_CTINAME.$\_core -dap auto0.dap -ap-num 0 -baseaddr [lindex $CTIBASE $\_core]  set \_command "target create ${\_TARGETNAME}.$\_core aarch64 \  -dap auto0.dap -dbgbase [lindex $DBGBASE $\_core] \  -coreid $\_core -cti $\_CTINAME.$\_core"  if {$\_core != 0} {  set \_smp\_command "$\_smp\_command $\_TARGETNAME.$\_core"  } else {  set \_smp\_command "target smp $\_TARGETNAME.$\_core"  }  eval $\_command  }  eval $\_smp\_command  targets $\_TARGETNAME.0  $\_TARGETNAME.0 configure -event reset-assert-pre "aarch64 dbginit"  $\_TARGETNAME.0 configure -event gdb-attach { halt } |

OpenOCDを起動したら、GDBを接続してデバッグをしましょう。

ただし、現在のところRaspberry Piの実機では、ldr疑似命令の実行で暴走してしまう問題があり、期待した実行結果を得ることができていません。

QEMUでは、サンプル・プログラムを期待通り実行することができます。