Nesse tutorial iremos compilar um programa para o HPS (Arm Cortex A) que será capaz de controlar os LEDs e ler os botões da placa que estão conectados ao HPS.

Note pelo diagrama anterior extraído do manual do usuário, existem LEDs e botões conectados diretamente ao HPS, e outros conectados a FPGA.

Duas são as possíveis abordagens para programarmos o HPS:

baremetal

Faríamos um programa que seria executado no ARM HPS sem nenhum sistema operacional. Como detalhado no diagrama :

- ref: Altera Bare Metal User Guide

Nessa maneira, a aplicação deve ser capaz de realizar toda a inicialização de HW necessária para que o processador rode corretamente. Se a aplicação for executada sobre um sistema operacional, toda essa etapa é de compilação é responsabilidade do SO.

Para isso, utiliza-se a IDE da ARM chamada de DS-5

Sistema operacional

Duas são as opções: Real Time ou Linux (existe outros sistemas operacionais para ARM que não o linux: BSD,...), a escolha dependerá da aplicação.

Com o uso de um sistema operacional a parte referente ao HW é responsabilidade do kernel (ou dos desenvolvedores que estão adequando o kernel ao HW, que é o caso de vocês). Diversos são os ganhos de utilizar um sistema operacional do tipo Linux (as perdas também são grandes: maior ocupação de memória, maior latências,...) tais como:

- Device drivers
- Portabilidade

- Segurança
- Rede

Linux

Nesse tutorial iremos compilar um programa e executar no Linux e esse programa que será executado no user space. Para isso iremos utilizar a toolchain definida no tutorial passado.

Iremos utilizar como base o código: `Tutoriais/HPS-BlinkLED/main.c` e crosscopilar esse código para o nosso HPS. Iremos seguir as etapas a seguir:

1. Instale o toolchain
2. Grave o SDcard com a imagem padrão fornecida
3. Compile o código
4. Passe o código para o HPS
    - via SDCard
5. Execute o linux no sistema emabarcado
6. Conecte-se ao terminal do target (ssh ou terminal)
7. Execute o código no HPS e veja os LEDs piscarem !

Compilando

Crie uma pasta para essa entrega, digamos: `HPS-Hello` , nela insira:

- um arquivo `main.c` com o conteúdo a seguir:

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include "hwlib.h"
#include "socal/socal.h"
#include "socal/hps.h"
#include "socal/alt_gpio.h"
```

```
#define HW_REGS_BASE ( ALT_STM_OFST )
#define HW_REGS_SPAN ( 0x04000000 )
#define HW_REGS_MASK ( HW_REGS_SPAN - 1 )

#define USER_IO_DIR     (0x01000000)
#define BIT_LED         (0x01000000)
#define BUTTON_MASK     (0x02000000)

int main(int argc, char **argv)
{
  void *virtual_base;
  int fd;
  uint32_t scan_input;
  int i;

  // map the address space for the LED registers into user space
so we can interact with them.
  // we'll actually map in the entire CSR span of the HPS since we
want to access various registers within that span
  if( ( fd = open( "/dev/mem", ( O_RDWR | O_SYNC ) ) ) == -1 ) {
    printf( "ERROR: could not open \"/dev/mem\"...\n" );
    return( 1 );
  }

  virtual_base = mmap( NULL, HW_REGS_SPAN, ( PROT_READ |
PROT_WRITE ), MAP_SHARED, fd, HW_REGS_BASE );

  if( virtual_base == MAP_FAILED ) {
    printf( "ERROR: mmap() failed...\n" );
    close( fd );
    return( 1 );
  }

  // GPIO
  // gpio register absolut address
  //
  uint32_t *gpio1_swporta_ddr_addr = (uint32_t *)(virtual_base +
((uint32_t)(ALT_GPIO1_SWPORTA_DDR_ADDR) & (uint32_t)
(HW_REGS_MASK)));
  uint32_t *gpio1_swporta_dr_addr  =(uint32_t *)(virtual_base +
((uint32_t)(ALT_GPIO1_SWPORTA_DR_ADDR) & (uint32_t)
(HW_REGS_MASK )));
  uint32_t *gpio1_porta_ext_addr   =(uint32_t *)(virtual_base +
((uint32_t)(ALT_GPIO1_EXT_PORTA_ADDR) & (uint32_t)
(HW_REGS_MASK) ));

  // initialize the pio controller
  // led: set the direction of the HPS GPIO1 bits attached to LEDs
to output
```

```
  alt_setbits_word( gpio1_swporta_ddr_addr , USER_IO_DIR );

  printf("led test\r\n");
  printf("the led flash 2 times\r\n");

  for(i=0;i<2;i++)
  {
    alt_setbits_word( gpio1_swporta_dr_addr, BIT_LED );
    usleep(500*1000);
    alt_clrbits_word( gpio1_swporta_dr_addr, BIT_LED );
    usleep(500*1000);
  }
  printf("user key test \r\n");
  printf("press key to control led\r\n");

  while(1){
    scan_input = alt_read_word(gpio1_porta_ext_addr);
    if(~scan_input&BUTTON_MASK)
      alt_setbits_word( gpio1_swporta_dr_addr, BIT_LED );
    else
      alt_clrbits_word( gpio1_swporta_dr_addr, BIT_LED );
  }

  // clean up our memory mapping and exit
  if( munmap( virtual_base, HW_REGS_SPAN ) != 0 ) {
    printf( "ERROR: munmap() failed...\n" );
    close( fd );
    return( 1 );
  }

  close( fd );
  return( 0 );
}
```

- um arquivo `Makefile` com o conteúdo a seguir:

```
TARGET = hps_gpio

ALT_DEVICE_FAMILY ?= soc_cv_av

HWLIBS_ROOT = $(SOCEDS_HWLIB)

CFLAGS = -g -Wall -Werror -I$(HWLIBS_ROOT)/include -I$
(HWLIBS_ROOT)/include/$(ALT_DEVICE_FAMILY) -D$(ALT_DEVICE_FAMILY)
LDFLAGS = -g -Wall -Werror

CROSS_COMPILE = $(GCC_Linaro)/arm-linux-gnueabihf-
```

```
CC = $(CROSS_COMPILE)gcc
LD = $(CROSS_COMPILE)gcc
ARCH = arm

.PHONY: build
build: $(TARGET)

$(TARGET): main.o
    $(LD) $(LDFLAGS)   $^ -o $@

%.o : %.c
    $(CC) $(CFLAGS) -c $< -o $@

.PHONY: clean
clean:
    rm -f $(TARGET) *.a *.o *~
```

E compile o projeto:

```
$ make
```

Copiando para o `target`

Com o cartão de memória ainda no `host`, copie o arquivo binário: `hps_gpio` para a pasta: `/root/` do cartão de memória.

Sempre que manipular um dispositivo de memória externo, será necessário fazer um flush do cache para forçar o linux alterar o dispositivo externo, caso contrário a alteração poderá ficar só local ao PC.

```
$ sync
```

> A função sync é blocante, ficará travada enquanto o linux faz o flush dos dados.

Executando no `target`

1. Executar o linux no `target`
2. Conectar `host` no `target` via terminal

Faça login no linux via terminal (user: root, ps: 1234) e execute o arquivo compilado:

```
$ /root/hps_gpio
```

E veja os LEDs piscarem ! Na verdade os LEDs piscam 2 vezes e depois o código fica verificando quando um botão (KEY2) é pressionado. **Cuidado para não apertar os botões do lado que são de reset !**

Modificando o código !

Faça o programa ler apenas duas vezes o botão, e depois disso termina a aplicação !

Entrega 4

Vamos melhorar esse nosso sistema de compilação e deploy !

- Entrega 4