LAB Assignment #4 for Hardware/Software Codesign with FPGAs

Assigned: Mon., Oct 28, 2013
Due: Mon., Nov. 18, 2013

Description: Implement a Bit Generation Engine

Top level signals (ports in your entity) include:

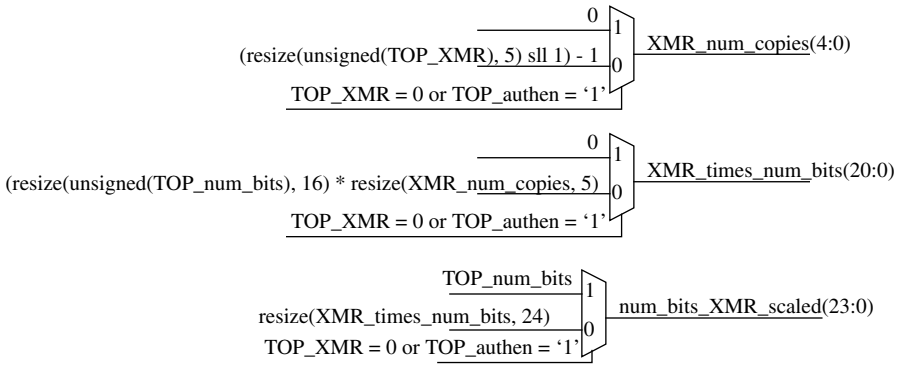
1. clk: 1-bit input signal which drives the sequential elements
2. RESET: 1-bit input signal that resets all sequential elements (synchronously or asynchonously -- up to you). 'ready_reg', 'bit_gen_success_reg' and 'equal_toggle_reg' should all be initialized to '1', and the other registers to 0 when a RESET is received.
3. TOP_enroll & TOP_authen: 1-bit input flags when '1', do enrollment or authentication, else do regeneration.
4. volt_diff_f, volt_diff_s: These are the outputs from your SamAnalysis engine (and inputs here) for two different SMCs. They are 11-bits or whatever you chose to do in SamAnalysis.
5. TOP_flip_frequency: An 8-bit unsigned value that indicates how often the comparison is flipped.
6. VDT_true_threshold: An 11-bit unsigned value that gives the threshold. Note, this width of this value is the same as the volt_diff_f/volt_diff_s signals.
7. TOP_XMR: A 4-bit unsigned value that indicates the XMR level.
8. TOP_num_bits: A 24-bit input signal indicating the total number of bits to generate.
9. TOP_valid_bit_ready, TOP_invalid_bit_ready, TOP_redundant_bit_ready: 1-bit output flags indicating the status of the current comparison.
10. TOP_bit_gen_done: A 1-bit output signal that indicates bitstring generation process is completed.
11. TOP_bstr_cur_len: A 24-bit input signal that indicates the total number of bits generated so far.
12. TOP_bstr_buf_ready_int: A 1-bit input signal indicating that 32-bits have been generated (of the 256, for example). You need to store the bits as they are generated by the BitGen engine in an interface register and assert this signal when you reach a multiple of 32-bits. When asserted, this causes the BitGen machine to 'wait' for the C program to indicate it has read the bits. This is done by asserting 'TOP_bstr_buf_continue'.
    (NOTE: TOP_bstr_buf_ready_int needs to be asserted by your VHDL code in the parent module in the same clock cycle that the BitGen engine asserts TOP_valid_bit_ready, i.e., it is a combinational circuit)
13. TOP_bstr_buf_ready: A 1-bit output signal that tells the C code that 32-bits are ready and are ready for transfer to the microprocessor memory and output over the serial link.
14. TOP_bstr_buf_continue: 1-bit input signal which is asserted by the C program to indicate that it has read the 32-bit bitstring register.
15. TOP_secret_bit: A 1-bit output signal that represents the secret bit just generated.
16. start: A 1-bit input signal which starts the BitGen engine
17. ready: A 1-bit output signal which indicates that the BitGen engine is in the 'idle' state.
18. init_state: A 1-bit input signal that resets a selected set of registers (Note: RESET also resets all of these registers + some others not listed).

You need to write the state machine that manages the public data. The signals that interface to the BitGen engine are as follows:

19. PDMC_ready: A 1-bit input from the Public Data Memory Controller (PDMC) indicating it is ready to carry out an operation. Only writes are issued by the BitGen engine.
20. PDMC_last_bit: A 1-bit output signal that tells the PDMC engine that this is the last write to public data and to store the partial word if the number of public data bits is not a multiple of 8, else just save a full word as usual.
21. PDMC_start: A 1-bit output signal to the PDMC to begin a read or write cycle.
22. PDMC_write_bit: A 1-bit output signal to the PDMC to carry out a write cycle.
23. PDMC_vb_in_store: A 1-bit output signal to the PDMC which indicates the bit to store.

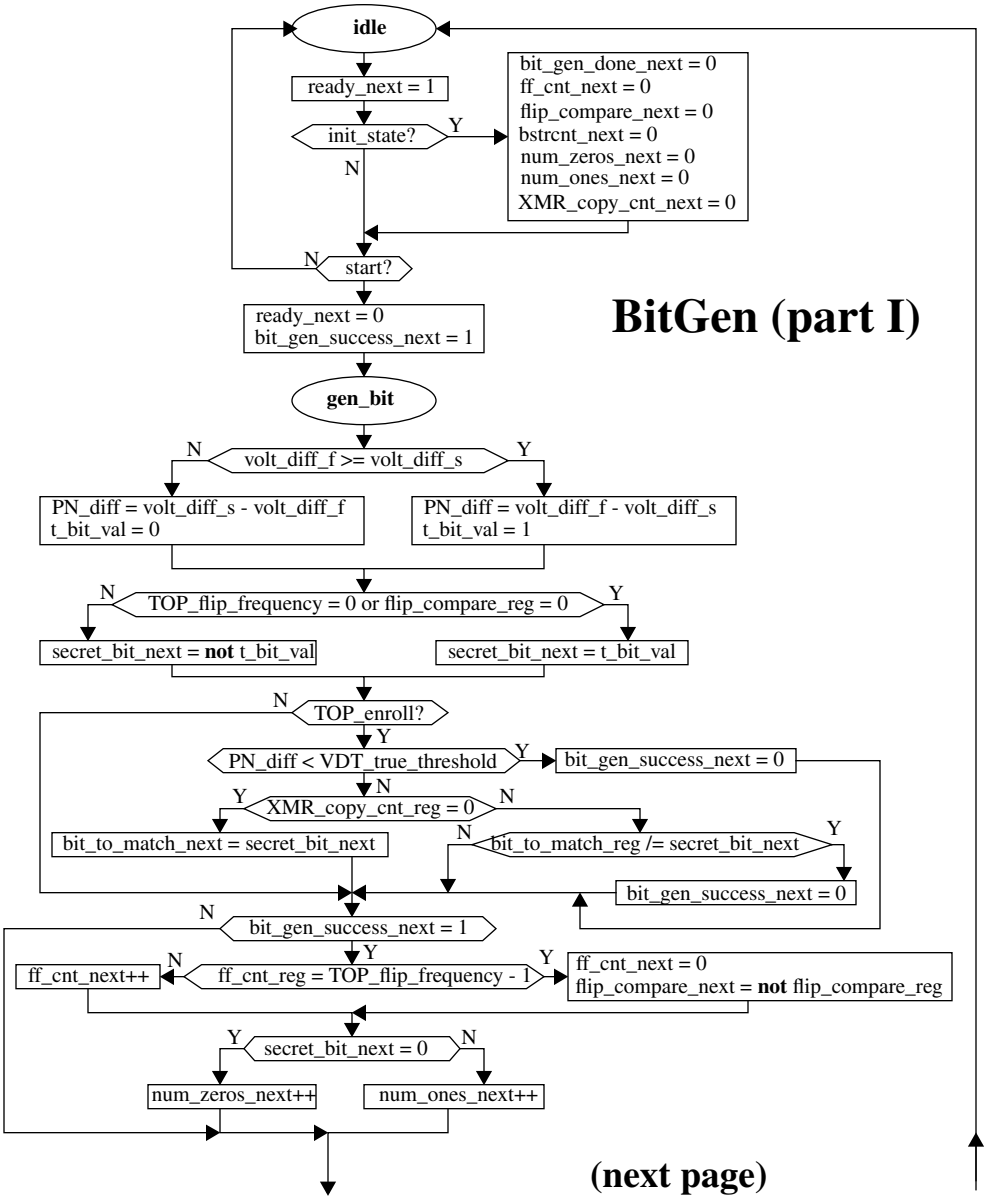
Your demo needs to show the secret bitstring generated, in 32-bit chunks as indicated by the TOP_bstr_buf_ready signal described above. You also need to store the public data as it is generated by the BitGen engine during enrollment so that it can be 'played back' during regeneration. Remember, during regeneration, you do NOT start the BitGen engine when the public data valid bit for a given comparison is '0'.

You do not need to include the PDMC FSMD in this lab. Use C code to emulate the LFSR and SamAnalysis engine as we discussed in class.




```
ready <= ready_reg;
PDMC_vb_in_store <= bit_gen_success_reg;
TOP_bstr_cur_len <= bstrcnt_next;
TOP_secret_bit <= secret_bit_next;
TOP_bstr_buf_ready <= bstr_buf_ready_reg;
```

ASMD diagram



BitGen (part II)