Контроль качества

МФТИ
Весна 2025

Обсуждение

Какую программу вы назовете качественной?

Критерии качественной программы

- Может выполнить прямые функции
- При возникновении ошибки выдает *понятную* диагностику
- Может обработать *любые* входные данные
- Работает согласно *требованиям*

- *Хорошо* спроектирована

Обсуждение

Как определить *качественность* программы?

Метрики

“Вы не можете контролировать то, что не можете измерить” –

Том Демарко

Обсуждение

Какие метрики *кода* вы знаете?

Метрики кода

- Объем кодовой базы (количество строк, файлов, классов и тд.)
- Количество строк на один assert
- Время сборки
- Количество тестов

Обсуждение

Какие метрики *качества кода* вы знаете?

Метрики качества кода

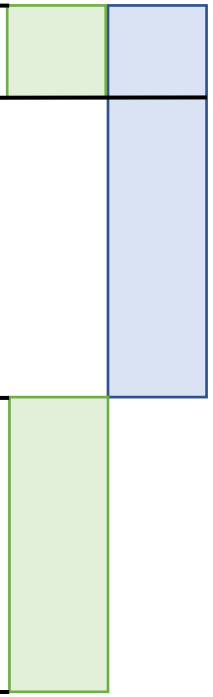
- Pass rate
  - Количество/процент проходящих тестов
  - Pass / Failed / Total
- Code coverage
  - Насколько написанные тесты «покрывают» исходный код
- Defect Leakage
  - Количество багов, не выявленных на этапе тестирования, но обнаруженных пользователями после релиза

Code coverage

- Function
  - Процент посещенных функций
- Statement
  - Процент посещенных statement’ов
  - По умолчанию подразумевается этот вид покрытия
- Edge
  - Процент выполненных комбинаций бранчей от всех возможных
- Condition
  - Процент выполненных комбинаций всех булевых значений программы

Пример: Statement Coverage Problem

```
void do_action(int n) {
  char *ptr = nullptr;
  if (n % 2 == 0) {
    ptr = “ololo”;
  }
  if (n % 3 == 0) {
    printf(“%s\n”, ptr);
  }
}
```



Test 1:

```
do_action(2);
```

Test 2:

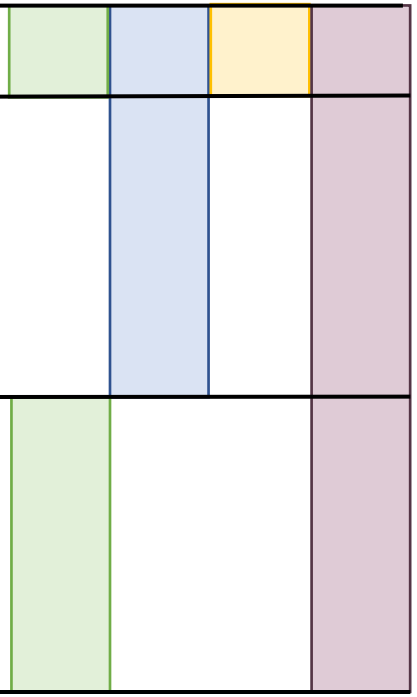
```
do_action(3);
```

Coverage report: 100%

Покрытие идеальное, но присутствует очевидный баг

Пример: Edge Coverage

```
void do_action(int n) {
  char *ptr = nullptr;
  if (n % 2 == 0) {
    ptr = “ololo”;
  }
  if (n % 3 == 0) {
    printf(“%s\n”, ptr);
  }
}
```



Test 1:

```
do_action(2);
```

Test 2:

```
do_action(3);
```

Test 3:

```
do_action(5);
```

Test 4:

```
do_action(6);
```

Coverage report: 100%

Баг будет найдет при 100% Edge Coverage

Повторение: Как разрабатывают аппаратуру?

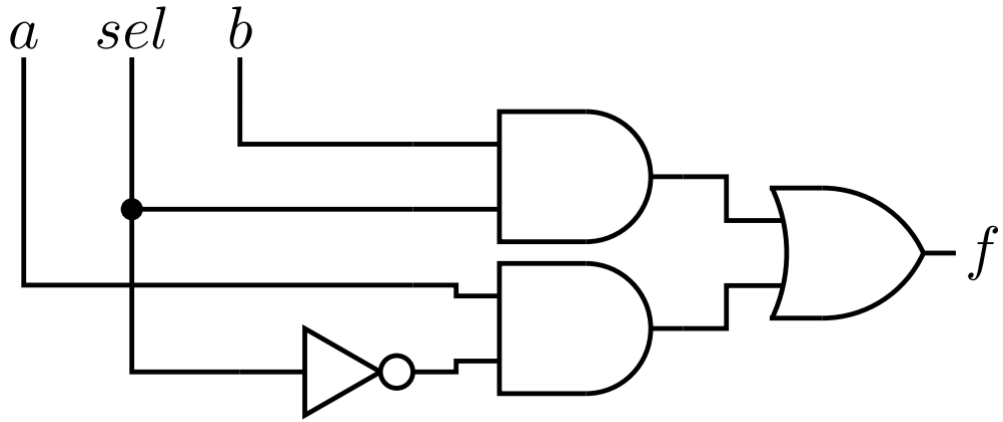
Микроархитектуру разрабатывают на языке описания аппаратуры (**HDL** – ***Hardware Description Language***)

Описание микроархитектуры так же часто называют **RTL** (***Register-Transfer Level***)

Из **RTL** *синтезируют* цифровую схему

Повторение: Пример RTL модуля на SystemVerilog

```
module mux (
      input  logic a, b, sel,
      output logic f
);
      logic n_sel, f1, f2;
      assign sel = ~n_sel;
      assign f1 = a & n_sel;
      assign f2 = b & sel;
      assign f = f1 | f2;
endmodule
```

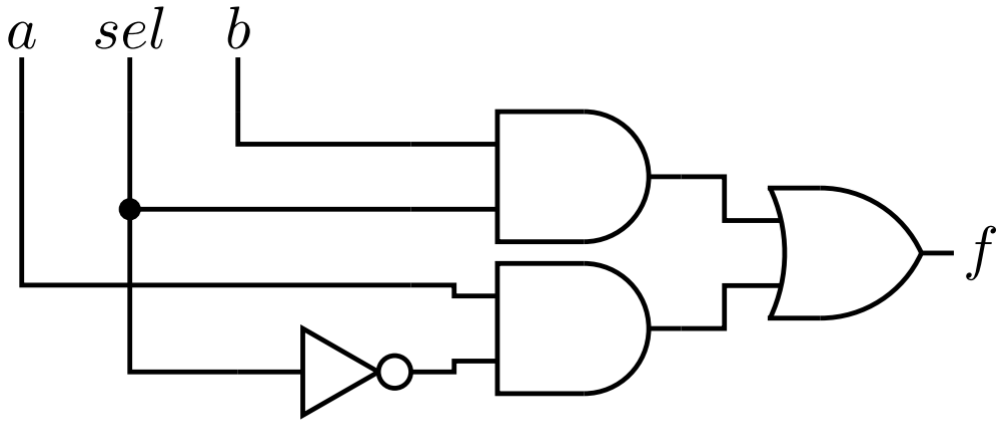


$f = sel\ ?\ b : a$

Повторение: Пример RTL модуля на SystemVerilog

```
module mux (
      input  logic a, b, sel,
      output logic f
);
      logic n_sel, f1, f2;
      assign sel = ~n_sel;
      assign f1 = a & n_sel;
      assign f2 = b & sel;
      assign f = f1 | f2;
endmodule
```

Найдите ошибку на слайде



$f = sel\ ?\ b : a$

Покрываем тестами RTL

- Современные процессоры представляют собой чрезвычайно сложные системы систем с несколькими уровнями абстракции
- Полное покрытие кода (RTL описания) уже при среднем дизайне становится едва решаемой задачей
- Основная проблема сложности покрытия вызвана большим количеством внутренних состояний
- Как это можно решить/компенсировать эту проблему?

Функциональное покрытие

- Зафиксируем только те состояния, которые соответствуют определенному внешнему поведению системы
- Покроем только этот выбранный набор состояний
- Такое покрытие называется *функциональным*

Тестовый генератор llvm-snippy

- Создает ассемблерные сниппеты с инструкциями с заданным случайным распределением
- Использует инфраструктуру LLVM
- Спроектирован кроссплатформенным
- Имеет встроенную возможность создавать трассы с помощью моделей
- Имеет возможность рандомизировать CF и вызовы функций
- Может вставлять код самопроверки корректности исполнения

syntacore/snippy

To be continued …

На следующем занятии

- Познакомимся подробнее с llvm-based генератором snippy