1. An answer to all questions in the lab (these will be in italics)

What did you change about the provided configurable ring-oscillator?
We doubled the size so that it used two CLBs and therefore had more delays built into it.
Why will a 50 MHz clock work for this design?
Because it is fast enough to ensure there is enough resolution in the frequency measurements to make meaningful comparisons.
Would any arbitrary clock speed work for this design?
A clock speed that is too slow would have insufficient resolution to detect differences in frequencies of the ring oscillators.
What did you decide to use for the max value of std_counter?
0xffffff
How does this value impact the PUF?
The larger this value is, the greater opportunity there is for variation in the frequencies of each of the ring oscillators.
Why would we want to know if the challenge response is complete?
So that we know when to begin performing operations (such as the sha algorithm) on the challenge response pair.
What is the purpose of this (why might we want to hash the challenge concatenated with the response)?
There's a chance that two or more challenges produce the same response, so just hashing the response would produce less unique outputs. Concatenating the two will create a unique input to the hash. This will also make it more difficult for machine learning algorithms to model our PUF.
Are the challenge-response pairs the same for each location?
No
Should they be (include details)?
No because each piece of hardware has natural differences due to the manufacturing process causing differences in delays.
What are the intra-board and inter-board hamming distances?
The intra-board hamming distance is the number of bits that differ between two responses for a given challenge on a given board, while the inter-board hamming distance is the number of bits that differ between two responses for a given challenge on two different boards.
What are ideal?

Ideally, intra-board hamming distances should be 0% so that a single challenge always produces the same response, while inter-board hamming distances should be 50% so that every PUF is as unique as possible.

Discuss anything of note about your particular implementation or results.

Our implementation of our ring oscillator differs from the recommended layout. Instead of using four modules of four ring ocilators our implementation only used a total of nine ring ocilators. Since every ring oscillator was configured with eight bits of input every challenge resulted in a different configuration of the ring oscillator. Thus every bit of the challenge is correlated to every bit of the output. In order to generate the eight bits of output we compare adjacent ring ocilators and compare there frequencies.

- ***How difficult would it be to expand the number of challenge bits and the bits of the response? Describe the process and any considerations needed to ensure the PUF functions correctly.***

To use more challenge bits we would need more ring oscillators. Modifying the code to add ring oscillators would be quick. We would increase the value of a constant, and if we were adding more than 6 ring oscillators we would increase the width of a counter. Finally we would place the new ring oscillators in p-blocks, so that the randomness isn't systematic.

- ***How might variations in temperature and voltage of the chip affect the PUF? Think both raw entropy and the result of the hash. What are some methods which could mitigate the effect?***

Variations in temperature and voltage can both change the PUF's response to a given challenge. This is difficult to mitigate because if the PUF was more stable it would likely be less random.

I don't think the PUF would be more random at stable temperatures or voltages higher or lower than typical. Temperature and voltage will cause the behavior of each of the devices in the PUF to change in a similar way. If the temperature and voltage are fluctuating randomly this would probably add randomness. Because of this, to create a stable PUF, insulation and voltage regulators might help.

- ***You might have noticed in the paper and in our implementation the hamming distance between adjacent challenges is fairly minimal (15% in Xin et. Al). What might cause this? Can you think of a way of changing how we use the counter output to increase the hamming distance? Hint: This might mean reducing the number of usable response bits in the counter.***

RO's allow for multiple challenge response pairs by adjusting the path a signal takes through the oscillator. Each challenge bit changes one stage in the path.

Therefore, challenges with low hamming distance will result in signals taking similar paths through the ROs, so it is likely that one RO is faster than the other in both cases.

One method which may mitigate this effect would be to design a scheme where a single challenge bit affects multiple parts of the RO. For example, the first RO mux could be controlled by the XOR of the first, second, and third challenge bits; and the second RO mux could be controlled by the XOR of the second, third, and fourth challenge bits.

How could this implementation be modified on a board so that it could be verified from an external source and ensure freshness? (Don't think about the BASYS 3 board, instead consider the fundamental structure of the device, including generic inputs and outputs)

The challenge and response bits would have to be accessible, then another device could query every challenge multiple times and record the response to find the most consistent challenges. If there are too many challenges to search exhaustively, a random search might work best. The hardware allowing the response bits to be read would have to be disabled otherwise an attacker could build a dictionary of CRPs and impersonate the device.

2. A description of approaches taken, problems encountered, and techniques used to overcome these challenges.

Building the PUF

The central challenge in designing a PUF is that Vivado really doesn't want to let you. Because PUFs are such strange devices, the ability to construct a PUF with Vivado, albeit after extensive finangling, shows that Vivado is surprisingly flexible. We found that when we try to describe the PUF in system verilog, Vivado does not implement each PUF in the same way. It uses different registers and LUTs in different locations. These differences would reduce randomness in the PUF. One PUF may always be faster than another PUF because of the way that it was implemented, regardless of the challenge bits or the device. To solve this problem we directly instantiated the hardware in each logic block, then used pblocks to confine each RO to exactly one logic block, and finally each RO uses exactly the amount of available hardware in a logic block. Even with these constraints we found that Vivado would still implement ROs slightly differently sometimes. To force Vivado to implement each RO exactly the same, we fixed the hardware in one RO in place; this generates a set of constraints. We then copied, and modified, these constraints for each other RO.

3. A working PUF challenge response function based on the design described in the “PUF Design” Section.
4. An analysis of the behavior of your produced design on three different locations on the FPGA.
   A. In your original configuration, you should see that challenges with low hamming distance end up producing the same or similar responses. Find 5 challenges that give very different responses in this original configuration.
   B. Document the output of the PUF across all these 5 challenges in each of the three configurations and determine the average hamming distance between the responses for the three locations.

Location 1: X4Y49 and X5Y49 through X4Y41 and X5Y41

Location 2: X24Y88 and X25Y88 through X24Y80 and X25Y80

Location 3: X22Y149 and X23Y149 through X22Y141 and X23Y141

Challenge	**Location 1**		**Location 2**		**Location 3**	
	Response	**Hamming Distance**	**Response**	**Hamming Distance**	**Response**	**Hamming Distance**
0x0b	0xc4	6	0xc5	5	0xdf	4
0x2b	0xa6	4	0xc4	7	0xdf	5
0x4b	0xb7	6	0x97	5	0xdf	3
0x52	0xaa	5	0x8a	4	0xdd	5
0xff	0xac	4	0xa7	3	0xf0	4

Average Hamming Distance		5		4.8		4.2