Nios II プログラミングその2 デバイス操作編

Nios IIでデバイスを使うにはどうすればいいか

- ドライバを使う(用意されていれば)
- デバイスに割り当てられたレジスタをｺﾞﾆｮｺﾞﾆｮする
  （メモリマップドI/Oって言うらしい）

  (細かい用語はこっちも理解していない気にしない気にしない)
  - HAL APIで用意されているマクロ関数を使う

    (こっちはやってることはだいたい↓と同じだけどこっちのがわかりやすい)
  - ポインタでレジスタのアドレスを指定して操作する

    (こっちはコンパイラの最適化とかの問題であんま推薦されないらしい)

ドライバを使う

- ドライバで簡単な操作くらいならできます。
- 大雑把な流れは
  1. ドライバでデバイスの構造体を開く
  2. ドライバを制御する関数をつかう
- まずはDE2-115 Media Computerでパラレルポートのデバイス(7seg, LED, スライダースイッチ)の制御をしてみませふ。

  （全部スライドに貼り付けると大変なんで添付資料を見てください。）
- 添付資料はpalarell_port_test.cです。ダウンロードしてみてください。

パラレルポート制御(ドライバ使用)

```
1  #include <stdio.h>
2  #include "altera_up_avalon_parallel_port.h"
3  #include "sys/alt_irq.h"
4  #include "system.h"
5
6  /* Global variables */
7  alt_up_parallel_port_dev *green_LEDs_dev;
8  alt_up_parallel_port_dev *red_LEDs_dev;
9  alt_up_parallel_port_dev *hex3_hex0_dev;
10 alt_up_parallel_port_dev *slider_switch_dev;
11 alt_up_parallel_port_dev *push_button_dev;
12
13 //interrupt handler
14 void button_handler();
15
```

毎度おなじみ標準ライブラリ

パラレルポートのドライバ

割り込み関連

デバイスの名とかレジスタのアドレスの定数とか

使うデバイスの構造体ポインタ

割り込みハンドラ(今回は省略)

パラレルポート制御(ドライバ使用)

```
22      //open device
23      green_LEDs_dev = alt_up_parallel_port_open_dev(GREEN_LEDS_NAME);
24      if (green_LEDs_dev == NULL) {
25          printf("Couldn't open green_LEDs_dev\n");
26          return -1;
27      }
28
29      red_LEDs_dev = alt_up_parallel_port_open_dev(RED_LEDS_NAME);
30      if (red_LEDs_dev == NULL) {
31          printf("Couldn't open green_LEDs_dev\n");
32          return -1;
33      }
34
35      hex3_hex0_dev = alt_up_parallel_port_open_dev(HEX3_HEX0_NAME);
36      if (hex3_hex0_dev == NULL) {
37          printf("Couldn't open hex3_hex0_dev\n");
38          return -1;
39      }
```

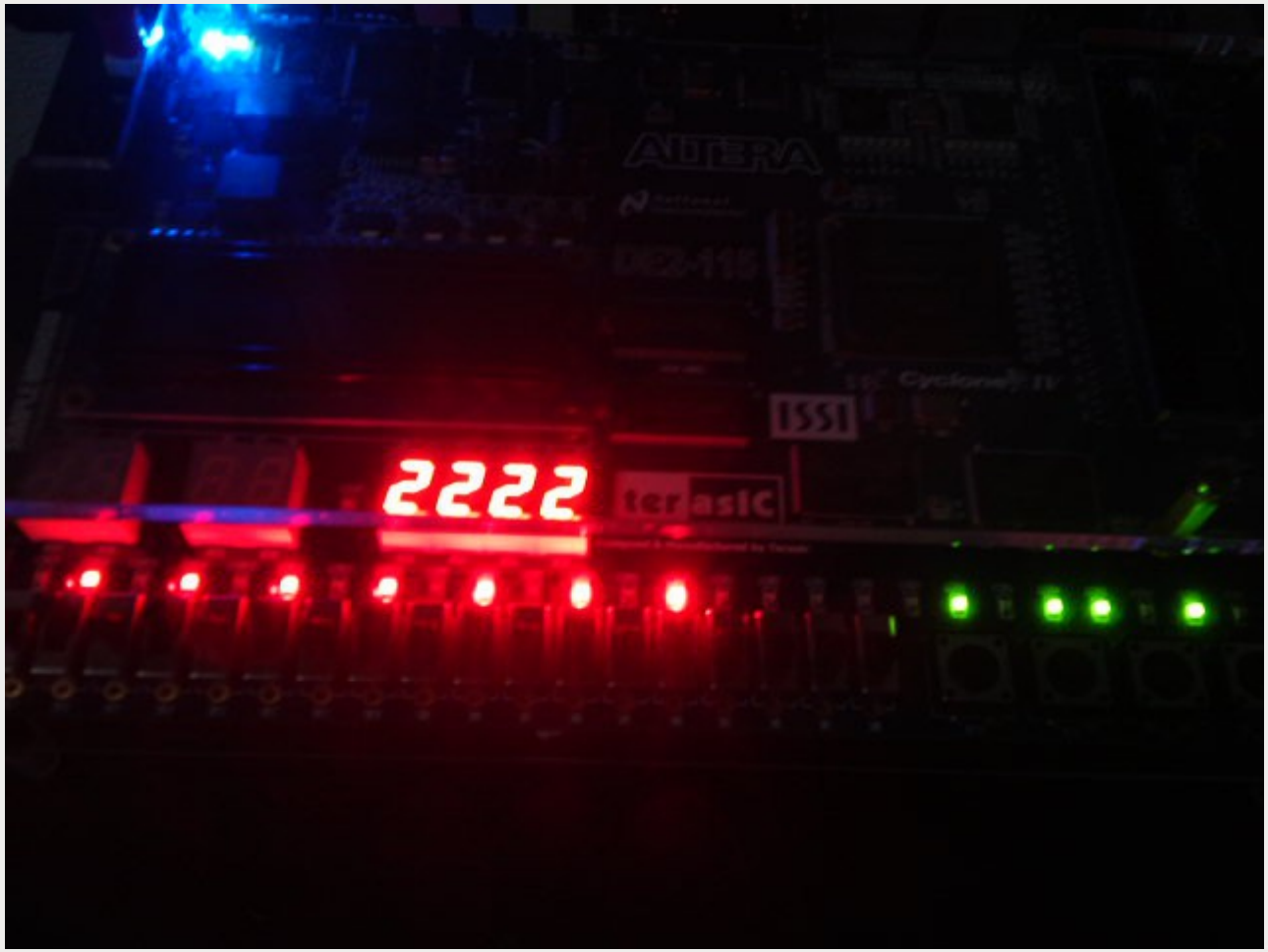
alt_up_parallel_port_open_dev()関数でデバイスを開く
引数はデバイス名を表す引数だ！
今回はマクロ定数を入れてるがこれはsystem.hに定義されてる。

パラレルポート制御(ドライバ使用)

```
54      //light green_LEDs
55      //0x5A = 0b01011010
56      alt_up_parallel_port_write_data(green_LEDs_dev, 0x5A);
57
58      //show 2222 on 7seg
59      alt_up_parallel_port_write_data(hex3_hex0_dev, 0x5B5B5B5B);
60
61      //read slider_switch_status
62      slider_switch_status = alt_up_parallel_port_read_data(slider_switch_dev);
63
64      //light red_LEDs
65      alt_up_parallel_port_write_data(red_LEDs_dev, slider_switch_status);
66
```

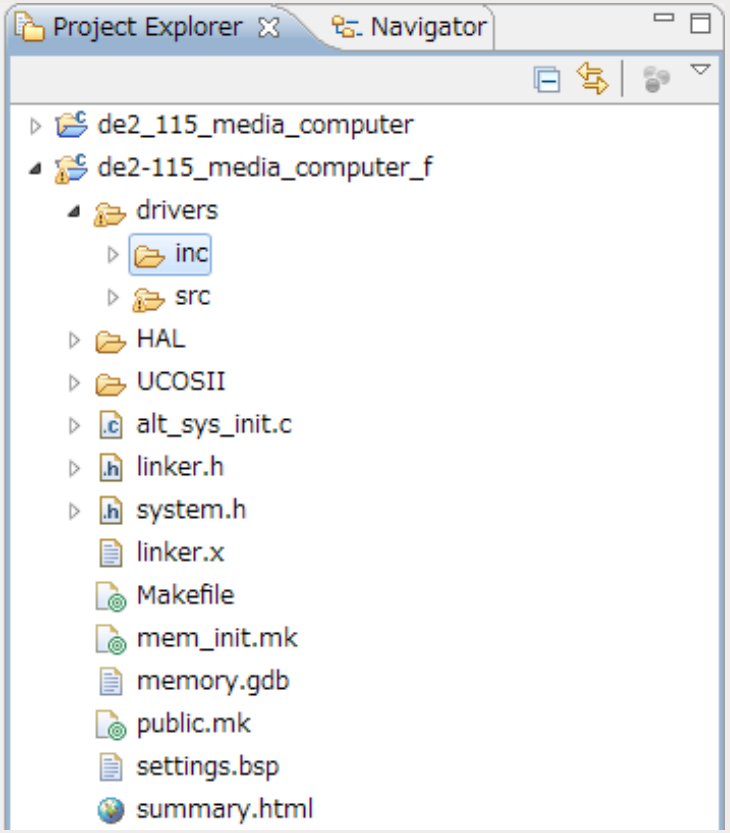
データを読んだり書いたりしてパラレルポートのデバイスを制御している。
引数に各種デバイスの構造体を渡してる。

パラレルポート制御(ドライバ使用)



実行したらこんな感じ
7segが2222って表示したりLEDが指定したとこだけ点灯してたり
(見えにくいけどプログラム実行前にスイッチを上に上げておくと赤色LEDが
点灯するようになってる)
(7segとかLEDのビット配列は各自調べてください…仕様書とかQsysの情報とか…)

で、ドライバはどこにあるの？



BSPプロジェクトにあるぞ。
ビルドするときにこいつら自動でincludeパスに追加されてるはず
なかったらパスに追加しよう(※要検索「eclipse includeパス」とかで)
ドライバは用意されてればどっかにある。なかったら直接レジスタごにょごにょしよう。

レジスタを直接いじっちゃう

- ドライバ？うんない。
- メモリマップドI/O
  - メモリ空間上にデバイスの読み書きとかする用のアドレスがあってそこの値をいじくり回せばいいらしい
  - くわしいことは知らない。調べるなり詳しい人に聞いてみて。
- 方法
  - それ用のマクロ関数があるからそれを使う。
  - 直接ポインタつかって読み書きする。
- 添付資料のparalell_port_pio_test.cも見てください(コピペ用)。

レジスタを直接いじっちゃう

```
1  #include <stdio.h>
2  #include "io.h"
3  #include "system.h"
4
5  int main(void){
6      volatile int *green_LED_addr = GREEN_LEDS_BASE;
7      int sw_val;
8
9      puts("---paralell_port_pio_test---");
10
11     //read slider_switch-bits
12     sw_val = IORD(SLIDER_SWITCHES_BASE, 0);
13
14     //light red_LEDs
15     IOWR(RED_LEDS_BASE, 0, sw_val);
16
17     //light all green LEDs
18     *green_LED_addr = 0x0FF;
19
20     return 0;
21 }
```

なにをしてるかって言うと、スライドスイッチの位置によって対応した赤色LEDが
点灯したり、緑色LEDを全部点灯させている。
どちらもデバイスのレジスタにアクセスしてるんだ。
(さっきのと同じでプログラム実行前にスイッチずらしとく必要あるけど…)

レジスタを直接いじっちゃう

```
1  #include <stdio.h>
2  #include "io.h"
3  #include "system.h"
4
5  int main(void){
6      volatile int *green_LED_addr = GREEN_LEDS_BASE;
7      int sw_val;
8
9      puts("---paralell_port_pio_test---");
10
11     //read slider_switch-bits
12     sw_val = IORD(SLIDER_SWITCHES_BASE, 0);
13
14     //light red_LEDs
15     IOWR(RED_LEDS_BASE, 0, sw_val);
16
17     //light all green LEDs
18     *green_LED_addr = 0x0FF;
19
20     return 0;
21 }
```

IORDでレジスタの値を読み込んでる。IOWRで書き込んでる。
こいつらがマクロ関数。io.hで宣言されてる。
～～_BASEみたいな定数がレジスタのアドレス。system.hで定義されてる。
超便利。わざわざ仕様書とかqsysでアドレスを確認しなくていい。

レジスタを直接いじっちゃう

```
1  #include <stdio.h>
2  #include "io.h"
3  #include "system.h"
4
5  int main(void){
6      volatile int *green_LED_addr = GREEN_LEDS_BASE;
7      int sw_val;
8
9      puts("---paralell_port_pio_test---");
10
11     //read slider_switch-bits
12     sw_val = IORD(SLIDER_SWITCHES_BASE, 0);
13
14     //light red_LEDs
15     IOWR(RED_LEDS_BASE, 0, sw_val);
16
17     //light all green LEDs
18     *green_LED_addr = 0x0FF;
19
20     return 0;
21 }
```

ポインタつかってアドレス指定して直接レジスタにアクセスしてる。
volatile修飾子でコンパイラに最適化しないようにしてるらしいけどどうも嘘くさいらしい。
だから、専用の命令のマクロ関数使いましょう。

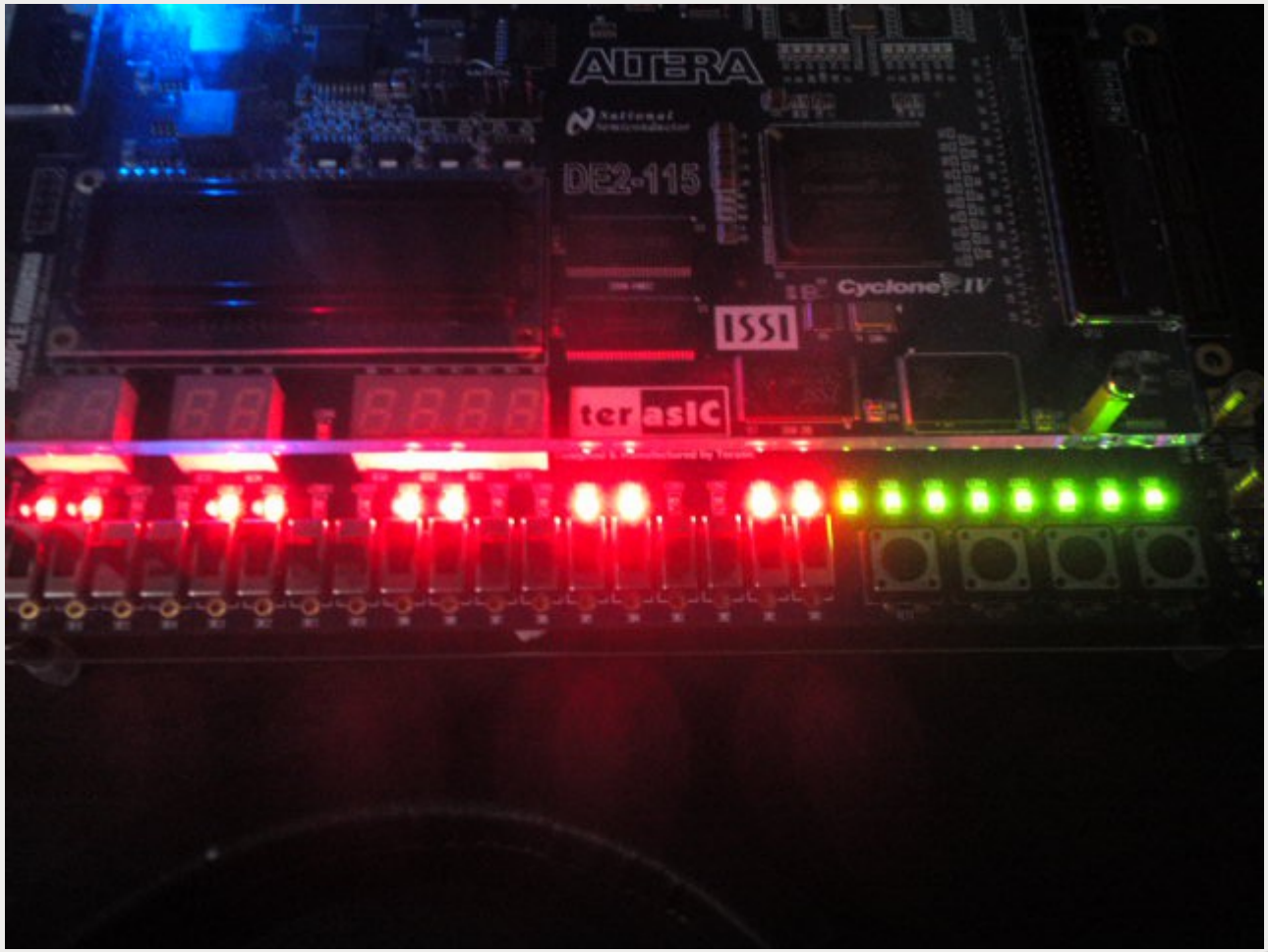
レジスタを直接いじっちゃう

```
1  #include <stdio.h>
2  #include "io.h"
3  #include "system.h"
4
5  int main(void){
6      volatile int *green_LED_addr = 0x10000010;
7      int sw_val;
8
9      puts("---paralell_port_pio_test---");
10
11     //read slider_switch-bits
12     sw_val = IORD(0x10000040, 0);
13
14     //light red_LEDs
15     IOWR(0x10000000, 0, sw_val);
16
17     //light all green LEDs
18     *green_LED_addr = 0x0FF;
19
20     return 0;
21 }
```

実際にはこー置き換わってるんだけどこれみたいに直接アドレスを打ち込むのはやめましょう。マジックナンバーっていって見た目ダサいし保守性が一気に下がるし回路仕様を変更したときにアドレスが変わってたり大変なことになります。

デスマーチ万歳

レジスタを直接いじっちゃう



さっきの実行結果とやってるのは同じ。
違いは7seg使ってないのと緑色LEDが全部点灯してるくらい。

とりあえず終わりです

- ベースアドレスってなんやねんって思うかもしれないけどレジスタの先頭アドレス(だと思う。自信ない…)

  データレジスタ→割り込みマスクレジスタ→エッジレジスタみたいな感じで並んでるんだけど細かいことは各種デバイスの資料で仕様を見てください。