**DAY 11 ASSIGNMENT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **29-05-2020** | **Name:** | **Ashish Shanbhag** |
| **Course:** | **Logic Design** | **USN:** | **4AL16EC008** |
| **Topic:** | 1. **Analysis of clocked sequential circuits** 2. **Digital clock design** | **Semester & Section:** | **8th A** |
| **Github Repository:** | **Ashish Shanbhag** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS**      **Analysis of clocked sequential circuits with D Flipflop**  The D-type flip-flop is a modified Set-Reset flip-flop with the addition of an inverter to prevent the S and R inputs from being at the same logic level. The D Flip Flop is by far the most important of the clocked flip-flops as it ensures that ensures that inputs S and R are never equal to one at the same time. The D-type flip flop are constructed from a gated SR flip-flop with an inverter added between the S and the R inputs to allow for a single D (Data) input. Then this single data input, labelled “D” and is used in place of the “Set” signal, and the inverter is used to generate the complementary “Reset” input thereby making a level-sensitive D-type flip-flop from a level-sensitive SR-latch as now S = D and R = not D as shown.  Single input is called the “DATA” input. If this data input is held HIGH the flip flop would be “SET” and when it is LOW the flip flop would change and become “RESET”. However, this would be rather pointless since the output of the flip flop would always change on every pulse applied to this data input. To avoid this an additional input called the “CLOCK” or “ENABLE” input is used to isolate the data input from the flip flop’s latching circuitry after the desired data has been stored. The effect is that D input condition is only copied to the output Q when the clock input is active. This then forms the basis of another sequential device called a D Flip Flop. The “D flip flop” will store and output whatever logic level is applied to its data terminal so long as the clock input is HIGH. Once the clock input goes LOW the “set” and “reset” inputs of the flip-flop are both held at logic level “1” so it will not change state and store whatever data was present on its output before the clock transition occurred. In other words the output is “latched” at either logic “0” or logic “1”.  The flip-flop input equations are:  D A = X xor B' D B = X or A  The sequential circuit output equation is:  Z = A xor B  The next-state equations for the flip-flops are:  A + = D A = X xor B' B + = D B = X or A  **The state table or transition table**    **The state diagram can then be drawn from the state table**    **Digital clock design**    The main parts of the circuit are as follows:  1- **Timer 555**: Responsible for generating the clock pulses for the counters, the frequency of the output shoul be 1 hz which means 1 second for each pulse.  2- **Counters**: Responsible for generating the time in BCD (Binary Coded decimal).  3- **Decoders**: Takes the BCD of the counter as input and produces 7 segment output .  4- 7 **segments**: Displays the time, of course!  \*Seconds have 2 displays , 2 decoders and 2 counters. The same for minutes and hours.  **The circuit works as follows :**  555 timer produces 1 second pulses to the clock input of the first counter which is responsible the first column of seconds, so its output will change every second.  The counter produces numbers from 0 to 9 in BCD form and automatically resets to 0 after that. So the output of the first counter will count from 0 to 9 every second and that's exactly what we want from it, so we are done here. let's move to the next one.  **What do we want here?**  **First**, we want the 2nd counter to start counting when the 1st one moves for 9 to 0 (that makes 10 seconds!)  **How can this be done?**  let's check the output of the first counter in BCD :  MSB---LSB  0:  0000  1:  0001  2:  0010  3:  0011  4:  0100  5:  0101  6:  0110  7:  0111  8:  1000  9:  1001  0:   0000 |

**DAY 11 ASSIGNMENT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **29-05-2020** | **Name:** | **Ashish Shanbhag** |
| **Course:** | **PYTHON** | **USN:** | **4AL16EC008** |
| **Topic:** | **Python for Image and Video Processing with OpenCV** | **Semester & Section:** | **8th A** |
| **Github Repository:** | **Ashish Shanbhag** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS**      **Python for Image and Video Processing with OpenCV**  Python provides lots of libraries for image processing, including −   * **OpenCV** − Image processing library mainly focused on real-time computer vision with application in wide-range of areas like 2D and 3D feature toolkits, facial & gesture recognition, Human-computer interaction, Mobile robotics, Object identification and others. * **Numpy and Scipy libraries** − For image manipuation and processing. * **Sckikit** − Provides lots of alogrithms for image processing. * **Python Imaging Library (PIL)** − To perform basic operations on images like create thumnails, resize, rotation, convert between different file formats etc.   **Python Code for Loading, Displaying, Resizing, and Writing Images in python**  import cv2  img cv2.imread ("galaxy. jpg".0)  print(type(img)  print(img)  print(img.shape)  print(img.ndim)  resize\_image=cv2.resize(img,(img.shape[1] ing.shape[0]))  cv2.imshow( "Galaxy",resized\_image)  cv2.waitkey(0)  cv2.destroyAllWindows()  **Face Detection**  OpenCV is the most popular library for computer vision. Originally written in C/C++, it now provides bindings for Python. OpenCV uses machine learning algorithms to search for faces within a picture. Because faces are so complicated, there isn’t one simple test that will tell you if it found a face or not. Instead, there are thousands of small patterns and features that must be matched. The algorithms break the task of identifying the face into thousands of smaller, bite-sized tasks, each of which is easy to solve.  **Python Code**  import cv2  face\_cascade=CV2.CascadeClassifier("haarcascade\_frontalface\_default.xml")  img-cv2.imread("news.jpg")  gray\_img=cv2.cvtcolor(img.cv2.COLOR\_BGR2GRAY)  faces=face\_cascade.detectMultiscale(gray\_img.scalefactor-1.es, minNeighbors=5)  for x, y, w, h in faces:  img-cv2.rectangle(img, (x,y),(x+w,y+h), (0,255,0),3)  print(type(faces))  print(faces)  resized-cv2.resize(ing, ( Int(img.shape[1]/3), int(ing.shape[0]/3))) |