演習3
フロントエンド作成 & HLS

L-III 松本直樹

ここまでの振り返り

- 論理回路を勉強した
- 回路を実装し、 Iyokan で実際に動かした
- **(今回) アプリケーションのフロントエンド作成**

フロントエンドに必要な要素

- TOML 形式の入力や出力の作成を毎回行いたくない
- データの暗号化や復号についても同様

- **ユーザーは準同型暗号や論理回路を知らない**
  → 目的に沿った分かりやすいUI
- 回路や lyokan の存在を隠ぺいする必要がある
- **CLI で引数を与えて実行 → いい感じに処理されて結果が得られる**
  ぐらいには簡単にしたい

演習3 フロントエンド作成

- 今回取り扱う例(文字の Capitalize)
- ./run.sh "enjoy your security camp!“

Input:enjoy your security camp!

Output:Enjoy Your Security Camp!

- 裏で回路の設定や秘密鍵の作成等を行う
- 入力を暗号化し、実行、結果を復号して出力する

演習3 フロントエンド作成

- **演習内容**
  - 回路 CapitalizeString.json のCLIフロントエンドを作成してください
  - 具体的には
    - TOML 形式の入力, 出力を処理
    - 鍵の生成や入力の暗号化,出力の復号(ex2 の test.sh が参考になる)

- 注意事項
  - CapitalizeString.json は Iyokan フォーマットに変換済み
  - 入出力の仕様については後述の例を参考にすること
  - 実装言語は問わない(python等のTOMLを扱いやすい言語がおすすめ)

演習3 フロントエンド作成

- 英単語の Capitalize を行う回路のフロントエンドを作成
- 処理の内容
  - スペースで区切られた単語ごとに処理
  - “one” を “One”, “two” を “Two”, “Three” を “Three”
  - “this is a pen” を “This Is A Pen”
- 文字列長の最大値は 32 文字(=32 * 8 = 256 bytes)
- 回路の “my_string” ポート(256 bit幅) に文字列を与えると “out” ポート(256 bit幅) から結果が得られる
- 処理は1サイクルで完了する

演習3 フロントエンド作成

- 処理の実装
  - C の実装を Google XLS で回路に変換

```
#define MAX_LENGTH 32

#pragma hls_top
void CapitalizeString(char my_string[MAX_LENGTH]) {
  bool last_was_space = true;
#pragma hls_unroll yes
  for (int i = 0; i < MAX_LENGTH; i++) {
    char c = my_string[i];
    if (last_was_space && c >= 'a' && c <= 'z') {
      my_string[i] = c - ('a' - 'A');
    }
    last_was_space = (c == ' ');
  }
}
```

演習3 フロントエンド作成

- 入力例(ASCII コードで入力)

```
[[bits]]
size = 256
name = "in"
bytes = [ 101, 110, 106, 111, 121, 32, 121, 111, 117, 114, 32, 115, 101, 99, 117, 114, 105, 116, 121, 32, 99, 97, 109, 112, 33,]
```

e n j o y y o u r s e c u r i t y c a m p !

- 出力例(ASCII コードで出力)

```
rom = []
ram = []
cycles = 1
[[bits]]
bytes = [69,110,106,111,121,32,89,111,117,114,32,83,101,99,117,114,105,116,121,32,67,97,109,112,33,0,0,0,0,0,0,0,]
size = 25
name = "
```

E n j o y Y o u r S e c u r i t y C a m p !

演習3 フロントエンド作成

- 回路の設定ファイル例
- 詳細な設定方法については Iyokan のリポジトリを参照

```
[[file]]
type = "iyokanl1-json"
path = "CapitalizeString_converted.json"
name = "CapString"

[connect]
"CapString/my_string[0:255]" = "@in[0:255]"
"@out[0:255]" = "CapString/out[0:255]"
```

- ここから先は **付録** なので読むだけで構いません

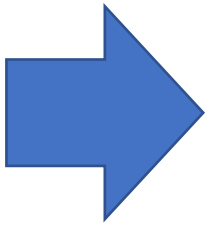
付録: XLS

- XLS : Google による 回路合成ソフトウェア
- XLS の experimental な要素として HLS(高位合成)機能を持つ
- C++ などで記述された処理を Verilog 等の回路記述に変換
- 記述には特有の作法がある

```
#define MAX_LENGTH 32

#pragma hls_top
void CapitalizeString(char my_string[MAX_LENGTH]) {
  bool last_was_space = true;
#pragma hls_unroll yes
  for (int i = 0; i < MAX_LENGTH; i++) {
    char c = my_string[i];
    if (last_was_space && c >= 'a' && c <= 'z') {
      my_string[i] = c - ('a' - 'A');
    }
    last_was_space = (c == ' ');
  }
}
```

HLS
(高位合成)



```
module CapitalizeString(
  input wire [255:0] my_string,
  output wire [255:0] out
);
  wire [7:0] my_string_unflattened[32];
  assign my_string_unflattened[0] = my_string[7:0];
  assign my_string_unflattened[1] = my_string[15:8];
  assign my_string_unflattened[2] = my_string[23:16];
  assign my_string_unflattened[3] = my_string[31:24];
  assign my_string_unflattened[4] = my_string[39:32];
  assign my_string_unflattened[5] = my_string[47:40];
  assign my_string_unflattened[6] = my_string[55:48];
  assign my_string_unflattened[7] = my_string[63:56];
  assign my_string_unflattened[8] = my_string[71:64];
  assign my_string_unflattened[9] = my_string[79:72];
  assign my_string_unflattened[10] = my_string[87:80];
```

XLS のビルド

- bazel(Google のビルドツール) の導入
  - sudo apt install apt-transport-https curl gnupg
  - curl -fsSL https://bazel.build/bazel-release.pub.gpg | gpg --dearmor > bazel.gpg
  - sudo mv bazel.gpg /etc/apt/trusted.gpg.d/
  - echo “deb [arch=amd64] https://storage.googleapis.com/bazel-apt stable jdk1.8” | sudo tee /etc/apt/sources.list.d/bazel.list
  - sudo apt update && sudo apt install bazel-4.0.0

```
ubuntu@xls:~$ curl -fsSL https://bazel.build/bazel-release.pub.gpg | gpg --dearmor > bazel.gpg
ubuntu@xls:~$ sudo mv bazel.gpg /etc/apt/trusted.gpg.d/
ubuntu@xls:~$ echo "deb [arch=amd64] https://storage.googleapis.com/bazel-apt stable jdk1.8" | sudo tee /etc/apt/sources
.list.d/bazel.list
deb [arch=amd64] https://storage.googleapis.com/bazel-apt stable jdk1.8
ubuntu@xls:~$ sudo apt update && sudo apt install bazel-4.0.0
Get:1 https://storage.googleapis.com/bazel-apt stable InRelease [2256 B]
Get:2 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Hit:3 http://archive.ubuntu.com/ubuntu focal InRelease
```

XLS のビルド

- 約 4000 ファイルのビルドを行うため 30 〜 60分ほどかかる(32C/64T マシンで約10分)
- 10GB 前後の空き容量が必要
- 依存パッケージのインストール
  - sudo apt install gcc git libtinfo5 python python3 python3-pip
- XLS のビルド
  - git clone --recursive https://github.com/google/xls
  - cd xls
  - bazel-4.0.0 build -c opt //xls/contrib/xlscc:xlscc //xls/tools:opt_main //xls/tools:codegen_main

```
ubuntu@xls:~$ cd xls/
ubuntu@xls:~/xls$ bazel-4.0.0 build -c opt //xls/contrib/xlscc:xlscc //xls/tools:opt_main //xls/tools:codegen_main
Extracting Bazel installation...
Starting local Bazel server and connecting to it...
```

XLS による回路の合成

- bazel-bin/xls/contrib/xlscc/xlscc CapitalizeString.cc > CapitalizeString.ir
- bazel-bin/xls/tools/opt_main CapitalizeString.ir > CapitalizeString.opt.ir
- bazel-bin/xls/tools/codegen_main CapitalizeString.opt.ir --generator combinational > CapitalizeString.v
- 生成された verilog ファイルを ex2/tests/build.sh に渡せば Iyokan 用の回路が出来る

```
ubuntu@xls:~/xls$ bazel-bin/xls/contrib/xlscc/xlscc CapitalizeString.cc > CapitalizeString.ir
Parsing file 'CapitalizeString.cc' with clang...
Generating IR...
ubuntu@xls:~/xls$ bazel-bin/xls/tools/opt_main CapitalizeString.ir > CapitalizeString.opt.ir
ubuntu@xls:~/xls$ bazel-bin/xls/tools/codegen_main CapitalizeString.opt.ir --generator combinational > CapitalizeString.
v
```