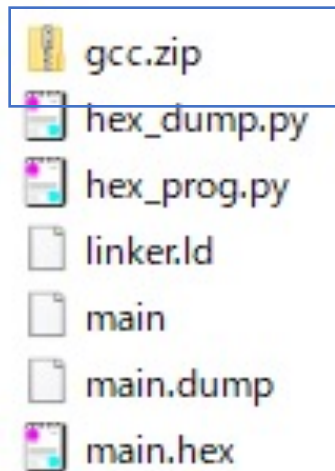


RISC-V_Cコンパイラ

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。
- 右の画像の状態始めることを前提として書いていきます。



hex_dump.py	2021/11/25 9:16	PY ファイル
hex_prog.py	2021/11/24 0:03	PY ファイル
linker.ld	2021/11/25 9:16	LD ファイル
Makefile	2021/11/25 9:16	ファイル
mem_rw.c	2021/11/24 16:19	C ファイル
riscv_boot.c	2021/11/24 6:41	C ファイル
riscv_boot.h	2021/11/24 6:41	H ファイル
riscv_dmm.c	2021/11/24 0:05	C ファイル
riscv-gcc_9.2.0.tar.gz	2021/11/25 9:16	GZ ファイル
setenv.sh	2021/11/25 9:16	SH ファイル
start.s	2021/11/25 9:15	S ファイル

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vivado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Build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起動までで使った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。

```
unsigned int riscv_imm( unsigned int *IMEM ) {  
    IMEM[ 0 ] = 0x00000093; ↓  
    IMEM[ 1 ] = 0x00000113; ↓  
    IMEM[ 2 ] = 0x00000193; ↓  
    IMEM[ 3 ] = 0x00000213; ↓  
    IMEM[ 4 ] = 0x00000293; ↓  
    IMEM[ 5 ] = 0x00000313; ↓  
    IMEM[ 6 ] = 0x00000393; ↓  
    IMEM[ 7 ] = 0x00000413; ↓  
    IMEM[ 8 ] = 0x00000493; ↓  
    IMEM[ 9 ] = 0x00000513; ↓  
    IMEM[10] = 0x00000593; ↓  
    IMEM[11] = 0x00000613; ↓  
    IMEM[12] = 0x00000693; ↓  
    IMEM[13] = 0x00000713; ↓  
    IMEM[14] = 0x00000793; ↓  
    IMEM[15] = 0x00000813; ↓  
    IMEM[16] = 0x00000893; ↓  
    IMEM[17] = 0x00000913; ↓  
    IMEM[18] = 0x00000993; ↓  
    IMEM[19] = 0x00000A13; ↓  
    IMEM[20] = 0x00000A93; ↓  
    IMEM[21] = 0x00000B13; ↓  
}
```

```
unsigned int riscv_dmm( unsigned int *DMEM ) {  
    DMEM[ 0 ] = 0x00000012; ↓  
    DMEM[ 1 ] = 0x00000034; ↓  
    return 2; ↓  
} ↓
```

```
1 unsigned int riscv_dmm( unsigned int *DMEM ); ↓  
2 unsigned int riscv_imm( unsigned int *IMEM ); ↓  
[EOF]
```

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變更してください。

- 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;
```