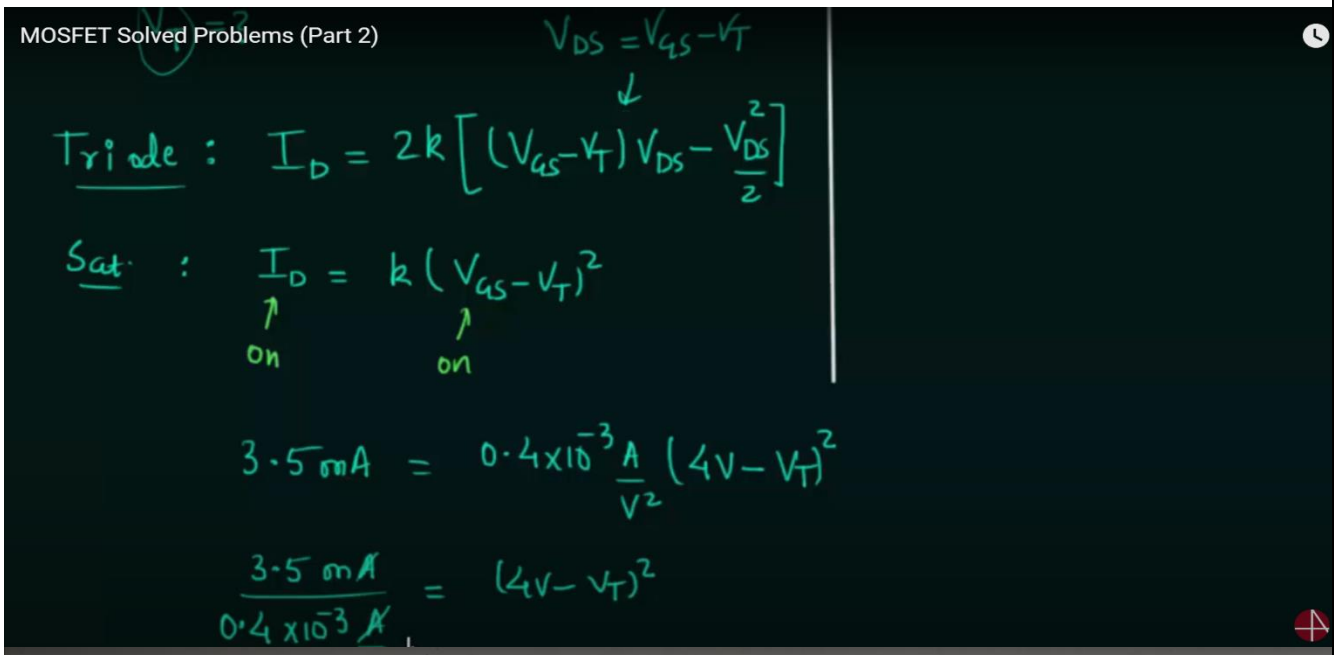
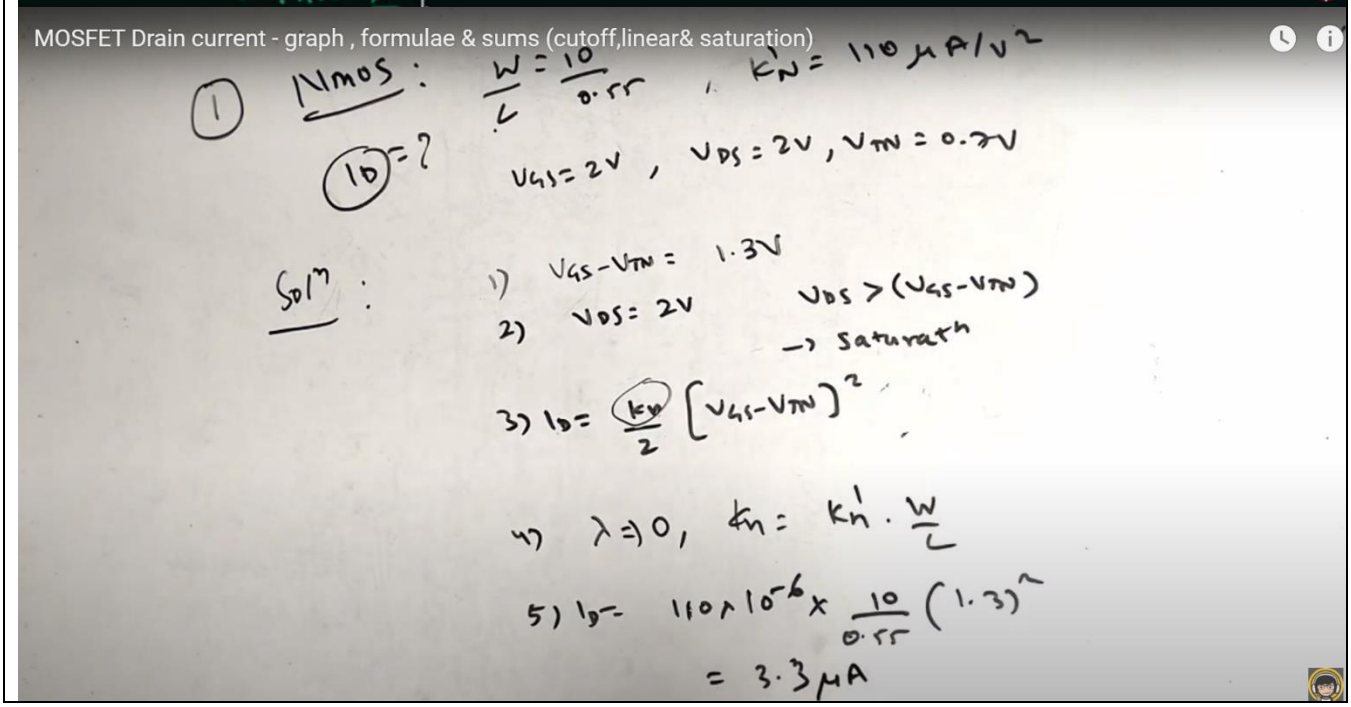
DAILY ASSESSMENT FORMAT

Date:	9th June 2020	**Name:**	Soundarya NA
Course:	VLSI	**USN:**	4AL16EC077
Topic:	VLSI	**Semester & Section:**	8th - B

FORENOON SESSION DETAILS

Image of session





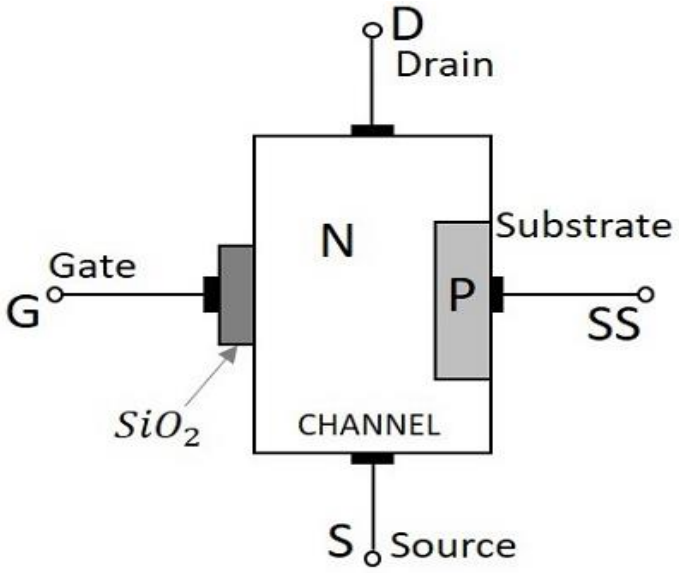
Report:

Enhancement Type MOSFET:

Construction:

The construction of a MOSFET is a bit similar to the FET. An oxide layer is deposited on the substrate to which the gate terminal is connected. This oxide layer acts as an insulator (sio2 insulates from the substrate), and hence the MOSFET has another name as IGFET. In the construction of MOSFET, a lightly doped substrate, is diffused with a heavily doped region. Depending upon the substrate used, they are called as P-type and N-type MOSFETs.

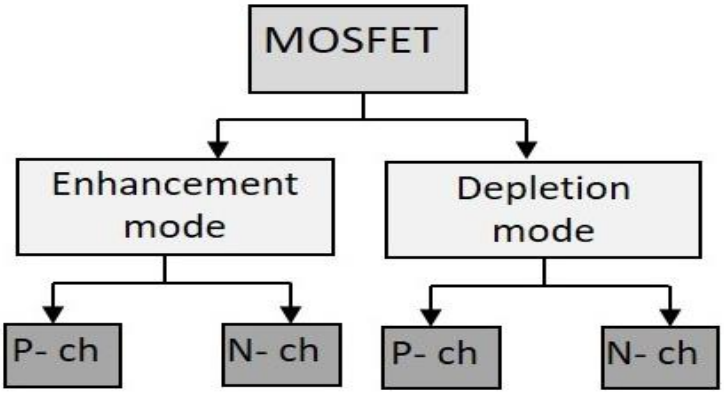
The following figure shows the construction of a MOSFET.



The voltage at gate controls the operation of the MOSFET. In this case, both positive and negative voltages can be applied on the gate as it is insulated from the channel. With negative gate bias voltage, it acts as depletion MOSFET while with positive gate bias voltage it acts as an Enhancement MOSFET.

Classification of MOSFETs:

Depending upon the type of materials used in the construction, and the type of operation, the MOSFETs are classified as in the following figure.



- P- ch = P- channel
- N- ch = N- channel

After the classification, let us go through the symbols of MOSFET.

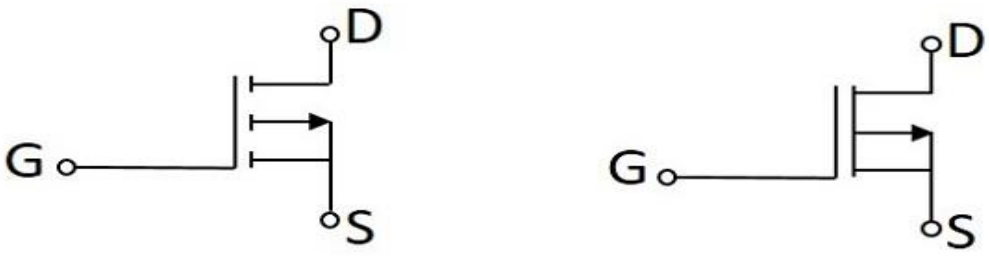
The N-channel MOSFETs are simply called as NMOS. The symbols for N-channel MOSFET are as given below.

Symbols of N-Channel MOSFET

Enhancement Mode Depletion Mode

The P-channel MOSFETs are simply called as PMOS. The symbols for P-channel MOSFET are as given below.

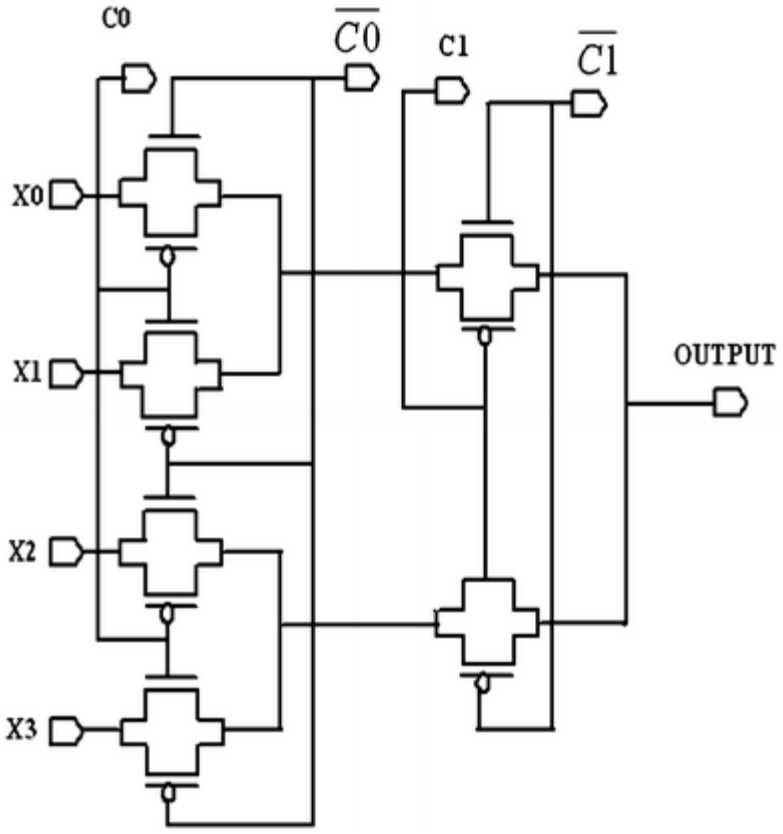
Symbols of P-Channel MOSFET



Enhancement Mode Depletion Mode

Transmission logic based 4:1 MUX:

This design is the transmission gate type of MUX structure implemented with very minimum transistors compared to the conventional CMOS based design. The back-to-back connected PMOS and NMOS arrangement acts as a switch is so called transmission gate. NMOS devices pass a strong 0 but a weak 1, while PMOS pass a strong 1 but a weak 0. The transmission gate combines the best of both the properties by placing NMOS in parallel with the PMOS device. Four transmission gates are connected as in Fig to form a MUX structure.



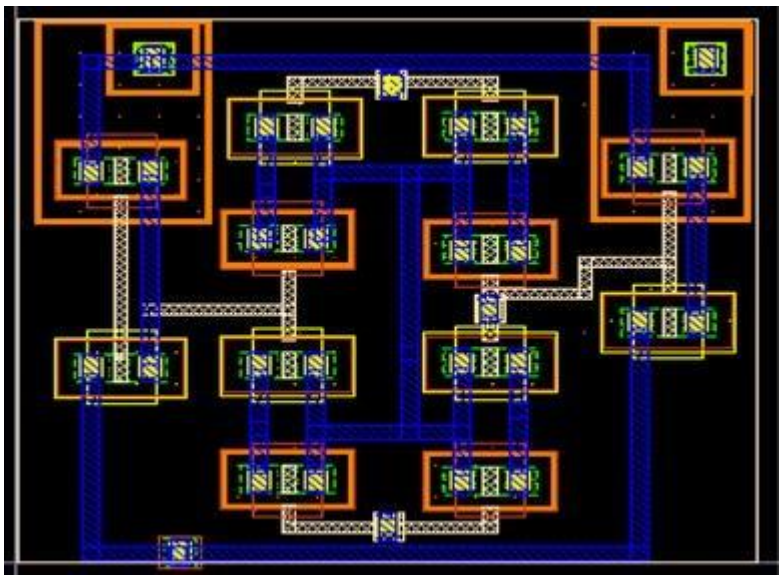
Transmission logic based 4:1 MUX

Each transmission gate acts as an AND switch to replace the AND logic gate which is used in a conventional gate design of MUX. Hence, the device count is reduced. The transmission gate based 4:1 MUX is shown in Fig.

Layout of transmission gate based 4:1 MUX:

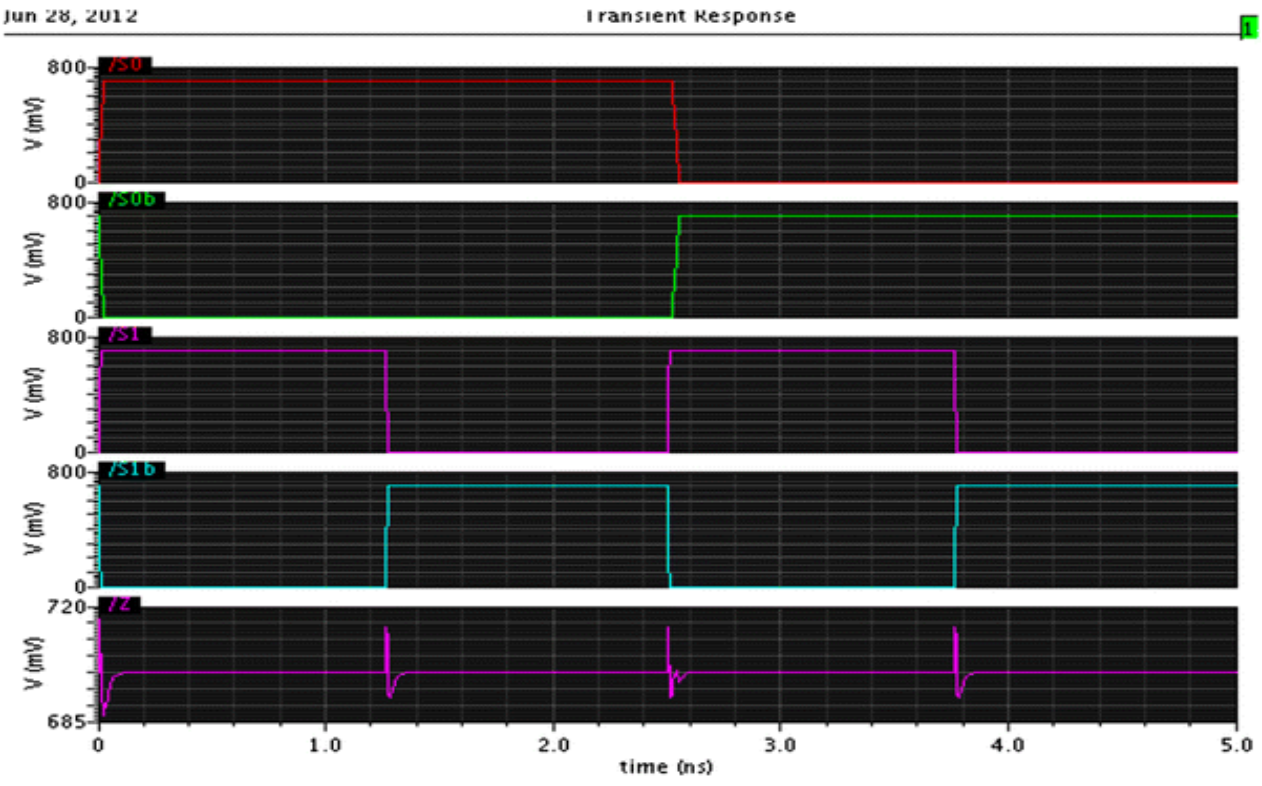
The advances in the CMOS processes are generally complex and somewhat inhibit the visualization of all the mask levels that are used in the actual fabrication process. Nevertheless, the design process can be abstracted to a manageable number of conceptual layout levels that represent the physical

features observed in the final silicon wafer. An advantage of the new MUX design is the remarkable gain in terms of transistors count. To the best of our knowledge, no 4:1 MUX has been realized with so few devices. Hence, the gain in area is a central result for the proposed MUX. The layout of transmission gate based 4:1 MUX is shown in Fig.



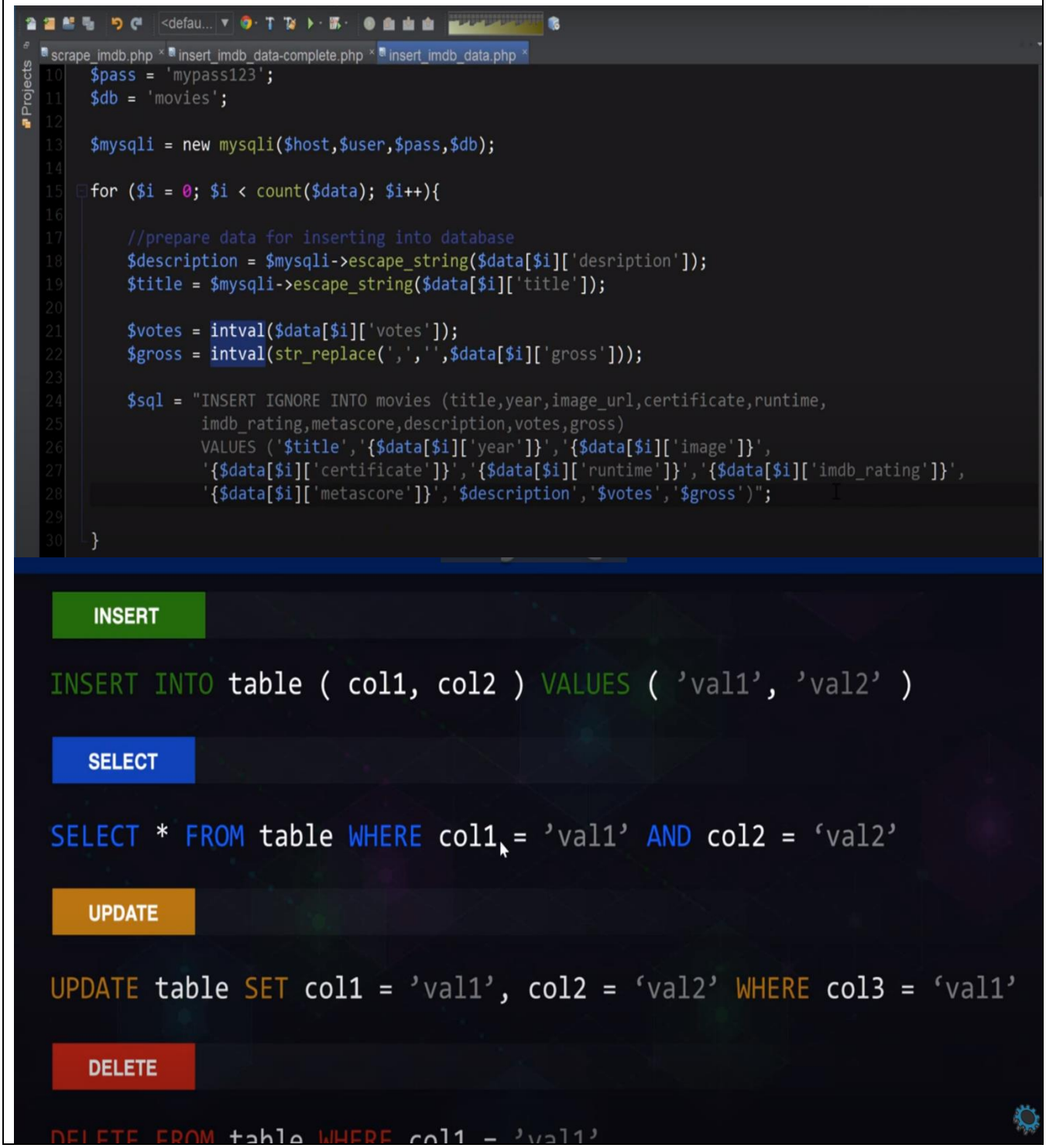
Layout of transmission gate based 4:1 MUX

Simulation results:



Date:	9th June 2020	Name:	Soundarya NA
Course:	UDEMY	USN:	4AL16EC077
Topic:	MySQL	Semester & Section:	8th - B

Image:



Report:
Outputting and processing data:

Syntax:

<OUTPUT_CLAUSE> ::=

{

[OUTPUT <dml_select_list> INTO { @table_variable | output_table } [(column_list)]]

[OUTPUT <dml_select_list>]

}

<dml_select_list> ::=

{ <column_name> | scalar_expression } [[AS] column_alias_identifier]

[,...n]

<column_name> ::=

{ DELETED | INSERTED | from_table_name } . { * | column_name }

| $action

Arguments:
Specifies a table variable that the returned rows are inserted into instead of being returned to the caller. @table_variable must be declared before the INSERT, UPDATE, DELETE, or MERGE statement.

Eg:

```
DELETE Sales.ShoppingCartItem
   OUTPUT DELETED.*;
```

Queues:

```
USE tempdb;
GO
CREATE TABLE dbo.table1
(
   id INT,
   employee VARCHAR(32)
);
GO

INSERT INTO dbo.table1 VALUES
     (1, 'Fred')
    ,(2, 'Tom')
```

```
    ,(3, 'Sally')
    ,(4, 'Alice');
GO

DECLARE @MyTableVar TABLE
(
   id INT,
   employee VARCHAR(32)
);

PRINT 'table1, before delete'
SELECT * FROM dbo.table1;

DELETE FROM dbo.table1
OUTPUT DELETED.* INTO @MyTableVar
WHERE id = 4 OR id = 2;

PRINT 'table1, after delete'
SELECT * FROM dbo.table1;

PRINT '@MyTableVar, after delete'
SELECT * FROM @MyTableVar;

DROP TABLE dbo.table1;

--Results
--table1, before delete
--id         employee
------------ ------------------------------
--1         Fred
--2         Tom
--3         Sally
--4         Alice
--
--table1, after delete
--id         employee
------------ ------------------------------
--1         Fred
--3         Sally
--@MyTableVar, after delete
--id         employee
------------ ------------------------------
--2         Tom
--4         Alice
```

Examples:

A. Using output into with a simple insert statement:

```
USE AdventureWorks2012;
GO
DECLARE @MyTableVar table( NewScrapReasonID smallint,
          Name varchar(50),
          ModifiedDate datetime);
INSERT Production.ScrapReason
        OUTPUT INSERTED.ScrapReasonID, INSERTED.Name, INSERTED.ModifiedDate
            INTO @MyTableVar
VALUES (N'Operator error', GETDATE());

--Display the result set of the table variable.
SELECT NewScrapReasonID, Name, ModifiedDate FROM @MyTableVar;
--Display the result set of the table.
SELECT ScrapReasonID, Name, ModifiedDate
FROM Production.ScrapReason;
GO
```

B. Using output with a delete statement:

```
USE AdventureWorks2012;
GO
DELETE Sales.ShoppingCartItem
OUTPUT DELETED.*
WHERE ShoppingCartID = 20621;

--Verify the rows in the table matching the WHERE clause have been deleted.
SELECT COUNT(*) AS [Rows in Table] FROM Sales.ShoppingCartItem WHERE ShoppingCartID
= 20621;
GO
```

C. Using OUTPUT INTO with an UPDATE statement

```
USE AdventureWorks2012;

GO


DECLARE @MyTableVar table(

   EmpID int NOT NULL,

   OldVacationHours int,

   NewVacationHours int,

   ModifiedDate datetime);
```

```
UPDATE TOP (10) HumanResources.Employee
SET VacationHours = VacationHours * 1.25,
   ModifiedDate = GETDATE()
OUTPUT inserted.BusinessEntityID,
     deleted.VacationHours,
     inserted.VacationHours,
     inserted.ModifiedDate
INTO @MyTableVar;

--Display the result set of the table variable.
SELECT EmpID, OldVacationHours, NewVacationHours, ModifiedDate
FROM @MyTableVar;
GO
--Display the result set of the table.
SELECT TOP (10) BusinessEntityID, VacationHours, ModifiedDate
FROM HumanResources.Employee;
GO
```

D. Using output into to return an expression:

```
USE AdventureWorks2012;
GO
DECLARE @MyTableVar table(
   EmpID int NOT NULL,
   OldVacationHours int,
   NewVacationHours int,
   VacationHoursDifference int,
   ModifiedDate datetime);

UPDATE TOP (10) HumanResources.Employee
SET VacationHours = VacationHours * 1.25,
   ModifiedDate = GETDATE()
OUTPUT inserted.BusinessEntityID,
     deleted.VacationHours,
     inserted.VacationHours,
     inserted.VacationHours - deleted.VacationHours,
     inserted.ModifiedDate
INTO @MyTableVar;
```

```
--Display the result set of the table variable.
SELECT EmpID, OldVacationHours, NewVacationHours,
   VacationHoursDifference, ModifiedDate
FROM @MyTableVar;
GO
SELECT TOP (10) BusinessEntityID, VacationHours, ModifiedDate
FROM HumanResources.Employee;
GO
```

E. Using output into with from_table_name in an update statement:

```
USE AdventureWorks2012;
GO
DECLARE @MyTestVar table (
   OldScrapReasonID int NOT NULL,
   NewScrapReasonID int NOT NULL,
   WorkOrderID int NOT NULL,
   ProductID int NOT NULL,
   ProductName nvarchar(50)NOT NULL);

UPDATE Production.WorkOrder
SET ScrapReasonID = 4
OUTPUT deleted.ScrapReasonID,
      inserted.ScrapReasonID,
      inserted.WorkOrderID,
      inserted.ProductID,
      p.Name
   INTO @MyTestVar
FROM Production.WorkOrder AS wo
   INNER JOIN Production.Product AS p
   ON wo.ProductID = p.ProductID
   AND wo.ScrapReasonID= 16
   AND p.ProductID = 733;

SELECT OldScrapReasonID, NewScrapReasonID, WorkOrderID,
   ProductID, ProductName
FROM @MyTestVar;
GO
```

Dealing with variables:

Code:

```
CREATE PROCEDURE proc_vars()

SPECIFIC proc_vars

LANGUAGE SQL
```

```
BEGIN

 DECLARE v_rcount INTEGER;

 DECLARE v_max DECIMAL (9,2);

 DECLARE v_adate, v_another  DATE;

 DECLARE v_total INTEGER DEFAULT 0;          -- (1)

 DECLARE v_rowsChanged BOOLEAN DEFAULT FALSE; -- (2)

 SET v_total = v_total + 1;                  -- (3)

 SELECT MAX(salary)                          -- (4)
  INTO v_max FROM employee;

 VALUES CURRENT_DATE INTO v_date;            -- (5)

 SELECT CURRENT DATE, CURRENT DATE           -- (6)
    INTO v_adate, v_another
 FROM SYSIBM.SYSDUMMY1;

 DELETE FROM T;
 GET DIAGNOSTICS v_rcount = ROW_COUNT;       -- (7)

 IF v_rcount > 0 THEN                        -- (8)
  SET is_done = TRUE;
 END IF;
END
```

Inserting and using Database data:

SQL insert into statement:

- The INSERT INTO statement is used to add new data to a database
- The INSERT INTO statement adds a new record to a table
- INSERT INTO can contain values for some or all of its columns
- INSERT INTO can be combined with a SELECT to insert records

The SQL insert into syntax:

The general syntax:

```
INSERT INTO table-name (column-names)
VALUES (values)
```

SQL insert into examples:

```
INSERT INTO Customer (FirstName, LastName, City, Country, Phone)
VALUES ('Craig', 'Smith', 'New York', 'USA', 1-01-993 2800)
```

Results: 1 new record inserted.

The SQL insert combined with a select:

```
INSERT INTO table-name (column-names)
SELECT column-names
  FROM table-name
 WHERE condition
```

SQL insert into with select example:

```
INSERT INTO Customer (FirstName, LastName, City, Country, Phone)
SELECT LEFT(ContactName, CHARINDEX(' ',ContactName) - 1),
      SUBSTRING(ContactName, CHARINDEX(' ',ContactName) + 1, 100),
      City, Country, Phone
  FROM Supplier
 WHERE CompanyName = 'Bigfoot Breweries'
```