CMR
INSTITUTE OF
TECHNOLOGY

USN | | | | | | | | | | |

Internal Assesment Test - II

Sub:	DSDV							Code:	14EC666
Date:	10 / 05 / 2017	Duration:	90 mins	Max Marks:	50	Sem:	VI	Branch:	ECE/TCE

Answer Any FIVE FULL Questions

		Marks	OBE	
			CO	RBT
1	Explain synchronous and asynchronous Static RAMs. Develop a verilog code for pipelined SSRAM.	[10]	CO4	L2
2	Develop a circuit that computes the function y=ci×x^2, where x is a binary coded input value and ci is a coefficient stored in a flow through SSRAM. X, ci and y are all signed fixed point values with 8 pre-binary points and 12 post-binary bit. The index i is also an input to the circuit, encoded as a 12 bit unsigned integer. Values for x and i arrive at the input during the cycle when a control input, start, is 1. The circuit should minimize area by using a single a single multiplier to multiply ci by x and then by x again.	[10]	CO4	L2
3	What is Error correction? Obtain ECC for data byte: - 01100001. Find, if received ECC 000111000100 is having error, if so correct it.	[10]	CO4	L2
4	Write a short note on : A) CPLDs. B) FPGA.	[10]	CO4	L1
5	For GUMNUT soft core, explain all instruction sets available with example.	[10]	CO5	L1
6	For given Opcode Pneumonic or code words (for GUMNUT CORE), obtain the Instruction encoded code word or Opcode Pneumonic respectively. (1) ADDC R3, R5, 24 (2) MASK R1, R4, R1 (3) 0C9C0 (4) 2ECFC	[10]	CO5	L2
7	Develop a Digital design model for interfacing microprocessor (GUMNUT core) or Microcontroller (8051) with External Memories.	[10]	CO5	L3

CMR
INSTITUTE OF
TECHNOLOGY

USN | | | | | | | | | | |

Internal Assesment Test - II

Sub:	DSDV							Code:	14EC666
Date:	10 / 05 / 2017	Duration:	90 mins	Max Marks:	50	Sem:	VI	Branch:	ECE/TCE

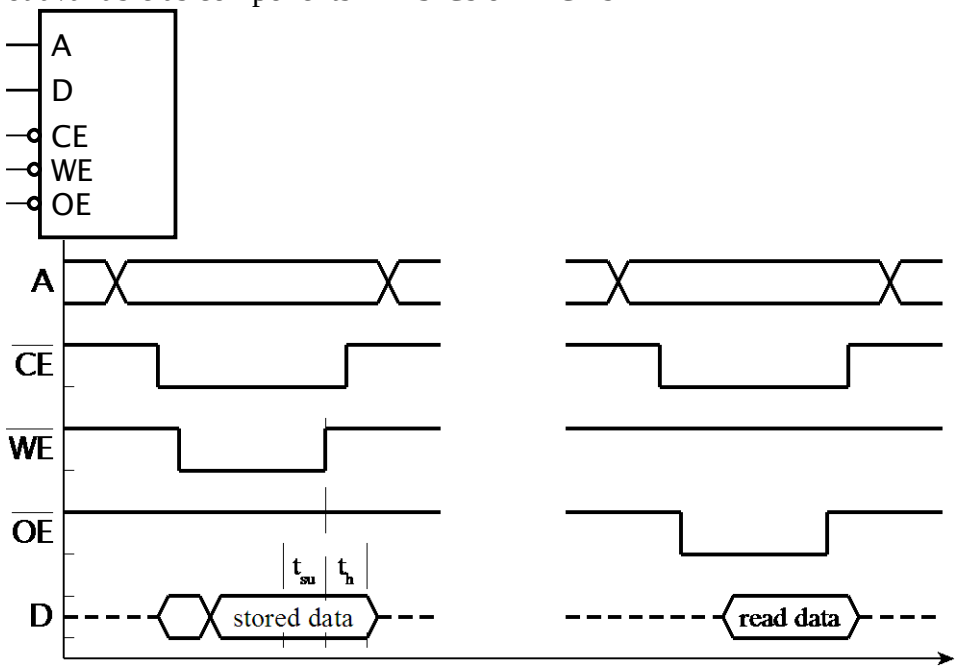
Answer Any FIVE FULL Questions

		Marks	OBE	
			CO	RBT
1	Explain synchronous and asynchronous Static RAMs. Develop a verilog code for pipelined SSRAM.	[10]	CO4	L2
2	Develop a circuit that computes the function y=ci×x^2, where x is a binary coded input value and ci is a coefficient stored in a flow through SSRAM. X, ci and y are all signed fixed point values with 8 pre-binary points and 12 post-binary bit. The index i is also an input to the circuit, encoded as a 12 bit unsigned integer. Values for x and i arrive at the input during the cycle when a control input, start, is 1. The circuit should minimize area by using a single a single multiplier to multiply ci by x and then by x again.	[10]	CO4	L2
3	What is Error correction? Obtain ECC for data byte: - 01100001. Find, if received ECC 000111000100 is having error, if so correct it.	[10]	CO4	L2
4	Write a short note on : A) CPLDs. B) FPGA.	[10]	CO4	L1
5	For GUMNUT soft core, explain all instruction sets available with example.	[10]	CO5	L1
6	For given Opcode Pneumonic or code words (for GUMNUT CORE), obtain the Instruction encoded code word or Opcode Pneumonic respectively. (1) ADDC R3, R5, 24 (2) MASK R1, R4, R1 (3) 0C9C0 (4) 2ECFC	[10]	CO5	L2
7	Develop a Digital design model for interfacing microprocessor (GUMNUT core) or Microcontroller (8051) with External Memories.	[10]	CO5	L3

Solutions

1. Asynchronous SRAM

a. Data stored in 1-bit latch cells
b. Address decoded to enable a given cell
c. Usually use active-low control inputs
d. Not available as components in ASICs or FPGAs



3 Marks

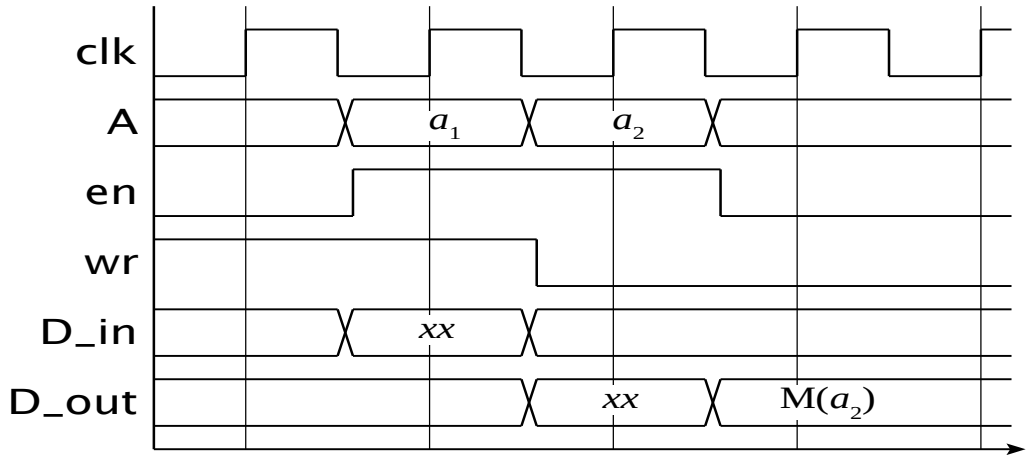
Synchronous SRAM

a. Clocked storage registers for inputs
b. address, data and control inputs
c. stored on a clock edge
d. held for read/write cycle

4 Marks

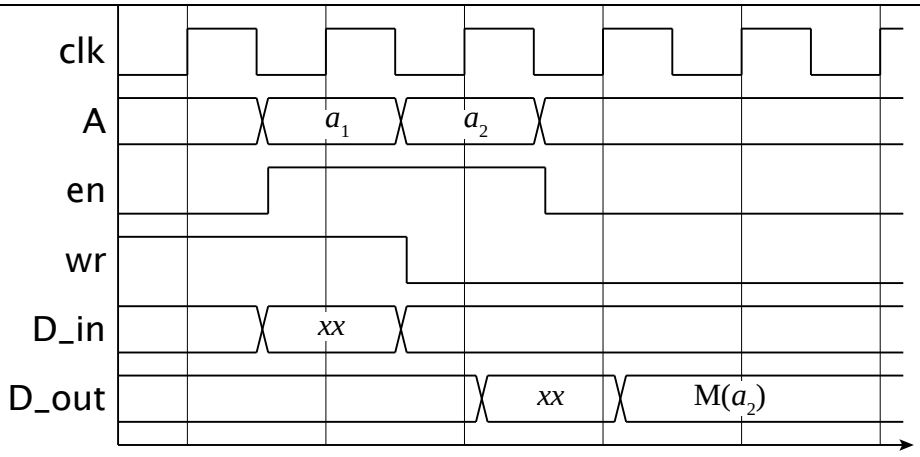
Flow-through SSRAM

- no register on data output



Pipelined SSRAM

- Data output also has a register
  - More suitable for high-speed systems
  - Access RAM in one cycle, use the data in the next cycle



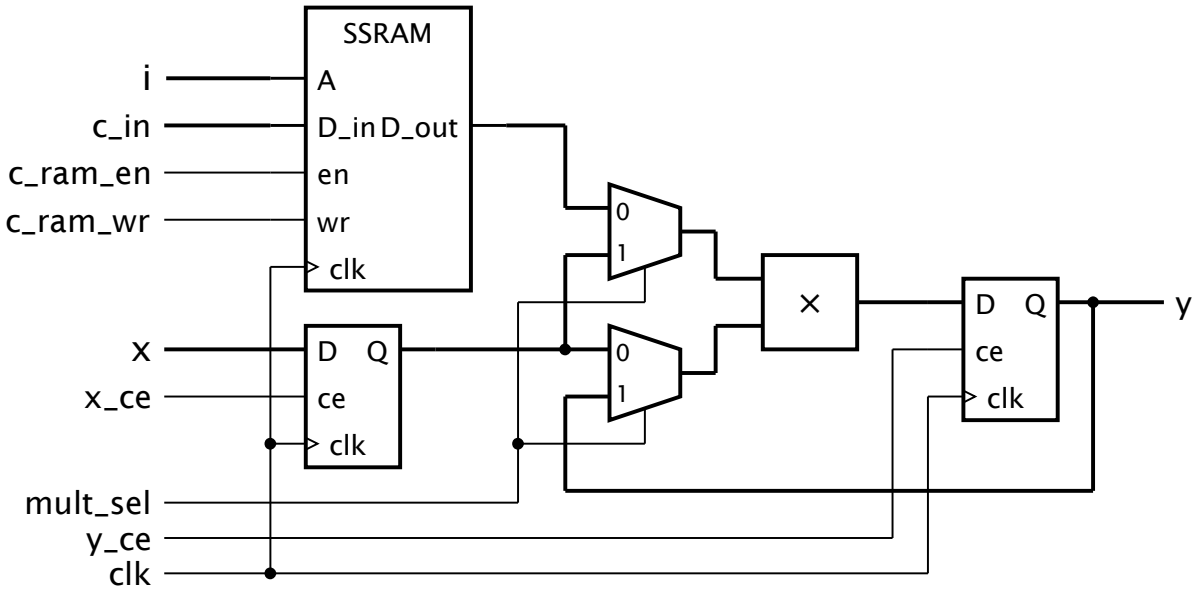
Verilog code for pipelined SSRAM

```
reg       pipelined_en;
reg [15:0] pipelined_d_out;
...
always @(posedge clk) begin
  if (pipelined_en) d_out <= pipelined_d_out;
  pipelined_en <= en;
  if (en)
    if (wr) begin
      data_RAM([a] <= d_in;  pipelined_d_out <= d_in;
    end
    else
      pipelined_d_out <= data_RAM[a];
end
```

3 Marks

2. 5 Marks

- Compute function
  - Coefficient stored in flow-through SSRAM
    - 12-bit unsigned integer index for *i*
  - *x*, *y*, c_i 20-bit signed fixed-point
    - 8 pre- and 8 post-binary point bits
  - Use a single multiplier
    - Multiply $c_i \times x \times x$



```
module scaled_square ( output reg signed [7:-12] y,
                       input     signed [7:-12] c_in, x,
                       input            [11:0]  i,
                       input                    start,
                       input                    clk, reset );
  wire             c_ram_wr;
  reg              c_ram_en, x_ce, mult_sel, y_ce;
  reg signed [7:-12] c_out, x_out;
  reg signed [7:-12] c_RAM [0:4095];
  reg signed [7:-12] operand1, operand2;
  parameter [1:0] step1 = 2'b00, step2 = 2'b01, step3 = 2'b10;
  reg       [1:0] current_state, next_state;
  assign c_ram_wr = 1'b0;

always @(posedge clk)  // c RAM - flow through
    if (c_ram_en)
      if (c_ram_wr) begin
        c_RAM[i] <= c_in;
        c_out    <= c_in;
      end
      else
        c_out <= c_RAM[i];
  always @(posedge clk)  // y register
    if (y_ce) begin
      if (!mult_sel) begin
        operand1 = c_out;
        operand2 = x_out;
      end
      else begin
        operand1 = x_out;
        operand2 = y;
      end
      y <= operand1 * operand2;
    end
```

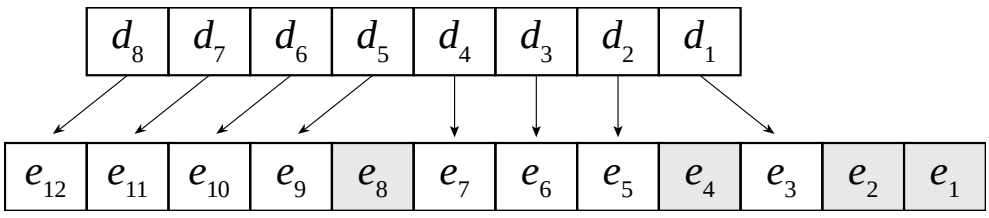
5 Marks

3

Error-Correcting Codes (ECC)

- Allow identification of the flipped bit
- Hamming Codes
  - E.g., for single-bit-error correction of N-bit word, need $\log_2 N + 1$ extra bits
- Example: 8-bit word, $d_1 \ldots d_8$
  - 12-bit ECC code, $e_1 \ldots e_{12}$

e_1, e_2, e_4, e_8 are check bits, the rest data



4Marks

$e_1 = e_3 \oplus e_5 \oplus e_7 \oplus e_9 \oplus e_{11}$

$e_2 = e_3 \oplus e_6 \oplus e_7 \oplus e_{10} \oplus e_{11}$

$e_4 = e_5 \oplus e_6 \oplus e_7 \oplus e_{12}$

$e_8 = e_9 \oplus e_{10} \oplus e_{11} \oplus e_{12}$

- Every data bit covered by two or more check bits
- On write: Compute check bits and store with data
- On read: Recompute check bits and XOR with read check bits
  - result called the *syndrome*
- 0000 => no error
- If data bit flipped
  - covering bits of syndrome are 1
  - = binary code of flipped ECC bit
- If stored check bit flipped
  - that bit of syndrome is 1
- On error, unflip bit and rewrite memory location

ii) The check bits are

$$e_1 = e_3 \oplus e_5 \oplus e_7 \oplus e_9 \oplus e_{11} = d_1 \oplus d_2 \oplus d_4 \oplus d_5 \oplus d_7 = 1 \oplus 0 \oplus 0 \oplus 0 \oplus 1 = 0$$

$$e_2 = e_3 \oplus e_6 \oplus e_7 \oplus e_{10} \oplus e_{11} = d_1 \oplus d_3 \oplus d_4 \oplus d_6 \oplus d_7 = 1 \oplus 0 \oplus 0 \oplus 1 \oplus 1 = 1$$

$$e_4 = e_5 \oplus e_6 \oplus e_7 \oplus e_{12} = d_2 \oplus d_3 \oplus d_4 \oplus d_8 = 0 \oplus 0 \oplus 0 \oplus 0 = 0$$

$$e_8 = e_9 \oplus e_{10} \oplus e_{11} \oplus e_{12} = d_5 \oplus d_6 \oplus d_7 \oplus d_8 = 0 \oplus 1 \oplus 1 \oplus 0 = 0$$

Thus the ECC word is 011000000110.

3Marks

iii) Determine whether there is an error in the ECC word 000111000100, and if so, correct it.
solution The check bits computed from the data bits of the ECC word are

$$e_1 = e_3 \oplus e_5 \oplus e_7 \oplus e_9 \oplus e_{11} = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$$

$$e_2 = e_3 \oplus e_6 \oplus e_7 \oplus e_{10} \oplus e_{11} = 1 \oplus 0 \oplus 1 \oplus 0 \oplus 0 = 0$$

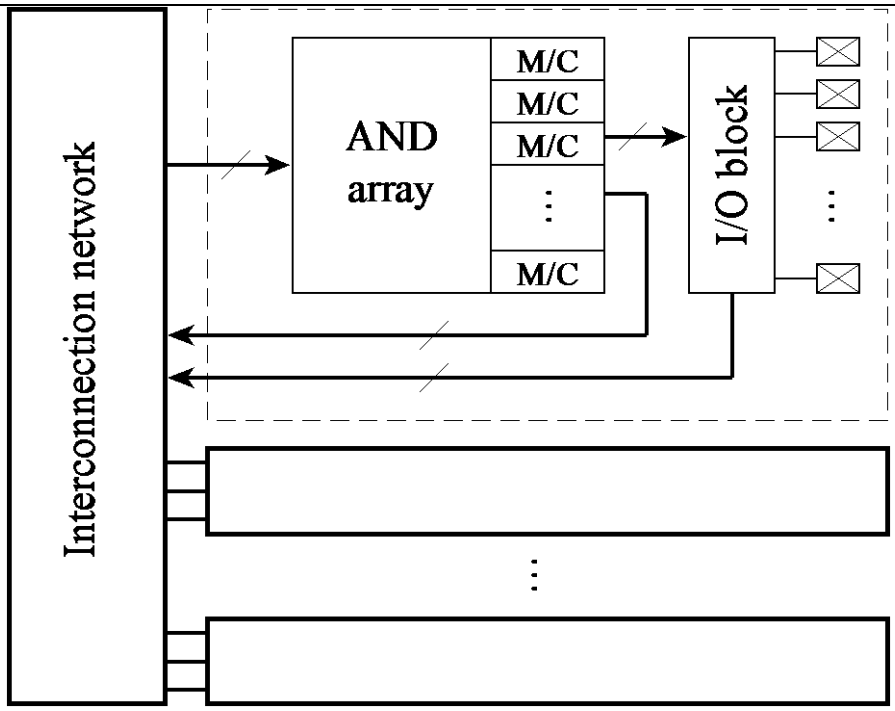
$$e_4 = e_5 \oplus e_6 \oplus e_7 \oplus e_{12} = 0 \oplus 0 \oplus 1 \oplus 0 = 1$$

$$e_8 = e_9 \oplus e_{10} \oplus e_{11} \oplus e_{12} = 0 \oplus 0 \oplus 0 \oplus 1 = 1$$

3Marks

The syndrome is 1101 $\oplus$1000 _ 0101. Thus, there is an error in bit e5 of the read ECC. That bit should be flipped back from 0 to 1, giving the corrected ECC word 000111010100.

4	Complex PLDs (CPLDs)	5Marks

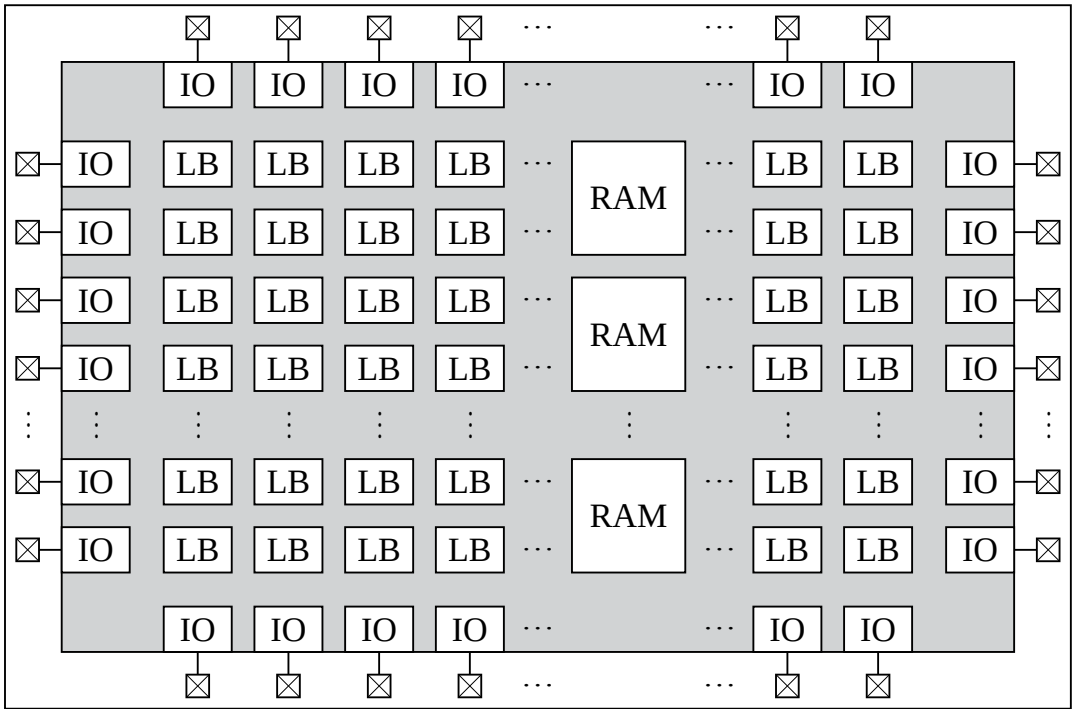


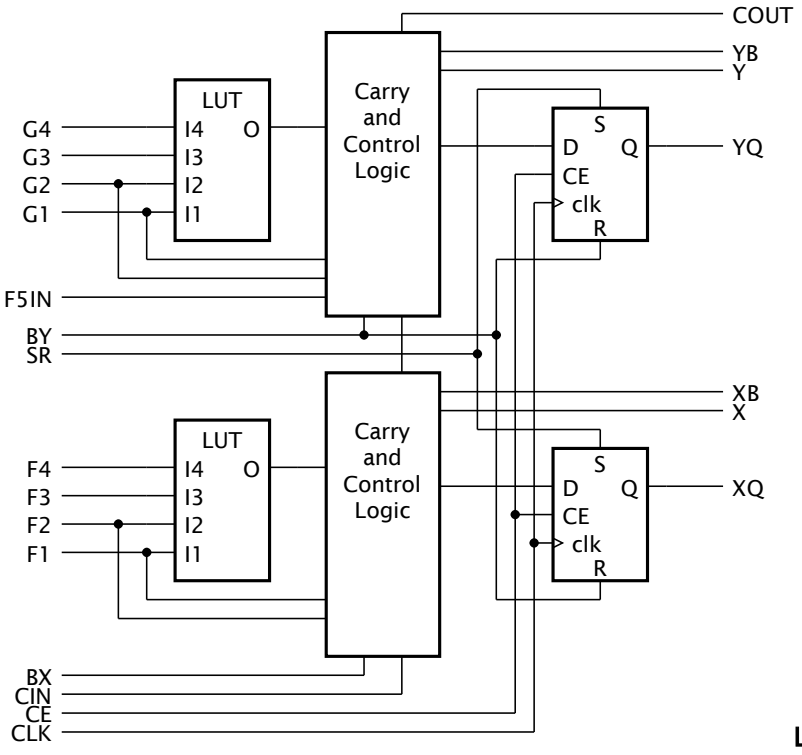
- Cramming multiple PALs into an IC
  - Programmable interconnection network
  - Use flash RAM technology to store configuration

Each of the PAL structures consists of an AND array and a number of embedded macrocells (M/Cs in the figure). The macrocells contain OR gates, mutiplexers and flip-flops, allowing choice among combinational or registered connections to other elements within the component, with or without logical negation, choice of initialization for flip-flops, and so on. They are essentially expanded forms of the simple macrocell shown in above, but without the direct connections to external pins. Instead, the external pins are connected to an I/O block, which allows selection among macrocell outputs to drive each pin. The network interconnecting the PAL structures allows each PAL to use feedback connections from other PALs as well as inputs from external pins. As well as providing more circuit resources than simple PLDs, modern CPLDs are typically programmed differently. Rather than using EPROMlike technology, they use SRAM cells to store configuration bits that control connections in the AND-OR arrays and the select inputs of multiplexers. Configuration data is stored in nonvolatile flash RAM within the CPLD chip, and is transferred into the SRAM when power is applied. Separate pins are provided on the chip for writing to the flash RAM, even while the chip is connected in the final system. Thus, designs using CPLDs can be upgraded by reprogramming the configuration information.

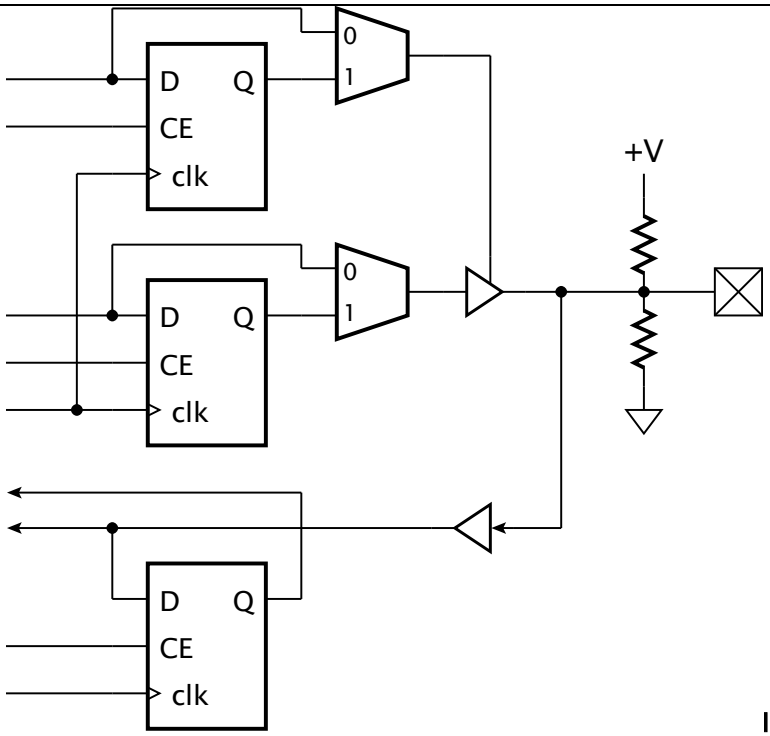
FPGA

5marks


IO
LB
RAM


COUT
YB
Y
LUT
I4
I3
I2
I1
O
G4
G3
G2
G1
Carry
and
Control
Logic
S
D
Q
CE
clk
R
YQ
F5IN
BY
SR
XB
X
F4
F3
F2
F1
XQ
BX
CIN
CE
CLK

LOGIC BLOCK



IO BLOCK

The basic elements within logic blocks are small 1-bit-wide asynchronous RAMs called lookup tables (LUTs). The LUT address inputs are connected to the inputs of the logic block. The content of an LUT determines the values of a Boolean function of the inputs. By programming the LUT content differently, we can implement any Boolean function of the inputs. The logic blocks also contain one or more flipflops and various multiplexers and other logic for selecting data sources and for connecting data to adjacent logic blocks. The logic block contains two such slices, together with a small amount of additional logic. Each slice consists of two 4-input LUTs, each of which can be programmed to implement any function of the four inputs. The carry and control logic consists of circuitry to combine the LUT outputs, an XOR gate and an AND gate for implementing adders and multipliers, as well as multiplexers that can be used to implement a fast carry chain.

The I/O block of an FPGA is typically organized as shown. The select inputs of the multiplexers are programmed to control whether the output is registered or combinational. The top flip-flop and multiplexer control the high-impedance state of the tristate driver that drives the pin as an output, and the middle flip-flop and multiplexer drive the output value. The output driver is programmable, allowing selection of logic levels (regular 5V TTL, low voltage TTL, or others) and control of the slew rate, that is, rate of voltage change at the output.

5	Gumnut Instruction sets	10Marks

INSTRUCTION	DESCRIPTION
Arithmetic and logical instructions	
add *rd*, *rs*, *op2*	Add *rs* and *op2*, result in *rd*
addc *rd*, *rs*, *op2*	Add *rs* and *op2* with carry, result in *rd*
sub *rd*, *rs*, *op2*	Subtract *op2* from *rs*, result in *rd*
subc *rd*, *rs*, *op2*	Subtract *op2* from *rs* with carry, result in *rd*
and *rd*, *rs*, *op2*	Logical AND of *rs* and *op2*, result in *rd*
or *rd*, *rs*, *op2*	Logical OR of *rs* and *op2*, result in *rd*
xor *rd*, *rs*, *op2*	Logical XOR of *rs* and *op2*, result in *rd*
mask *rd*, *rs*, *op2*	Logical AND of *rs* and NOT *op2*, result in *rd*
Shift instructions	
shl *rd*, *rs*, *count*	Shift *rs* value left *count* places, result in *rd*
shr *rd*, *rs*, *count*	Shift *rs* value right *count* places, result in *rd*
rol *rd*, *rs*, *count*	Rotate *rs* value left *count* places, result in *rd*
ror *rd*, *rs*, *count*	Rotate *rs* value right *count* places, result in *rd*
Memory and I/O instructions	
ldm *rd*, (*rs*) ± *offset*	Load to *rd* from memory
stm *rd*, (*rs*) ± *offset*	Store to memory from *rd*
inp *rd*, (*rs*) ± *offset*	Input to *rd* from input controller register
out *rd*, (*rs*) ± *offset*	Output to output controller register from *rd*
Branch instructions	
bz ± *disp*	Branch if Z is set
bnz ± *disp*	Branch is Z is not set
bc ± *disp*	Branch if C is set
bnc ± *disp*	Branch if C is not set
Jump instructions	
jmp *addr*	Jump to *addr*
jsb *addr*	Jump to subroutine at *addr*
Miscellaneous instructions	
ret	Return from subroutine
reti	Return from interrupt
enai	Enable interrupts
disi	Disable interrupts
wait	Wait for interrupts
stby	Enter low-power standby mode

6

- Encoding for `addc r3, r5, 24`
  - Arithmetic immediate, fn = 001

1	3	3	3	8
0	*fn*	*rd*	*rs*	*immed*
0	0 0 1	0 1 1	1 0 1	0 0 0 1 1 0 0 0

- 05D18

- Instruction encoded by 3ECFC

1	1	1	1	1	0	1	1	0	0	1	1	1	1	1	1	0	0

	6	2	2	8
Branch	1 1 1 1 1 0	*fn*		*disp*

- bnc -4

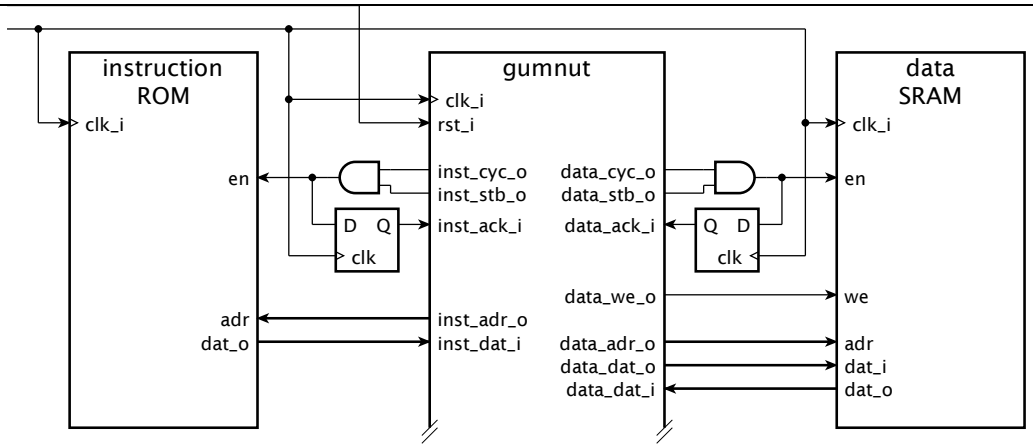
2.5Marks

2.5Marks

MASK R1,R4,R1 => 38c27 — 2.5Marks

0C9C0 => SUBB R1,R1,192 — 2.5Marks

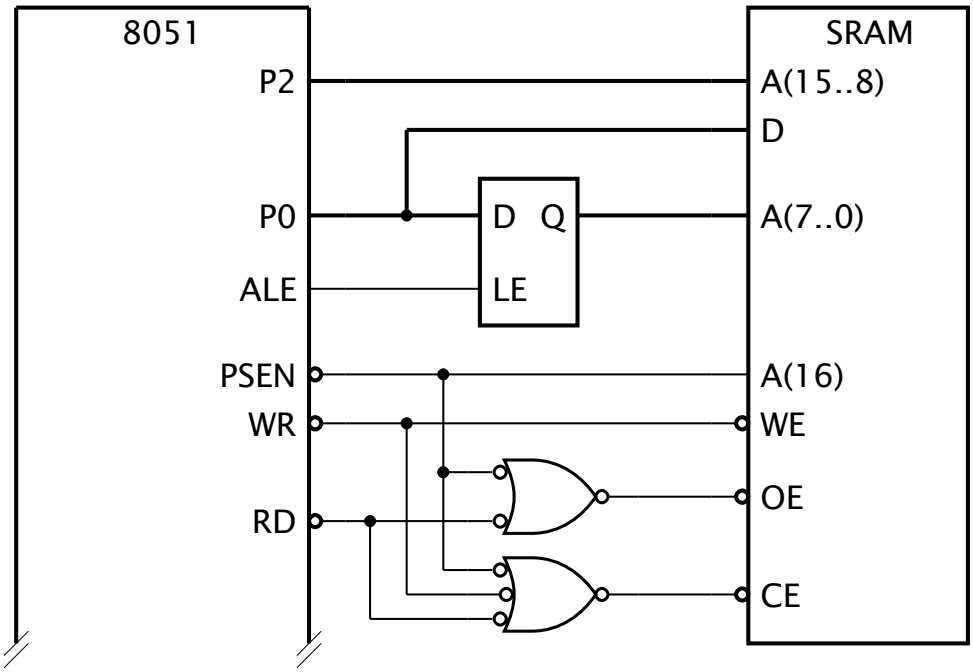
7 GUMNUT memory Interface — 10 Marks




```
always @(posedge clk)  // Instruction memory
   if (inst_cyc_o && inst_stb_o) begin
     inst_dat_i <= inst_ROM[inst_adr_o[10:0]];
     inst_ack_i <= 1'b1;
   end
   else
     inst_ack_i <= 1'b0;
always @(posedge clk)  // Data memory
   if (data_cyc_o && data_stb_o)
     if (data_we_o) begin
       data_RAM[data_adr_o] <= data_dat_o;
       data_dat_i           <= data_dat_o;
       data_ack_i           <= 1'b1;
     end
     else begin
       data_dat_i <= data_RAM[data_adr_o];
       data_ack_i <= 1'b1;
     end
   else
     data_ack_i <= 1'b0;
```

(OR)

MICROCONTROLLER 8051 memory interface



The 8051 can access up to 64K bytes of instruction memory and 64K bytes of data memory, however, there are

	only 256 bytes of data memory and 4K to 16K bytes of instruction memory on the chip. The chip has two 8-bit input/output ports, P0 and P2, as well as a number of control signals that can be used to connect to external memory. Figure shows how they would be used to connect to an external 128K x 8-bit asynchronous SRAM, in which the lower 64K locations are used for instructions and the upper 64K locations for data. P2 provides the most significant address byte, and P0 is multiplexed with the least significant address byte and instruction and data bytes. Since information transfer on P0 is bidirectional, tristate drivers are used internally in the microcontroller and in the memory data pins. The 8051 activates the address-latch enable (ALE) signal when it drives the least significant address bits on P0. We provide an 8-bit latch to hold these bits for the remainder of the memory access cycle. During an instruction read access, the 8051 activates the program-store enable (!PSEN) signal, driving it to a low logic level. At other times, including data accesses, the signal is at a high logic level. Hence, we can use this signal directly as the most significant address bit to distinguish between instruction and data accesses to the external memory. The 8051 activates the RD signal during data read accesses and the (!WR) signal during data write accesses. We use !WR directly to control the memory's write enable (!WE) signal. However, we need a small amount of glue logic to derive the chip enable (!CE) and output enable (!OE) signals.	