Digital System Design

by
Dr. Lesley Shannon
Email: lshannon@ensc.sfu.ca
Course Website: http://www.ensc.sfu.ca/~lshannon/courses/ensc350

Simon Fraser University

Slide Set: 5
Date: February 2, 2009

Slide Set Overview

- Lab 2
  - DES decryption circuit
  - Testbed

DES

- For lab 2, you will implement a structural design of a Data Encryption Standard (DES) decryption circuit and an accompanying testbed
- In this slide set, we'll talk about:
  - the DES standard
  - Testbed implementation

DES

- This lab is a ***lot*** of work
  - That's why you've got 3 weeks
- Don't procrastinate or you ***will*** regret it

DES

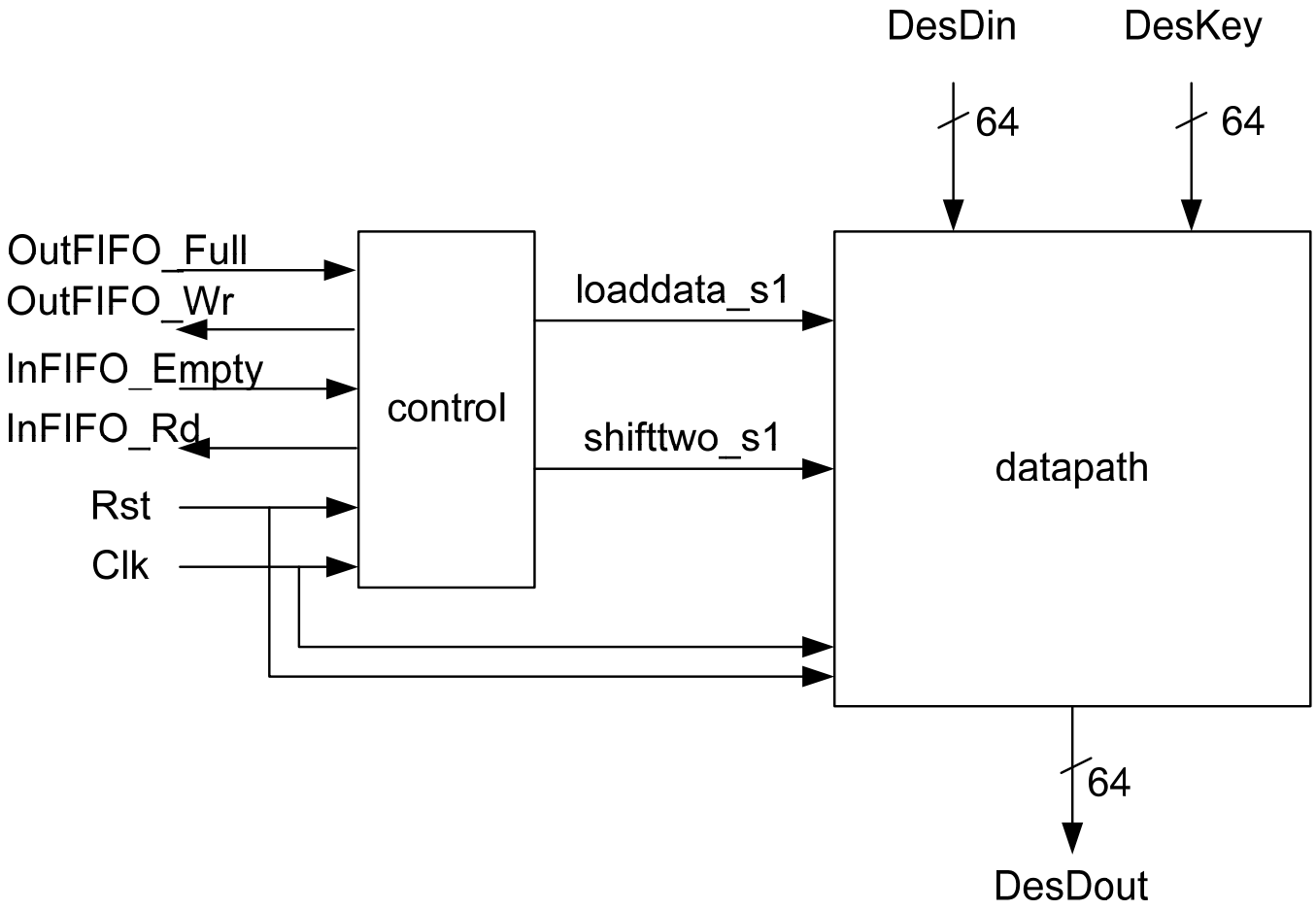
- Your DES decryption block is going to decrypt 64-bit blocks (“chunks”) of data using a 64-bit key
  - Larger block/key sizes exist, but it’s all the same idea
- Your design will allow you to change both the input data sets and the key

DES

- What does this look like?

DES

- What does this look like?

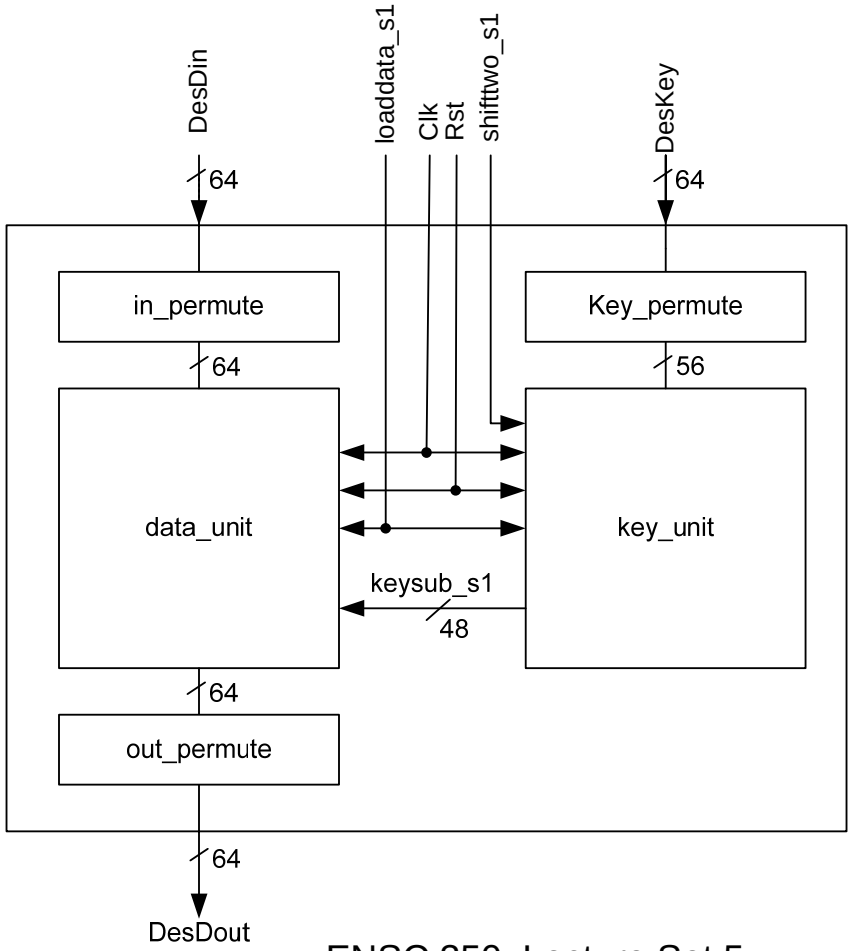


DES

- What does the datapath look like?

DES

- What does the datapath look like?



DES

- The datapath in_permute block swaps bits according to the table:

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

DES

- The datapath out_permute block swaps bits according to the table:

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

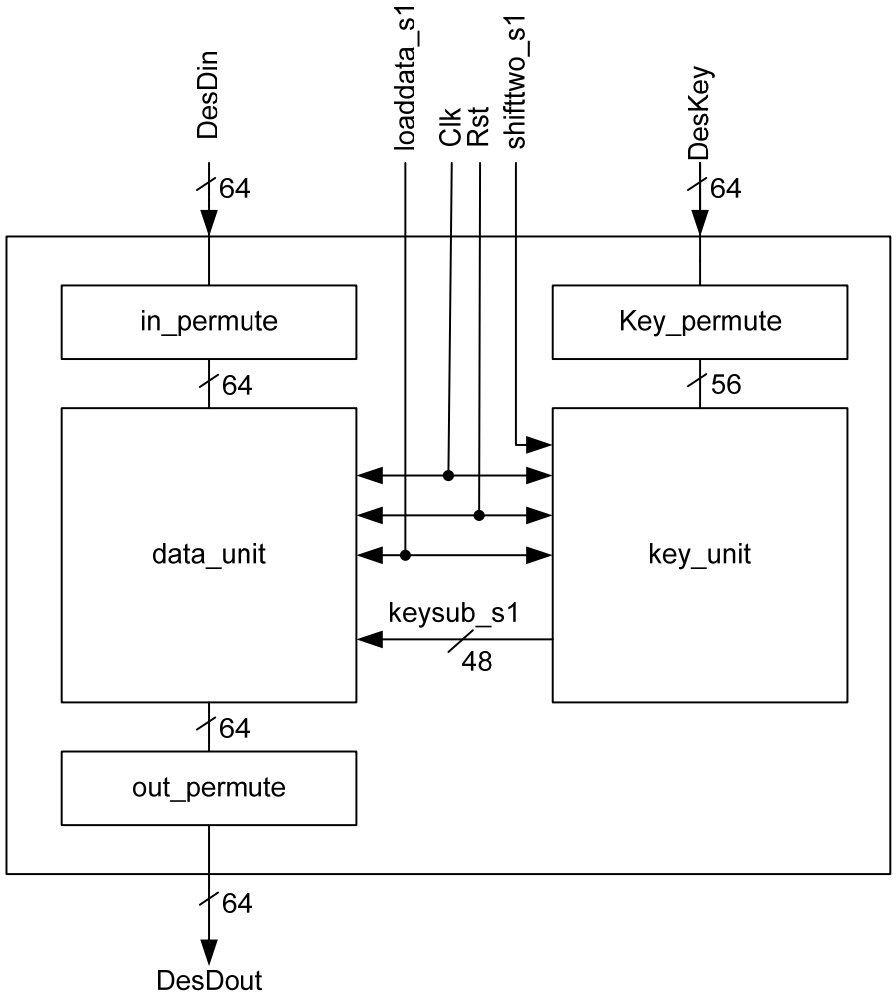
DES

- The datapath key_permute block swaps bits according to the table (note only 56 bits are used):

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

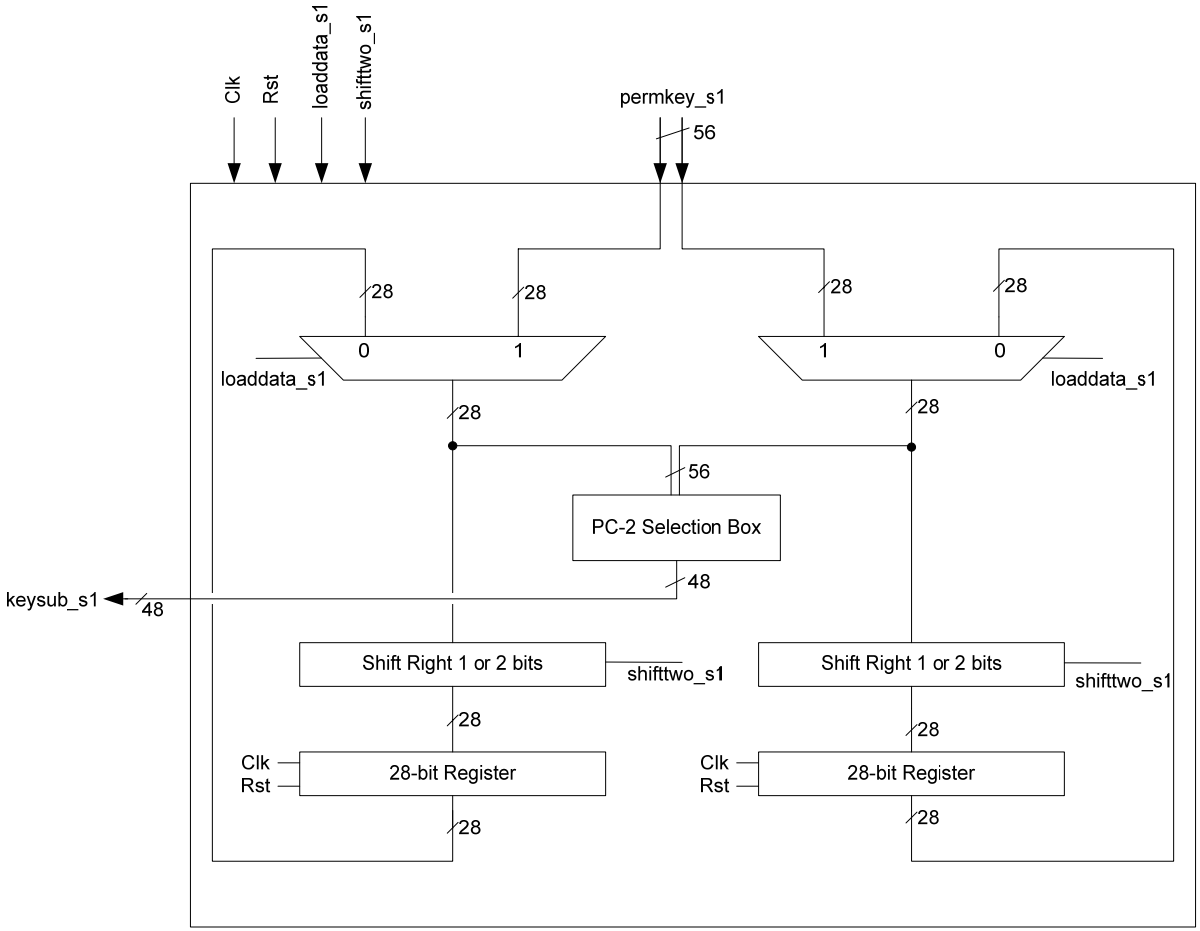
DES

- Recall the datapath:



DES

- The key_unit schematic is:



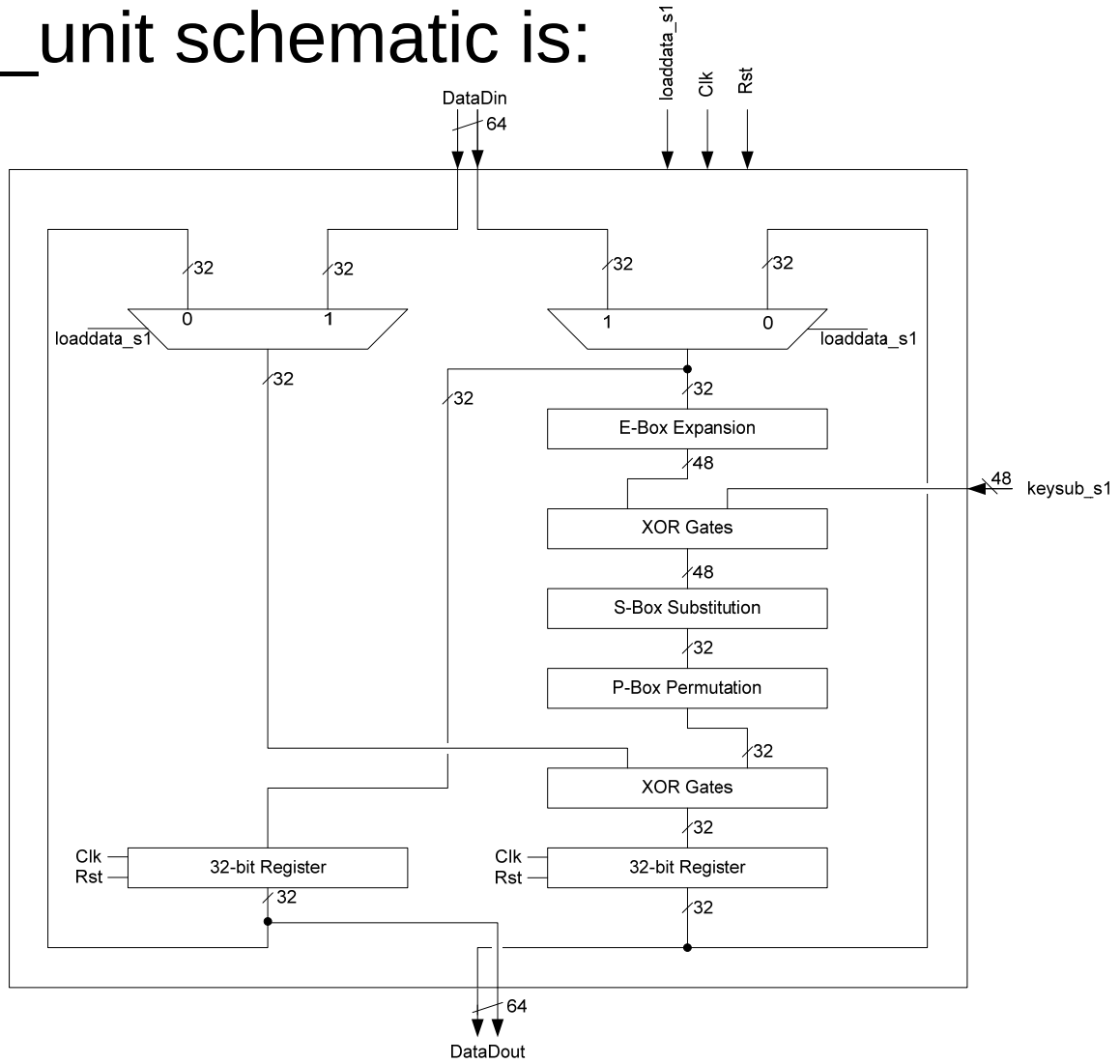
DES

- The datapath PC-2 Selection Box block swaps bits according to the table (note 56 bits are the combined datapath):

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

DES

- The data_unit schematic is:

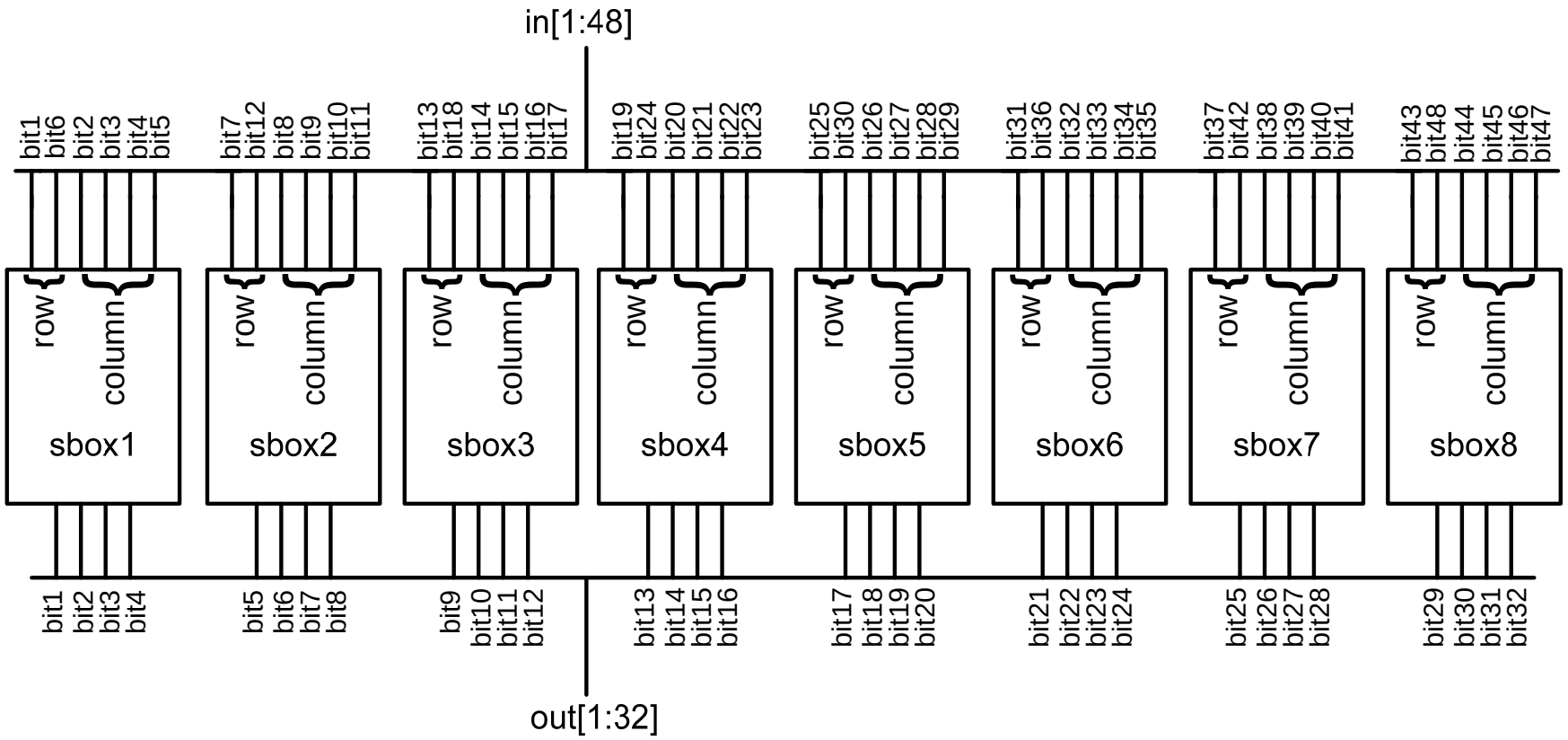


DES

- The E-box and P-box behave the same as the other tables permuting bits.
- The S-box is different:
  - It uses a 48-bit address to generate a 32-bit data word
    - A 2^{48} memory is too large to implement though
    - 32 logic equations with 48 inputs aren't feasible either
    - Instead, eight 2^6 4-bit ROMs are used to generate the 32-bit word

DES

- The substitution box schematic is:



DES

- The pattern for S-box 1 is shown here (the rest are in the lab document), with the addressing scheme Row: Bits {1,6} Column: Bits {2,3,4,5}:

Column Number

Row	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

DES

- The pattern for S-box 1 is shown here (the rest are in the lab document), with the addressing scheme Row: Bits {1,6} Column: Bits {2,3,4,5}:

Column Number

Row	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

DES

- The output values for the control path state machine for loaddata_s1 and shifttwo_s1 are:

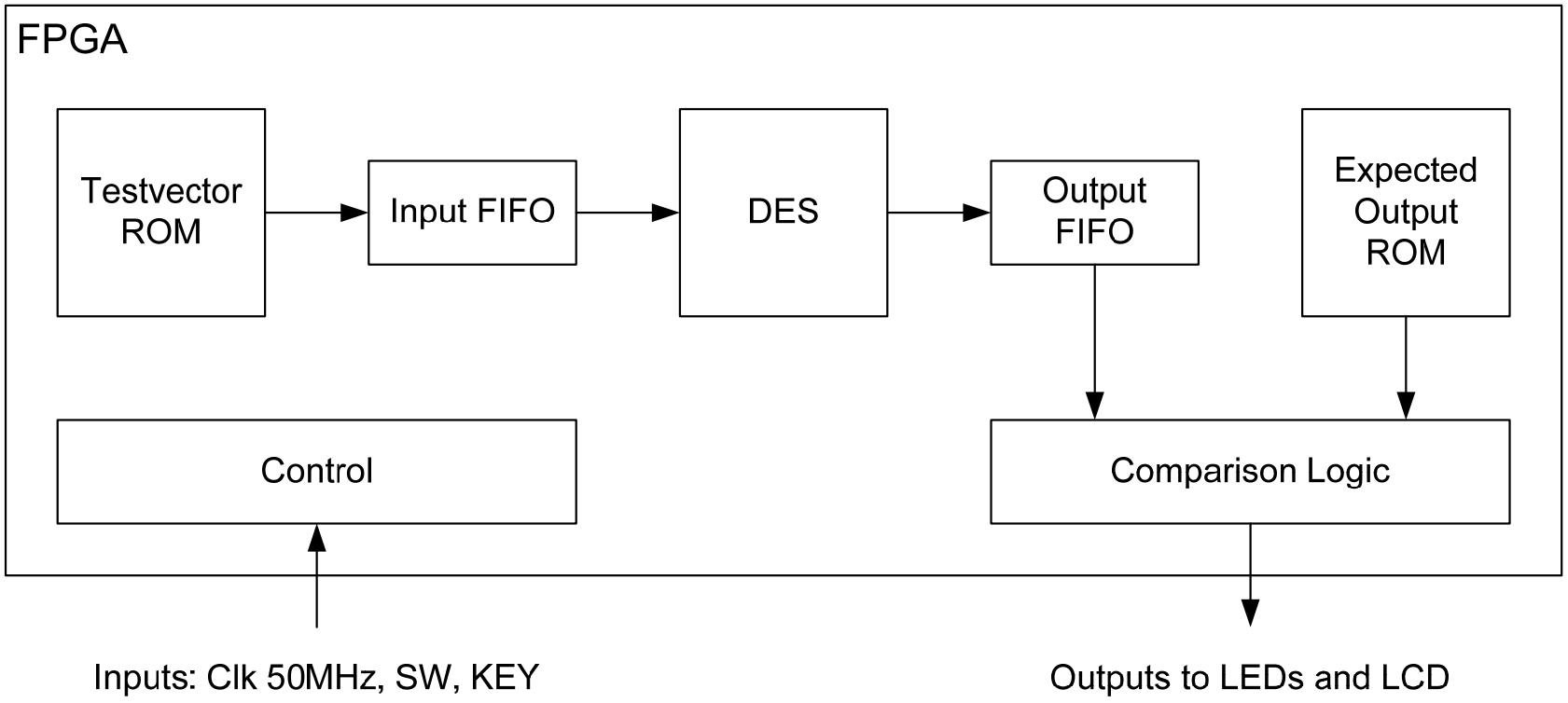
Cycle	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
loaddata_s1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
shifttwo_s1	0	1	1	1	1	1	1	0	1	1	1	1	1	1	0	0

DES

- Note that there are lots of details in the schematic and tables
- I’ve checked these slides, but their might be typos
  - ***The lab2 document is correct and should be followed if there are any differences.***
- And now onto your testbed…

DES Testbed

- The schematic for your testbed is as follows:



DES Testbed

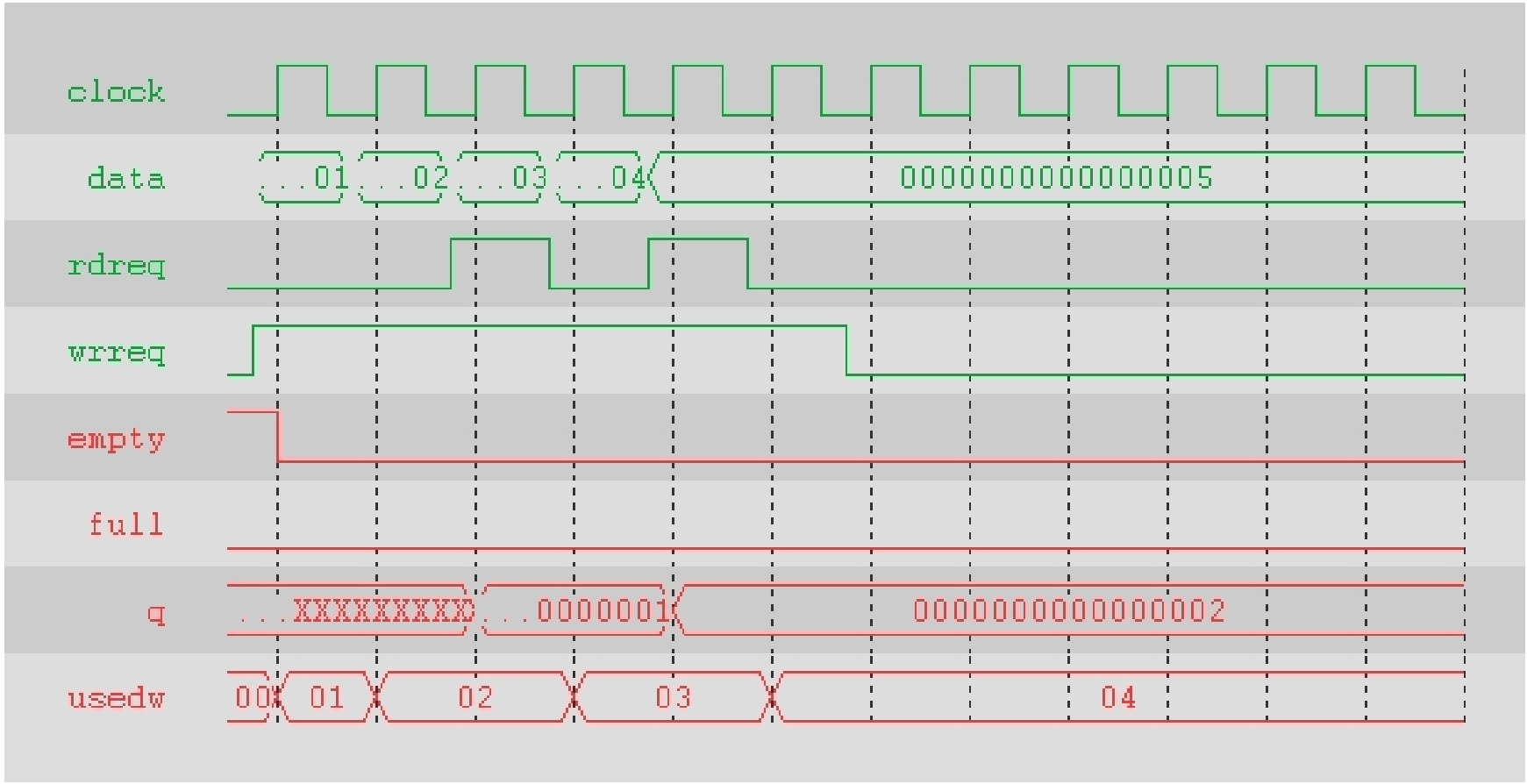
- You'll be given some of these components and you'll have to design the rest
- Note we will be evaluating your design with different keys and input vectors/expected output vectors than you are given
  - So you really need to hammer your design to make sure it is correct
  - Although it is not explicity part of the lab, you might want to create more test bench vectors using some HLL implementation (e.g. using Matlab)

DES Testbed

- The FIFOs and ROMs are created using Intellectual Property (IP) cores provided with the Quartus tools
- The ROMs are used to store test vectors. They get initialized using .mif files
- The FIFOs are used to buffer the data to be decrypted and the final results.
  - The following slides illustrate the necessary handshaking for the FIFOs

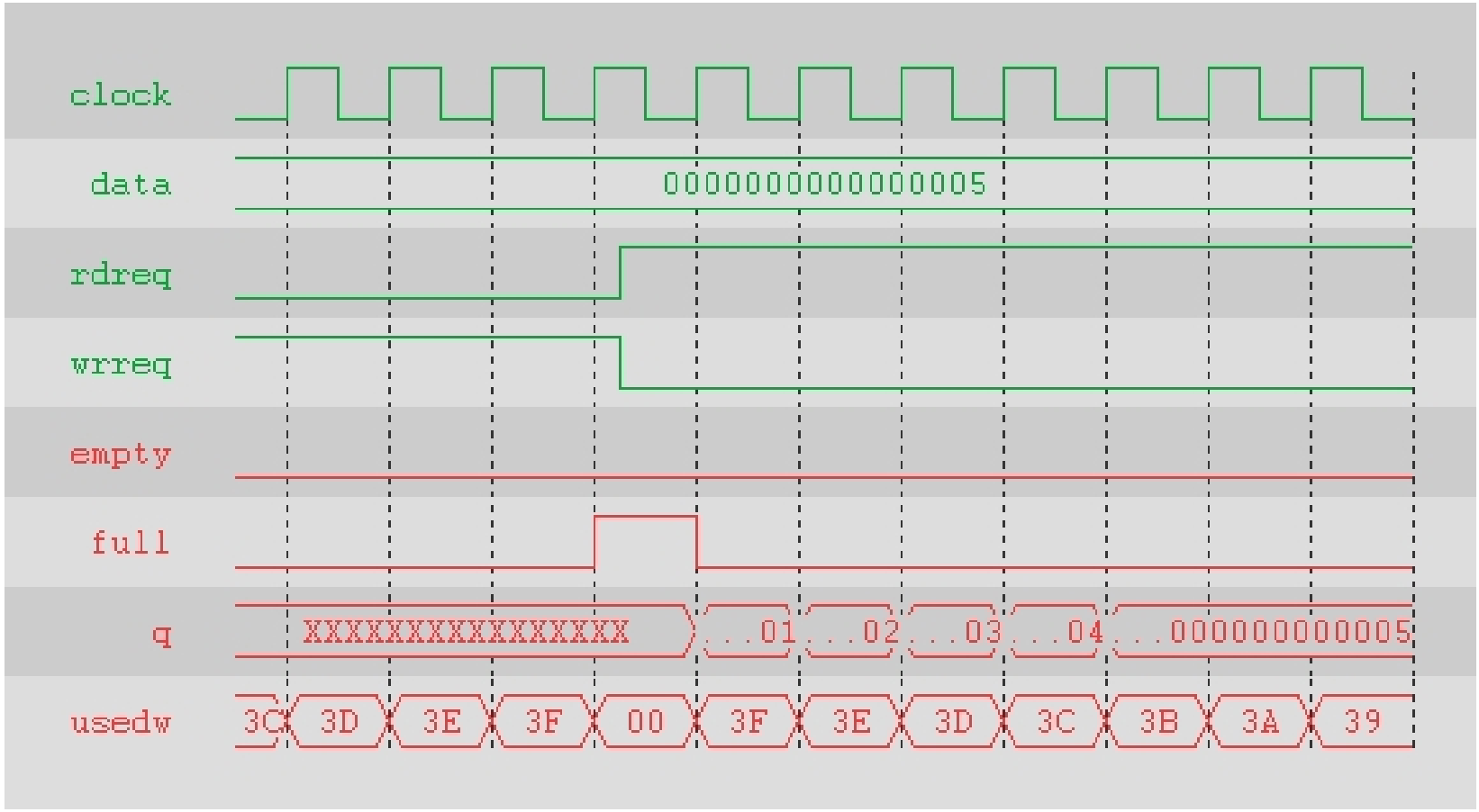
DES Testbed

- FIFO Read and Write operations:



DES Testbed

- FIFO full behaviour:



DES Lab Task 1

- Task 1: (Complete by February 8th)
  - Write the VHDL that descibes each of the subcomponents.
  - The top-level code will be structural
  - For each of the components, make the HDL behavioural or structural as is appropriate
    - Remember: All behavioural HDL ***must*** be synthesizable
  - Follow the templates for each component and test them ***well*** before proceeding with integration
    - Otherwise, you'll never be able to debug your final design

DES Lab Task 2

- Task 2: (Complete by February 11th)
  - Integrate all subcomponents
  - Maximize the (1/area * Maximum Throughput) product
  - Perform preliminary testing using simulation

DES Lab Task 3

- Task 3: (Complete by February 15th)
  - Create your Testbed for the DE2 Board
  - Although this may sound ironic, the Testbed will need to be tested independent of your DES circuit
    - Otherwise, how will you know where your buts come from (your DES circuit or the Testbed?

DES Lab Task 4

- Task 4: (Complete by February 22nd)
  - Do a thorough testing of your DES design using the DE2 board testbed
    - Test for different input vectors ***and*** different keys
  - Obtain resource usage and Performance statistics
    - Use this information to calculate your (1/area*Maximum Throughput) product

DES Lab Task 5

- Task 5: (Complete by February 26th by 3:30pm)
  - Demonstrate your design
    - Be prepared to answer questions on your design
    - Remember: We'll be using different test vectors and keys to verify your design than those with which we provide you
  - Write up and submit a report on your design to include:
    - A system block diagram and state diagram
    - Resource and performance statistics
    - HDL hierarchy and anything special about your design
    - Submit a soft copy of your source files as a zip file to the TA via WebCT
  - For more details, check out the lab document

DES Lab Marking Scheme

- Demo Marking Scheme (Total 4 marks):
  - 2 marks for working Simulations and reasonable responses to questions
  - 2 marks for working on the DE2-70 and reasonable responses to questions
- Report Marking Scheme (Total 6 marks):
  - 1 mark for accurate State Diagram (must match VHDL)
  - 1 mark for accurate System Block Diagram (must match VHDL)
  - 1 mark for resource usage report
  - 1 mark for min Clk period report and max throughput calculation
  - 1 mark for hierarchy tree and VHDL coding practices
  - 1 mark for readability of report

DES Lab Marking Scheme- Bonus & Reminders

- Bonus (not recommended for anyone without ***extensive*** VHDL skills):
  - The design we've outlined here assumes 16 clock cycles are required to decrypt each data word. For those of you that are already comfortable with synthesizable VHDL, you can try to design a ***pipelined*** version of this circuit to increase throughput. This will be ***incredibly*** challenging and should not be considered by the inexperienced or faint of heart. If successful, you will be awarded a maximum of 2 bonus marks on your final mark.
- Reminders:
  - Submit your report to the ensc350 drop box by 3:30pm on February 26th
  - Submit your VHDL to the TA via WebCT by 3:30pm on February 26th as well
  - No soft copies of the report will be accepted
  - The submission deadline is fixed (even being late by 1 hour you will be severely penalized by at least 50%)
  - Even if you miss the demo for medical reasons, you are expected to help with the report

NOTE: I’m not promising there are no typos in the document (it ***is*** 17 pages, so there are likely typos), but the datapath tables are correct. If you have any other questions, post to WebCT

Now… An Overview Testbeds

Slide Set Overview

- How we test circuits:
  - Terminology
  - Designing testbeds
    - What we are doing
    - What the HDL looks like

Terminology

- Error/Fault:
  - Incorrect operation of part of the circuit
- Failure:
  - Incorrect behaviour exhibited by the season
- Note: multiple errors may prevent system failure

Simulation/Emulation/Verification

- Simulation:
  - Typically a software program used to simulate how hardware would be have (eg a circuit, a processor)
- Emulation:
  - One hardware platform is used to emulate (mimic) another hardware platform

Simulation/Emulation/Verification

- Verification:
  - Used to verify (demonstrate) that the circuit you actually implemented behaves as originally specified

Simulation/Emulation/Verification

- Order of operation:
  - Simulation (think waveform generator)
  - Emulation
  - Verification

Testing vs Verification

- **Testing** = running the design with a set of inputs to gain confidence that the design has few errors (design is "tested")
  - *Goal*: reduce the frequency of *failures*
  - *When done*: after the design of a component/system is complete
  - *Methodology*: develop test cases, normally including boundary conditions; run the design with each test case
- **Verification** = formally proving that the design has no errors/faults (design is "correct")
  - *Goal*: detect *errors* and eliminate *failures*
  - *When done*: before, during and after the design is complete
  - *Methodology*: write separate specifications for the design; prove that the design and the specifications are ***mathematically*** equivalent

Testbed

- What does a testbed look like from the architectural perspective?

- SUT = System Under Test

Testbed – “in the HDL”

```
LIBRARY ieee;
LIBRARY std;
USE std.textio.all;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_textio.all;
USE ieee.std_logic_signed.all;

ENTITY testbench IS
END testbench;
```

Testbed – “in the HDL”

```
ARCHITECTURE testbench_arch OF testbench IS
    CONSTANT set_width      :       NATURAL := 24;
    SIGNAL R, D, Q, Qc      :       STD_LOGIC_VECTOR(set_width-
                                        1 DOWNTO 0);
    SIGNAL clk, Done        :       STD_LOGIC;
    SIGNAL Load, Resetn     :       STD_LOGIC;
    FILE vectors            :       TEXT IS IN "in_vector.vec";
    FILE dump               :       TEXT IS OUT "vhdl_output.vec";
```

Testbed – “in the HDL”

```
COMPONENT Divider
GENERIC( width: NATURAL := set_width );
PORT(       R, D            :  IN       STD_LOGIC_VECTOR
                                        (width-1 DOWNTO 0);
            Load, Resetn    :  IN       STD_LOGIC;
            clk             :  IN       STD_LOGIC;
            Done            :  BUFFER   STD_LOGIC;
            Q               :  BUFFER   STD_LOGIC_VECTOR
                                        (width-1 DOWNTO 0));
END COMPONENT;

BEGIN
```

Testbed – “in the HDL”

```
CLKP:   PROCESS
    CONSTANT clk_off_period: TIME := 10 ns;
    CONSTANT clk_on_period: TIME := 10 ns;
    BEGIN

        clk <= '0';
        WAIT FOR clk_off_period;
        clk <= '1';
        WAIT FOR clk_on_period;

    END PROCESS;
```

Testbed – “in the HDL”

```
RSTP:  PROCESS
    CONSTANT reset_period: TIME := 5 ns;
    BEGIN

        Resetn <= '0';
        WAIT FOR  reset_period;
        Resetn <= '1';
        WAIT;

    END PROCESS;
```

Testbed – “in the HDL”

```
LDP:    PROCESS
    CONSTANT load_period: TIME := 10 ns;
    BEGIN

        Load <= '1';
        WAIT FOR  load_period;
        Load <= '0';
        WAIT UNTIL Done = '1';

    END PROCESS;
```

Testbed – “in the HDL”

```
P0:    PROCESS
       VARIABLE buf_in     :     LINE;
       VARIABLE buf_out    :     LINE;
       VARIABLE x          :     BIT_VECTOR(set_width-1 DOWNTO 0);
       VARIABLE y          :     BIT_VECTOR(set_width-1 DOWNTO 0);
       VARIABLE z          :     BIT_VECTOR(set_width-1 DOWNTO 0);
       VARIABLE Diff       :     STD_LOGIC_VECTOR(set_width-1
                                                  DOWNTO 0);

        BEGIN
```

Testbed – “in the HDL”

```
WHILE NOT ENDFILE(vectors) LOOP
        READLINE(vectors, buf_in); --Read line from file.
        READ(buf_in, x);
        READ(buf_in, y);
        READ(buf_in, z);

        R <= TO_STDLOGICVECTOR(x);
        D <= TO_STDLOGICVECTOR(y);
        Qc <= TO_STDLOGICVECTOR(z);

          WRITE(buf_out, string'("R = "));
          WRITE(buf_out, x);
          WRITE(buf_out, string'("  D = "));
          WRITE(buf_out, y);
          WRITELINE(dump, buf_out);
```

Testbed – “in the HDL”

```
            WAIT UNTIL (Done = '1'); --Need to generate results
            IF( Q >= Qc) THEN           --Compare HDL and C quotients
                  Diff := Q - Qc;
            ELSE
                  Diff := Qc - Q;
            END IF;
            WRITE(buf_out, string'("   VHDL output = "));
            WRITE(buf_out, TO_BITVECTOR(Q));
            WRITE(buf_out, string'("    C output = "));
            WRITE(buf_out, TO_BITVECTOR(Qc));
            WRITELINE(dump, buf_out);                    --Write out line
            WRITE(buf_out, string'("          Difference = "));
            WRITE(buf_out, TO_BITVECTOR(Diff));
            WRITELINE(dump, buf_out);
            WAIT UNTIL (Done = '0'); --Wait before starting next divide
      END LOOP;
      WAIT;                              --Done simulation of test vectors
END PROCESS;
```

Testbed – “in the HDL”

```
    DIV0: Divider PORT MAP(R, D, Load, Resetn, clk, Done, Q);

END testbench_arch;

CONFIGURATION tb_Divider OF testbench IS
  FOR testbench_arch
  END FOR;
END tb_Divider;
```

Summary slide

Terminology matters!! Details matter!!

Pay attention to the details and be methodical in your design

This will matter for all your written tests:

- “Correct” and “almost correct” are not the same

Questions

- Does the FIFO in your testbed use first word fall through? If not, what would be the difference?

- How do the permute blocks in the DES algorithm work (e.g. P-box, in_permute, etc.)?

Questions

- Since key-permute uses only 56-bits of the 64-bit key, what does that imply about the remaining 8 bits?

- Why do we use an area-performance product to quantify the quality of a design?

Questions

- Why not simply use a single 2^{48} address ROM or 32 logic equations with 48 inputs to implement the Substitution Block?

- How do we implement the Substitution Block to determine what bits map where?

Questions

- What is the shifttwo_s1 signal used for?

- Why is the loaddata_s1 only high during the first cycle of the state machine?

Summary Questions

- Be able to define:
  - Error/Fault
  - Failure
  - Simulation
  - Emulation
  - Verification
  - Testing
  - SUT