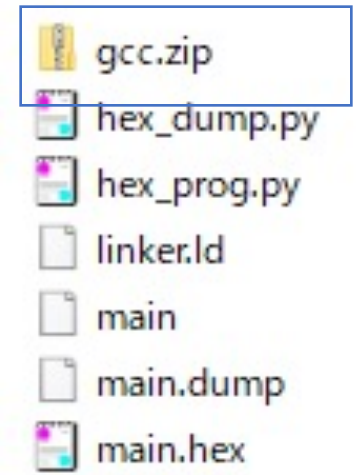
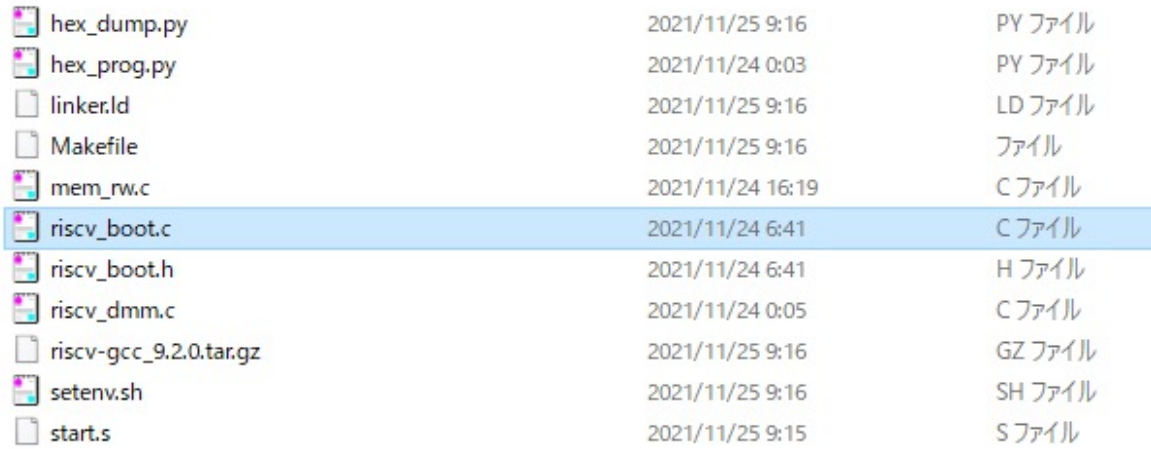
RISC-V_Cコンパイル

2021年11月30日

事前準備

- 提供ファイルにあるC_Compile->gcc.zipを解凍してください。
- 今回は全て事前に用意していますが、もし１から行うならこの手順を参考にしてください。
- 右の画像の状態で始めることを前提として書いていきます。





はじめにやること

- riscv-gcc_9.2.0.tar.gzファイルを解凍します。解凍したらgccというファイルが生成されるのでそのファイルの中にsetenv.shを入れてください。

- その後Makefileを開いて、今回のプログラム(mem_rw.c)に合わせて修正を行います。

  修正箇所

  ・SRC = hello_world.c simrv.c start.s　→　SRC = mem_rw.c start.s に変更します。

  ・HDR = simvr.h → HDR = に変更します。

  ・./hex_dump.py main → /usr/bin/python3 ./hex_dump.py main に変更します。

- mem_rw.cを開いてDMEM_BASEのアドレスが合っているかviviadoのアドレスマップで確認してください。またIMEMのサイズを確認してください。今回のvivadoだと４Kになっているはずです

- hex_dump.pyを開いて work_region = 4096 になっていることを確認してください。もしIMEMが８Kなら 8 * 1024にしてください。

コマンド操作

- 変更したらコマンドプロンプトを開いて、cdコマンドで提供ファイル->C_Compile階層まで移動します。

- source ./gcc/setenv.shとコマンドを打ってください。

　もしエラーが出た際はsetenv.shの改行コードを使用しているPCに合わせて変更してください。まず初めにmv setenv.sh setenv.sh_orgとコマンドを打ちます。そうするとsetenv.sh_orgが作成されます。その後、

Windows → nkf –CRLF setenv.sh_org > setenv.sh
Mac OS(9以前) → nkf –CR setenv.sh_org > setenv.sh
UNIX系 → nkf –LF setenv.sh_org > setenv.sh

- その後、上記で作成されたsetenv.shをgccファイルにコピーしてもう一度source ./gcc/setenv.shと実行してみてください。

コンパイル

- **通ればriscv64-unknown-elf-gcc –-versionを実行してみてください。下記の画像のように出るはずです。**

```
riscv64-unknown-elf-gcc (GCC) 9.2.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE.
```

- **makeコマンドを行うとコンパイルが始まります。**
  main、main.hex、main.dump、main_rtl.hexの４つのファイルが出来るはずです。

- **コンパイル中にlibpfr.so.4: cannot open shared object file;というエラーが出たらsudo ln -s /usr/lib/x86_64-linux-gnu/libmpfr.so.6 /usr/lib/x86_64-linux-gnu/libmpfr.so.4をしてからmakeを実行してみてください。**

Vitisで動かすなら

- hex_prog.pyはmain_rtl.hexをriscv_imm.cに変換します
  python hex_prog.py と実行するとriscv_imm.cが生成されます。
  もしエラーが出た際は/usr/bin/python3 hex_prog.pyと打ってください。
  riscv_imm.cが同様に生成されるはずです。

- Vitisでプロジェクトを作り(ひな型はHelloworldで)、Helloworld.cを削除してください。その後、先ほど作成したriscv_imm.cとriscv_boot.c、riscv_boot.h、riscv_dmm.cをプロジェクトのsrcに置いてBulidします。

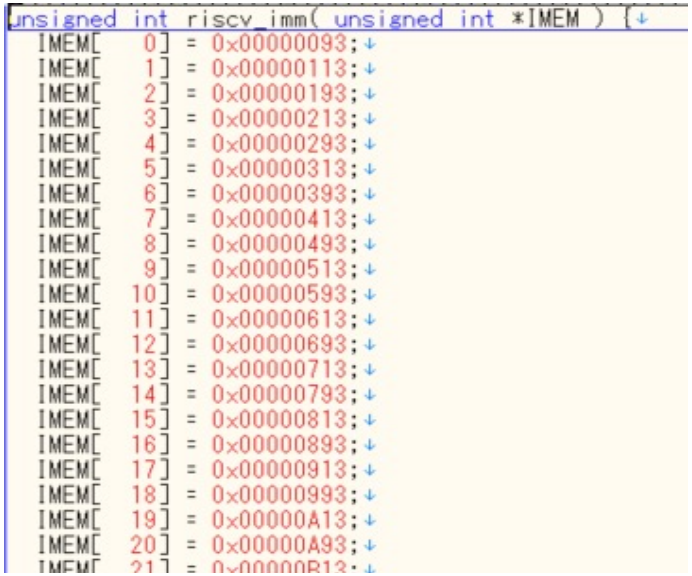
- これでエラーがなければ、Vitisで動作させることが可能です。また今回作成したプログラムはvivado立ち上がり～Vitis起動までで使用したriscv_read_add_write.cと同じで0x12+0x34=0x46を行い、合っていれば
  LED0点滅→LED1点滅→LED1点灯を行います。

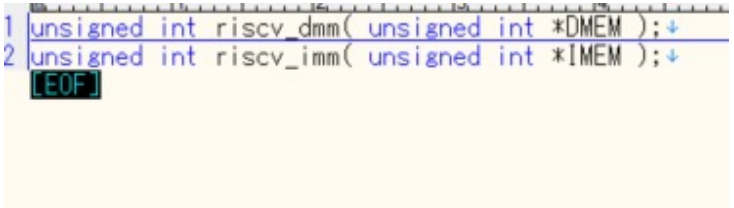
プログラム説明(riscv_imm.c、 riscv_dmm.c、 riscv_boot.h)

- **左の図がriscv_imm.c、 右上の図がriscv_dmm.c、 右下の図がriscv_boot.hとなっています。**

- **IMEMは[0]～[117]まで値が割り振られており、[118]以降は0になっています。**
- **DMEMは[0]に0x12、[1]に0x34が割り振られています。**

- **riscv_boot.hではこのように定義を書いています。**





プログラム説明(riscv_boot.c)

- GPIO_BASE、IMEM_BASE、DMEM_BASEはVivadoのアドレスマップを参考に変更してください。
- Inum=riscv_imm(IMEM)

  for(n=0;n<inum;n++){REG(IMEM_BASE + (n*4)) = IMEM[n] }

  これはriscv_imm.cに記載されているIMEM[n]の値をREG(IMEM_BASE +n*4)に値を入れています。DMEMも同様です。

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "sleep.h"
#include "riscv_boot.h"

#define REG(address) *(volatile unsigned int*)(address)

#define GPIO_BASE (0xA0010000) /* XPAR_AXI_GPIO_0_BASEADDR */
#define GPIO_DATA (GPIO_BASE + 0x0000)
#define GPIO_TRI  (GPIO_BASE + 0x0004)

#define IMEM_BASE  (0xA0000000)
#define DMEM_BASE  (0xA0002000)

// GPIO[0]=RISC_V_RESET Low active
// GPIO[1]=LED0
// GPIO[2]=LED1

int main()
{
  unsigned int inum,dnum,c,n;
    unsigned int IMEM[4096];
    unsigned int DMEM[4096];
    init_platform();

    print("Hello World¥n¥r");
    REG(GPIO_TRI) = 0x00;
    REG(GPIO_DATA) = 0x02; // LED0
```

```
    c = 0;
    inum=riscv_imm(IMEM);
    for(n=0;n<inum;n++){
      REG(IMEM_BASE+(n*4))=IMEM[n];
    }
    dnum=riscv_dmm(DMEM);
    for(n=0;n<dnum;n++){
      REG(DMEM_BASE+(n*4))=DMEM[n];
    }
    sleep(1);
    REG(GPIO_DATA) = 0x04; // LED1
    sleep(1);
    REG(GPIO_DATA) = 0x05; // LED2,Reset off
    sleep(1);
    REG(GPIO_DATA) = 0x06; // LED1,3,Reset on
    sleep(1);
    c =  REG(DMEM_BASE+8);
    printf("%x¥n¥r",c);
    if(c==0x00000046){ // 0x12+0x34=0x46
        printf("Pass¥n¥r");
    }else{
        printf("Fail¥n¥r");
    }
    sleep(1);

    print("Successfully ran Hello World application¥n¥r");
    cleanup_platform();
    return 0;
```