DESIGN OF DIGITAL CIRCUITS AND SYSTEMS

Proposal Workshop

Instructor: Justin Hsia

Teaching Assistants:

Colton Carroll
Hemil Patel
Rasya Fawwaz
Grace Zhou
Quinlyn Donohue
Rose Maresh

Relevant Course Information

- Quiz 5 (50 min) is *this* Thursday @ 11:**3**0 AM
  - Static Timing Analysis, Pipelining, Clock Domain Crossing
  - Scientific calculator allowed!
- Homework 6 due Monday (6/1)
- Lab 6 proposal due tomorrow (5/27)
  - (1) Description of major project features
  - (2) Top-level block diagram
  - (3) Images/sketches of VGA output
- Lab 6 report and video due 6/8

Randomization Review Questions

- What is the difference between `rand` & `randc`?

  rand is "uniformly" distributed (like rolling a die)

  randc is cyclically distributed (like drawing cards without replacement)

- How do you randomize an object? What happens when this fails?

```
MemRead mr = new();
mr.randomize(); // returns 0 on failure
```

- Define *random stability*.

  separate randomization calls produce different results,
  but separate testbench executions will produce the same results for the same seed.

  can change seed using `mr.srandom(<seed>);`

- Name a reason one might want to split constraints into multiple `constraint` blocks.

  readability - use one constraint block per randomizeable variable

Ranges and Sets

- `[A:B]` declares a **range** of integers between A and B, inclusive
  - Just like the notation used in array declarations
  - A and B can be constants and/or variables

[0:4] [min:max]

- A random variable can be chosen from a **set** of values using the `inside` keyword

{}

  - Can be used with both `rand` and `randc` variables
  - Sets can notated as the concatenation of values, ranges, and array variables
  - *e.g.,*

random fibonacci number

set

1,2,3, 5,8,13, 21

```
rand bit [7:0] f;
bit [7:0] vals[] = {5, 8, 13};
constraint c_fib { f inside {[1:3], vals, 21}; }
```

range array value

Weighted Distributions

- You can define a weighted non-uniform distribution with the constraint expression
  `<var> dist {<distribution>};`
  - Can only be used with `rand` variables

- Distribution notated in comma-separated list of values and their relative weights
  - Values can be expressed by themselves or in a range or set
  - Weights in distribution become normalized (*i.e.*, don't have to sum to 100)
  - *e.g.*,

```
constraint c_weight { coin dist {0:=5, 1:=5}; }
```

random variable
values
weights
(total weight 10)

Weighted Distributions

- Weight distribution operators for ranges and sets
  - `:=` assigns same weight to multiple values
  - `:/` distributes the assigned weight across multiple values
- Example:

```
constraint c_dist1 {
   x dist {0:=30,
           [1:3]:=30};
}
```

```
constraint c_dist2 {
   x dist {0:/30,
           [1:3]:/30};
}
```

no difference

each gets 30

all split 30

weight	x	Probability
30	0	1/4
30	1	1/4
30	2	1/4
30	3	1/4
120		

weight	x	Probability
30	0	1/2
10	1	1/6
10	2	1/6
10	3	1/6
60		

Constraint Exercise #2

- Modify your `MemRead` class from Exercise #1 to have the following updated constraints:
  - Constrain `data` to always be `5`
  - Constrain `addr` to probabilistically be `4'd0` 10% of the time, `4'd15` 10% of the time, and between those two the rest of the time (80%)

```
constraint c_data {
    data == 5;
}
constraint c_addr {
    addr dist { 0:=10, 15:=10, [1:14]:/80};
}
```

Constraints with Variables (1/2)

- Instead of hardcoding constraints, use variables with default values
  - Avoid magic numbers; code becomes more readable
  - Can change before performing randomization

```
class Packet;
   rand bit [31:0] length;
   constraint c_len {
      length inside [1:100]};
   }
endclass
```

```
class Packet;
   rand bit [31:0] length;
   int max_len = 100;
   constraint c_len {
      length inside [1:max_len]};
   }
endclass
```

Constraints with Variables (2/2)

- Instead of hardcoding constraints, use variables with default values
  - Avoid magic numbers; code becomes more readable
  - Can change before performing randomization

```
class Packet;
   rand bit [31:0] length;
   constraint c_len {
      length inside [1:100]};
   }
endclass
```

```
class Packet;
   rand bit [31:0] length;
   int max_len = 100;
   constraint c_len {
      length inside [1:max_len]};
```

```
initial begin
   Packet p1 = new();
   p1.max_len = 200;
   if (!p1.randomize())
      $finish;
end
```

BONUS SLIDES

Implication and Equivalence Operators are included here as additional (and more complex) constraint operators.

You are *not* expected to study or need to use these in the context of this class.

Implication and Equivalence Operators

- Implication: `A->B`
  - *Same meaning, but different syntax from assertions!*
    - Equivalent to (!A || B)
  - When used in a constraint, the solver will pick values such that the implication holds true

A	B	A->B
F	F	T
F	T	T
T	F	F ✗
T	T	T

not allowed as part of a valid solution

- Equivalence: `A<->B`
  - Bidirectional implication: `(A->B) && (B->A)`
    - Equivalent to XNOR
  - Possible confusion that == also sometimes referred to as an "equivalence operator," but these are different

A	B	A<->B
F	F	T
F	T	F ✗
T	F	F ✗
T	T	T

Solution Probabilities

```
rand bit x;          // 0, 1
rand bit [1:0] y;    // 00, 01, 10, 11
```

// unconstrained!

x	y	Probability
0	0	1/8
0	1	1/8
0	2	1/8
0	3	1/8
1	0	1/8
1	1	1/8
1	2	1/8
1	3	1/8

Implication and Equivalence Examples (1/2)

```
rand bit x;
rand bit [1:0] y;
constraint c_imp1 {
    (x==0)->(y==0);
}
```

when x is 0, y must be 0

x	y	Probability
0	0	1/2
0	1	0
0	2	0
0	3	0
1	0	1/8
1	1	1/8
1	2	1/8
1	3	1/8

50% total

50% total

```
rand bit x;
rand bit [1:0] y;
constraint c_imp2 {
    y > 0;
    (x==0)->(y==0);
}
```

since y cannot be 0, neither can x

x	y	Probability
0	0	0
0	1	0
0	2	0
0	3	0
1	0	0
1	1	1/3
1	2	1/3
1	3	1/3

Implication and Equivalence Examples (2/2)

```
rand bit x;
rand bit [1:0] y;
constraint c_eqv1 {
    (x==0)<->(y==0);
}  // pick x first
```

A → B
$\bar{A}$ → $\bar{B}$

x	y	Probability
0	0	1/2
0	1	0
0	2	0
0	3	0
1	0	0
1	1	1/6
1	2	1/6
1	3	1/6

50% total (A)

50% total ($\bar{A}$)

same valid solutions, but different probabilities

```
rand bit x;
rand bit [1:0] y;
constraint c_eqv2 {
    (x==0)<->(y==0);
}  // pick y first
```

B → A
$\bar{B}$ → $\bar{A}$

x	y	Probability
0	0	1/4
1	0	0
0	1	0
1	1	1/4
0	2	0
1	2	1/4
0	3	0
1	3	1/4

25% total (B)

25% total ($\bar{B}$)

25% total ($\bar{B}$)

25% total (B)

TECHNOLOGY

BREAK

Lab 6 Proposal Workshop

- Rough schedule:
  - Pairings (1) 11:20 – 11:35, (2) 11:35 – 11:50, (3) 11:50 – 12:05, (4) 12:05 – 12:20

- Notes:
  - Make sure that you introduce and talk about *both* projects
  - Be curious – ask questions, point out potential issues, and share ideas!
    - Requirements: VGA usage? "Significant" memory? User input? Digital logic complexity?
    - What do you think the major blocks/components are?
    - Any cool extensions or pivots you can think of?
  - Course staff will be circling to listen in and answer questions