**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **01/06/2020** | **Name:** | **Namratha S Hipparagi** |
| **Course:** | **HDL design** | **USN:** | **4AL16EC040** |
| **Topic:** | **Industry Applications of FPGA**  **FPGA vs ASIC Design Flow** | **Semester & Section:** | **8 A** |
| **Github Repository:** | **namrathahipparagi\_1** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session** |
| **Report**  A field-programmable gate array is an integrated circuit designed to be configured by a customer or a designer after manufacturing – hence the term "field-programmable". An FPGA is a (mostly) digital, (re-)configurable ASIC.  I put re- in parenthesis because there are actually one-time-programmable FPGAs, where once you configure them, that’s it, never again.  However, most FPGAs you’ll come across are going to be re-configurable.  I say mostly because there are analog and mixed-signal aspects to modern FPGAs.  For example, some have A/D converters and PLLs.  FPGA-basics-gates   1. **High data-to-clock-rate-ratio** –The advantage is that you’re not tying up a centralized processor. Each function can operate on its own. 2. **Large quantities of deterministic I/O** – If there are too many operations within your required loop rate on a sequential processor, you may not even have enough time to close the loop to update all of the I/O within the allotted time. The amount of determinism that you can achieve with an FPGA will usually far surpass that of a typical sequential processor. 3. **Parallel processes** – if you need to process several input channels of information (e.g. many simultaneous A/D channels) or control several channels at once (e.g. several PID loops).   **Signal processing** – includes algorithms such as digital filtering, demodulation, detection algorithms, frequency domain processing, image processing, or control algorithms.   1. **Sorting/searching** – this really falls into the category of a sequential process. There are algorithms that attempt to reduce the number of computations involved, but in general, this is a sequential process that doesn’t easily lend itself to efficient use of parallel logical resources. Check out the sorting section [here](http://bigocheatsheet.com/) and check out this article [here](http://www.embedded.com/design/configurable-systems/4006440/Sorting-data-in-two-clock-cycles) for some more info. 2. **Complex calculations infrequently** – If the majority of your algorithms only need to make a computation less than 1% of the time, you’ve generally still allocated those logic resources for a particular function (there are exceptions to this), so they’re still sitting there on your FPGA, not doing anything useful for a significant amount of time. 3. **Floating point arithmetic** – historically, the basic arithmetic elements within an FPGA have been fixed-point binary elements at their core. In some cases, floating point math can be achieved (see [Xilinx FP Operator](http://www.xilinx.com/products/intellectual-property/floating_pt.html) and [Altera FP White Paper](https://www.altera.com/en_US/pdfs/literature/wp/wp-01028.pdf) ), but it will chew up a lot of logical resources. 4. **Very low power** – Some FPGAs have low power modes (hibernate and/or suspend) to help reduce current consumption, and some may require external mode control ICs to get the most out of this.. 5. **Very low cost** – while FPGA costs have come down drastically over the last decade or so, they are still generally more expensive than sequential processors. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date:** | **01/6/2020** | **Name:** | **Namratha S Hipparagi** | |
| **Course:** | **Python** | **USN:** | **4al16ec040** | |
| **Topic:** | **Application 6: Build a Webcam Motion Detector** | **Semester & Section:** | **8 A** | |
| **AFTERNOON SESSION DETAILS** | | | |
| **REPORT**  Motion detection is the technique that make this magic happened, it can detect a change in the position of an object relative to its surroundings. And it don't need any motion sensor, all it need is to turn on the webcam on your PC or Mobile. Videos can be treated as stack of pictures called frames. We compare two images by comparing the intensity value of each pixels. In python we can do it easily as you can see in following code:  # Python program to implement  # Webcam Motion Detector  # importing OpenCV, time and Pandas library  import cv2, time, pandas  # importing datetime class from datetime library  from datetime import datetime  # Assigning our static\_back to None  static\_back = None  # List when any moving object appear  motion\_list = [ None, None ]  # Time of movement  time = []  # Initializing DataFrame, one column is start  # time and other column is end time  df = pandas.DataFrame(columns = ["Start", "End"])  # Capturing video  video = cv2.VideoCapture(0)  # Infinite while loop to treat stack of image as video  while True:  # Reading frame(image) from video  check, frame = video.read()  # Initializing motion = 0(no motion)  motion = 0  # Converting color image to gray\_scale image  gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  # Converting gray scale image to GaussianBlur  # so that change can be find easily  gray = cv2.GaussianBlur(gray, (21, 21), 0)  # In first iteration we assign the value  # of static\_back to our first frame  if static\_back is None:  static\_back = gray  continue  # Difference between static background  # and current frame(which is GaussianBlur)  diff\_frame = cv2.absdiff(static\_back, gray)  # If change in between static background and  # current frame is greater than 30 it will show white color(255)  thresh\_frame = cv2.threshold(diff\_frame, 30, 255, cv2.THRESH\_BINARY)[1]  thresh\_frame = cv2.dilate(thresh\_frame, None, iterations = 2)  # Finding contour of moving object  cnts,\_ = cv2.findContours(thresh\_frame.copy(),  cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  for contour in cnts:  if cv2.contourArea(contour) < 10000:  continue  motion = 1  (x, y, w, h) = cv2.boundingRect(contour)  # making green rectangle arround the moving object  cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 3)  # Appending status of motion  motion\_list.append(motion)  motion\_list = motion\_list[-2:]  # Appending Start time of motion  if motion\_list[-1] == 1 and motion\_list[-2] == 0:  time.append(datetime.now())  # Appending End time of motion  if motion\_list[-1] == 0 and motion\_list[-2] == 1:  time.append(datetime.now())  # Displaying image in gray\_scale  cv2.imshow("Gray Frame", gray)  # Displaying the difference in currentframe to  # the staticframe(very first\_frame)  cv2.imshow("Difference Frame", diff\_frame)  # Displaying the black and white image in which if  # intensity difference greater than 30 it will appear white  cv2.imshow("Threshold Frame", thresh\_frame)  # Displaying color frame with contour of motion of object  cv2.imshow("Color Frame", frame)  key = cv2.waitKey(1)  # if q entered whole process will stop  if key == ord('q'):  # if something is movingthen it append the end time of movement  if motion == 1:  time.append(datetime.now())  break  # Appending time of motion in DataFrame  for i in range(0, len(time), 2):  df = df.append({"Start":time[i], "End":time[i + 1]}, ignore\_index = True)  # Creating a CSV file in which time of movements will be saved  df.to\_csv("Time\_of\_movements.csv")  video.release()  # Destroying all the windows  cv2.destroyAllWindows()  **Threshold Frame :**If the intensity difference for a particular pixel is more than 30(in my case) then that pixel will be white and if the difference is less than 30 that pixel will be black    **Color Frame :**In this frame you can see the color images in color frame along with green contour around the moving objects    **Difference Frame :** Difference frame shows the difference of intensities of first frame to the current frame. | | | |
|  | | | |