Daily Assessment Journal

Date: 29/05/2020 Name: Jyoti S. Doni

Course: Logic design USN: 4AL17EC037

Topic: Analysis of clocked sequential ckts

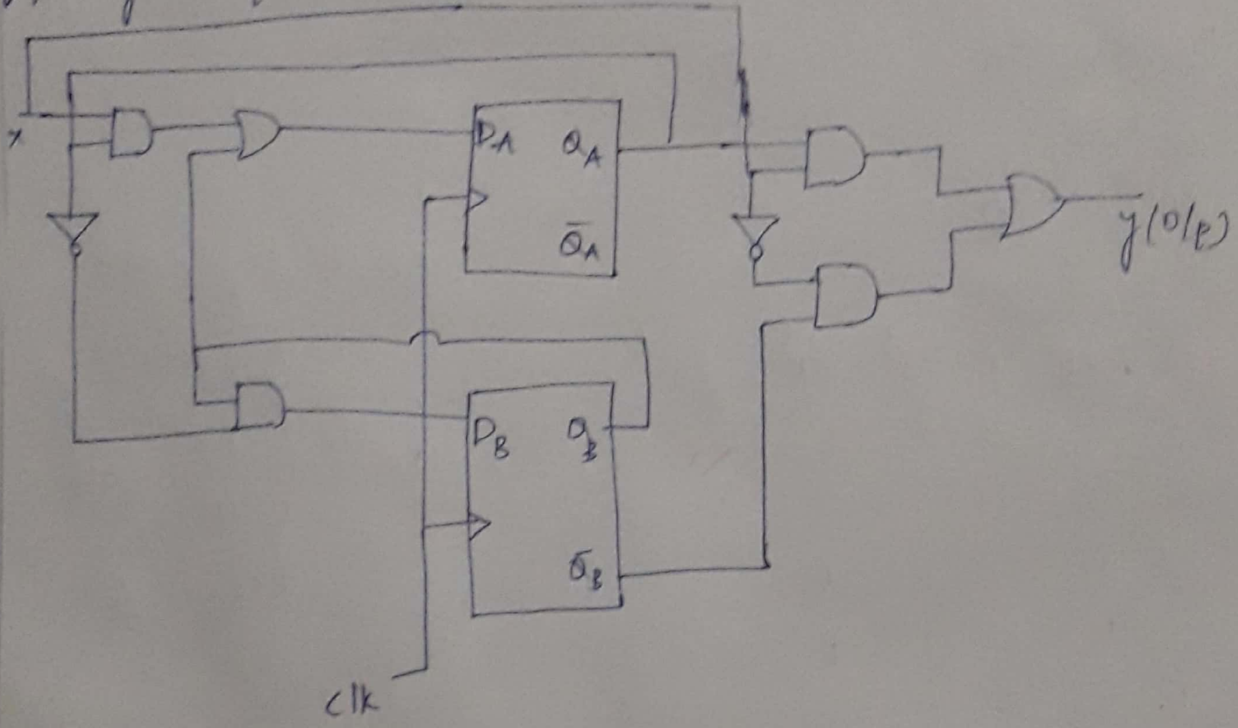
· Digital clock signals

Github repository: jyoti-courses

forenoon session details

Image of session

Analysis of clocked sequential ckts (with DFF)



state table

P.S			N.S		
Q_A	Q_B	x	Q_A^+	Q_B^+	y
0	0	0	0	0	1
0	0	1	0	0	0

$Q_A^+ = D_A = xQ_A + Q_B$
$= 0.0 + 0 = 0$

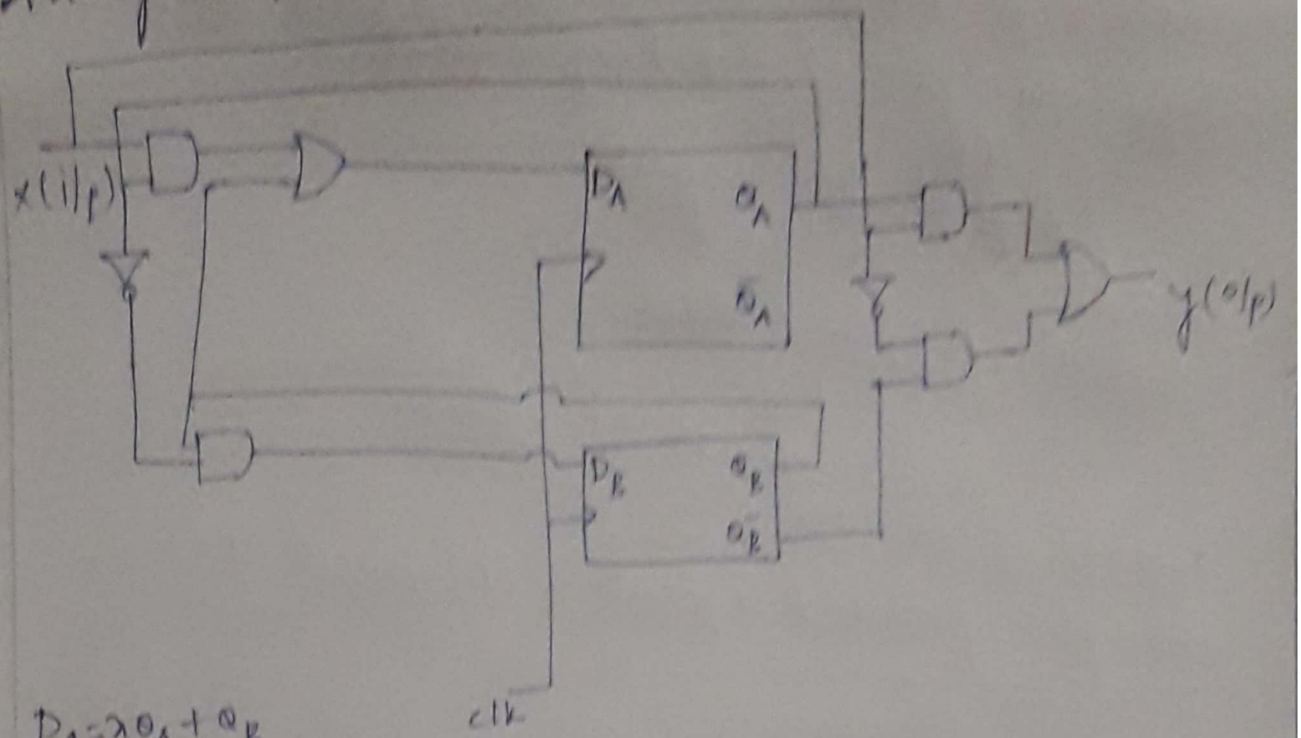
$Q_B^+ = D_B = \bar{Q}_A Q_B$
$= 1.0 = 0$

$y = 1.1 + 0.0 = 1$

$y = 0.1 + 1.0 = 0$

Report

Analysis of clocked sequential ckt



$D_A = xQ_A + Q_B$

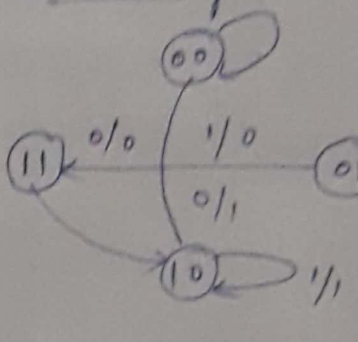
$D_B = Q_A Q_B$

$y = x\bar{Q}_B + xQ_A$

state table

Q_A	Q_B	x	Q_A^+	Q_B^+	y
0	0	0	0	0	1
0	0	1	0	0	0
0	1	0	1	1	0
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	1
1	1	0	1	0	0
1	1	1	1	0	1

state diagram



Digital clk design

A digital clk is a type of clk that displays the time digitally at opposed to an analogue clk where the time indicated by the portions of hands.

Analyses of clocked sequential ckts

- Some flipflops have asynchronous inputs that are used to force the flipflop to a particular state independently of the clk.
- the input that sets the flip-flop to 1 is called preset or direct set. the input that clears the flipflop to 0 is called clear or direct reset.
- when power is turned on in a digital system, the state of the flip-flop is unknown, the direct inputs are useful for bringing all flip-flops in the system to a known starting state prior to the clocked operation.

Scanned with CamScanner

Date: 29/05/2020
course: python
Topic: object oriented programming

Name: Jyothi S Donni
Usn: 4AL17EC037

Afternoon session details

image of session

```
import sqlite3
class Database:
    def connect():
        conn=sqlite3.connect("books.db")
        cur=conn.cursor()
        cur.execute("create table if not exists book
         (id integer primary key, title text, autho
         text, year integer, isbn integer)")
        conn.commit()
        conn.close()
    def insert(title,author,year,isbn):
       conn=sqlite3.connect("books.db")
       cur=conn.cursor()
       cur.execute("insert into book values(null,
        ?,?,?,?)",(title,author.year,isbn))
       conn.commit()
       conn.close()
    def view():
       conn=sqlite3.connect("books.db")
       cur=conn.cursor()
       cur.execute("select.from book")
      rows=cur fetchall()
      conn.close()
      return rows
   def search(title="",author="",year="",isbn
                     =""):
   conn=sqlite3.connect("books.db")
```

Here are the frontend.py & backened.py scripts in oop style to execute this program you should execute the frontend.py file

```
# frontend.py
from tkinter import*
from backend import database
database=Database("books.db")
class window(object):
    def_init_(self,window):
        self.window=window
        self.window.wm_title("BookStore")
        l1=Label(window,text="Title")
        l1.grid(row=0, column=0)
        l2=Label(window, text = "year")
        l2.grid(row=0,column=2)
        l3=Label(window, text = "Author")
        l3=Label(window, column=0)
        l3.grid(row=1,column=0)
        l4=Label(window,text=" Isbn")
        l4.grid(row=1, column=2)

        self.title_text=stringVar()
        self.e1=Entry(window,textvariable=self.title_text)
        self.e1.grid(row=0, column=1)

        self.Author_text=stringVar()
        self.e2=Entry(window,textvariable=self.title_text)
        self.e2.grid(row=0, column=3)

        self.year_text=stringVar()
        self.e3=Entry(window,textvariable=self.year_text)
        self.e3.grid(row=1,column=1)
self.isbn_text=StringVar()
self.e4=Entry(window, textvariable=self.isbn_text)
self.e4.grid(row=1, column=3)

self.list1=Listbox(window, height=6, width=35)
self.list1.grid(row=2, column=0, rowspan=6, columnspan
            =2)

sb1=Scrollbar(window)
sb1.grid(row=2, column=2, rowspan=6)

self.list1.configure(yscrollcommand=sb1.set)
sb1.configure(command=self.list1.yview)

self.list1.blind('<<ListboxSelect>>' self.get_selected_row)
b1=Button(window, text="view all", width=12, command=self
      view_command)
b1.grid(row=2, column=3)
b2=Button(window, text="search entry", width=12, command=
       self.search_command)
b2.grid(row=3, column=3)
b3=Button(window, text="Add entry", width=12, command=
      self.add_command)
b3.grid(row=4, column=3)
b4=Button(window, text="update selected", width=12, comma
        = self.update_command)
b4.grid(row=5, column=3)
b5=Button(window, text="Delete selected", width=12, comma
      = self.delete_command)
b5.grid(row=6, column=3)
b6=Button(window, text="close", width=12, command=
      self.destroy)
b6.grid(row=7, column=3)
```

```
window=Tk()
window(window)
window(mainloop)
```

And below you will also find the backend.py script in oop

```
#backend.py
import sqlite3
class Database:
    def_init_(self,db):
        self.conn=sqlite3.connect(db)
        self.cur=self.conn.cursor()
        self.cur.execute("CREATE TABLE IF NOT EXISTS book(
          id INTEGER PRIMARY KEY,title text, author text,
          year integer,isbn integer")
        self.conn.commit()
        def insert(self,title,author,year,isbn):
          self.cur.execute("INSERT INTO book VALUES(NULL,?,?,
             (title,author,year,isbn))
          self.conn.commit()
    def view(self):
        self.cur.execute("SELECT* from book")
        rows=self.cur.fetchall()
        return rows
    def search(self,title="",author="",year="",isbn=""):
        self.cur.execute("SELECT*FROM book WHERE title=? OR
           author=?,year=? OR isbn=?",(title,author,
                 year,isbn))
        rows=self.cur.fetchall()
        return rows
        def delete(self,id):
          self.cur.execute("DELETE FROM book WHERE id=?
                (id,))
```

```
def get_selected_row(self,event):
    index=self.list1.curselection()[0]
    self.selected_tuple=self.list1.get(index)
    self.e1.delete(0,END)
    self.e1.insert(END,self.selected_tuple[1])
    self.e2.delete(0,END)
    self.e2.insert(END,self.selected_tuple[2])
    self.e3.delete(0,END)
    self.e3.insert(END,self.selected_tuple[3])
    self.e4.delete(0,END)
    self.e4.insert(END,self.selected_tuple[4])

def view_command(self):
    self.list1.delete(0,END)
    for row in database.view():
        self.list1.insert(END,row)
def search_command(self):
    self.list1.delete(0,END)
    for row in
    database.search(self.title_text.get(),self.author_text.get()
        self.year=text.get(),self.isbn_text.get():
    self.list1.insert(END,row)
def add_command(self):
database.insert(self.title_text.get(), self.author_text.get(),
    self.year_text.get(), self.isbn_text.get()).
self.list1.delete(0,END)
self.list1.insert(END,(self.title_text.get(),self.author_text.
    get(), self.year_text.get(), self.isbn_text get())

def delete_command(self):
    database.delete(self.selected_tuple[0])
def update_command(self):
database.update(self.selected_tuple[0],self.title_text.get(),
    self.author_text.get(), self.year_text.get(), self.isbn_
                text.get()))
    self.conn.commit()
def update(self,id,title,author,year,isbn):
    self.cur.execute("UPDATE book SET title=?, author=?,
            year=?, isbn=? WHERE id=?", (title,author,year,
            isbn))
    self.conn.commit()
def_del_(self):
     self.conn.close()
```

Date: 29/05/2020

Name: Jyoti. S. Donne

USN: 4AL17EC037

Course:

Topic: preparation for the next normal

<u>Report</u>

<u>Case study</u>

There will be thousands of case studies written in the next one year post covid. will you be an inspiring case study <u>or</u> a 'what not to do" case study!

weiji (Crisis)

<u>wei (Danger)</u>

- suffer from short-term setbacks
- React to ensure business continuity & minimize negative impacts.

<u>Ji</u> (opportunity)

- Rebalance resources & investment
- proactively position for rebound & Long-term opportunities.

<u>Some industries are more impacted than others</u>

- Apparel
- Automotive manufacture
- Automotive suppliers
- consumer Durables
- Gaming
- Beverages
- chemicals
- manufacturing
- media
- metals & mining
- oils. gas/ oilfield services
- property developers
- protein & agriculture
- services companies
- steel producers
- Technology hardware
- Telecoms
- waste management
- Food/ food retails
- Rental
- packaging etc.

challenges during times of disruption

Organizational barriers

- inability to plan effectively due to uncertainties
- financial constraints

Shifting customer emotions

Delight ⟶ Anxiety
trust ⟶ skepticism
Loyalty ⟶ Indifference

Business impact

$96B ⟶ crowd funding activity by 2025
75% ⟶ of total work force by 2025 will be millennials
50% ⟶ of all college courses will be offered online

We can't solve problems with the same thinking we used when we created them. ⟶ Alexander

If you want something new, you have to stop doing something old ⟶ peter. Drucker

Preparation for the next normal

- Education "Anywhere Anytime Anyplace" will be the future
- Digital transformation
- Resilient Dynamism
- Economic crisis
- Rethink business models
- video & Audio collaboration tools
- Empathy & tolerance

the future of work

- led world ⟶ GIG crowd Economy, fueled by innovations, AI, future technologies.

- Green world $\longrightarrow$ companies care, social responsibility, Embedding sustainability
- Orange & blue world $\longrightarrow$ small is beautiful, collaboration networking & specialisation
  corporate is king big company capitalism

Defining GIG
use of outside on-demand labour to perform work
- collaborative economy
- on-demand economy
- online economy
- crowd economy
- 1099 economy
- passion economy
- sharing "
- freelance "
- matching "
- talent marketplace
- Labour cloud

Benefits of on-demand GIG talent
- speed
- scale
- Resiliency