**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **1/6/2020** | **Name:** | **CHANDANA.R** |
| **Course:** | Digital design using HDL | **USN:** | **4AL16EC017** |
| **Topic:** | Industry applications of FPGA,  FPGA Business Fundamentals  FPGA vs ASIC Design flow  FPGA Basics-A look under the hood. | **Semester & Section:** | **8(A)** |
| **Github Repository:** | **chandana-shaiva** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session** |
| **Report –**   * **ASIC:** * ASIC stands for Application-Specific Integrated Circuit. Furthermore, as the name implies, it is a chip that serves the purpose for which it has been designed and does not permit reprogramming or modification. * Which, in turn, means that it cannot perform another function or execute another application once programming is complete.Since the ASIC’s design is for a specific function, * **FPGA:** * [Field Programmable Gate Array or FPGA](https://www.cadence.com/content/cadence-www/global/en_US/home/tools/pcb-design-and-analysis/design-authoring/fpga-planner.html) is in direct competition with ASIC chip technology. * Also, FPGA is, in essence, a chip that can be programmed and reprogrammed to perform numerous functions at any single point in time.Furthermore, a single chip is comprised of thousands of units called logic blocks, that are linked with programmable interconnects. * The FPGA’s circuit is made by connecting several configurable blocks, and it has a rigid internal structure. In summary, an FPGA is essentially a programmable version of an ASIC.      * Overall, the FPGA affords general functionality that allows programming to your specifications. * However, like most things in life, there are side effects of FPGA’s versatility. In this case, it is an increased cost, increased internal delay, and limited analog functionality. * **FPGA versus ASIC: A Side-By-Side Comparison** * **NRE**: NRE stands for Non-Recurring Engineering costs. As you can imagine, with the words recurring and costs, in the same sentence, every business is concerned when they hear those two words. * So, it is safe to say that this is an essential deciding factor. Moreover, in the case of ASIC, this is exceptionally high, whereas, with FPGA, it is nearly non-existent. * However, in the grand scheme, the total cost gets lower and lower the more significant the quantity you require in terms of ASIC. Furthermore, FPGA can cost you more overall since its individual costs are higher per unit than ASIC. * **Design Flow**: Every engineer and PCB designer prefer a more trouble-free and simplistic design process. Just because what you do is complex, does not mean that you want the process itself to be complicated. * Therefore, in terms of the simplicity of design flow, FPGA is hands down less complicated than ASIC. * This is due to the FPGA’s flexibility, versatility, shorter time to market, and the fact that it is reprogrammable. Whereas, with ASIC, it is more involved in terms of design flow because it is not reprogrammable, and it requires costly [dedicated EDA tools](https://resources.pcb.cadence.com/blog/2019-eda-tools-for-pcb-design-what-you-should-be-looking-for) for the design process. * **Performance and Efficiency**: In terms of performance, ASICs outperforms FPGAs by a small margin, primarily due to lower power consumption and the various possible functionalities that you can layer onto a single chip. * Also, FPGA has a more rigid internal structure, whereas, with an ASIC, you can design it to excel in power consumption or speed. * **Cost**: Even with the increased NRE cost, ASICs are thought to be more cost-effective, all things considered as compared to FPGAs, which are only profitable when developed in smaller quantities. * **Power Consumption**: As I mentioned previously, ASICs require less power and thus provide a better option than the higher power consumption FPGA. Especially with electronic devices that are battery operated. * **Size**: In terms of size, it is a matter of physics. With an ASIC, its design is for one functionality; therefore, it consists of precisely the number of gates required for the desired application. However, with FPGA’s multifunctionality, a single unit will be significantly larger, because of its internal structure and a specific size that you cannot change. * **Time to Market**: Also, as mentioned earlier, FPGA affords a faster time to market than ASIC due to its simplicity in terms of the design flow. Moreover, ASICs also require layouts, back end processes, and advanced verification, all of which are time-consuming.   **TASK FOR DAY 1**  **Verilog code for NAND gate – All modelling styles**  **GATE LEVEL MODELLING**    module NAND\_2\_gate\_level (output Y, input A, B);  wire Yd;  and (Yd,A,B);  not(Y,Yd);  endmodule  **DATA FLOW MODELLING**  module NAND\_2\_data\_flow(output Y,input A,B);  assign Y = ~(A & B);  endmodule  **BEHAVIOURAL MODELLING**  module NAND\_2\_behavioural(output reg Y,input A,B) ;  always @(A or B) begin  if( A == 1’b1 & B == 1’b1) begin  Y =1’b0;  end  else  Y = 1’b1;  end  endmodule  **TESTBENCH OF THE NAND GATE USING VERILOG**  `include "NAND\_2\_behavioral.v"  module NAND\_2\_behavioral\_tb;  reg A, B;  wire Y;  NAND\_2\_behavioral Indtance0 (Y, A, B);  initial begin  A = 0; B = 0;  #1 A = 0; B = 1;  #1 A = 1; B = 0;  #1 A = 1; B = 1;  end  initial begin  $monitor ("%t | A = %d| B = %d| Y = %d", $time, A, B, Y);  $dumpfile("dump.vcd");  $dumpvars();  end  endmodule |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date:** | **1/6/2020** | **Name:** | **CHANDANA.R** | |
| **Course:** | **Python** | **USN:** | **4AL16EC017** | |
| **Topic:** | **Application 6: Build a webcam motion detector** | **Sem &Section:** | **8( A)** | |
| **Github Repository:** | **chandana-shaiva** |  |  | |
| **AFTERNOON SESSION DETAILS** | | | |
| **SESSION IMAGE :**    **REPORT:** Code: importcv2  importsys  cascPath=sys.argv[1]  faceCascade=cv2.CascadeClassifier(cascPath)  video\_capture=cv2.VideoCapture(0)  whileTrue:  # Capture frame-by-frame  ret,frame=video\_capture.read()  gray=cv2.cvtColor(frame,cv2.COLOR\_BGR2GRAY)  faces=faceCascade.detectMultiScale(  gray,  scaleFactor=1.1,  minNeighbors=5,  minSize=(30,30),  flags=cv2.cv.CV\_HAAR\_SCALE\_IMAGE  )  # Draw a rectangle around the faces  for(x,y,w,h)infaces:  cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),2)  # Display the resulting frame  cv2.imshow('Video',frame)  ifcv2.waitKey(1)&0xFF==ord('q'):  break  # When everything is done, release the capture  video\_capture.release()  cv2.destroyAllWindows()  Now let’s break it down…  importcv2  importsys  cascPath=sys.argv[1]  faceCascade=cv2.CascadeClassifier(cascPath)  This should be familiar to you. We are creating a face cascade, as we did in the image example.  video\_capture=cv2.VideoCapture(0)  This line sets the video source to the default webcam, which OpenCV can easily capture.  **NOTE**: You can also provide a filename here, and Python will read in the video file. However, you need to have [ffmpeg](https://www.ffmpeg.org/) installed for that since OpenCV itself cannot decode compressed video. Ffmpeg acts as the front end for OpenCV, and, ideally, it should be compiled directly into OpenCV. This is not easy to do, especially on Windows.  whileTrue:  # Capture frame-by-frame  ret,frame=video\_capture.read()  Here, we capture the video. The read() function reads one frame from the video source, which in this example is the webcam. This returns:   1. The actual video frame read (one frame on each loop) 2. A return code   The return code tells us if we have run out of frames, which will happen if we are reading from a file. This doesn’t matter when reading from the webcam, since we can record forever, so we will ignore it.  # Capture frame-by-frame  ret,frame=video\_capture.read()  gray=cv2.cvtColor(frame,cv2.COLOR\_BGR2GRAY)  faces=faceCascade.detectMultiScale(  gray,  scaleFactor=1.1,  minNeighbors=5,  minSize=(30,30),  flags=cv2.cv.CV\_HAAR\_SCALE\_IMAGE  )  # Draw a rectangle around the faces  for(x,y,w,h)infaces:  cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),2)  # Display the resulting frame  cv2.imshow('Video',frame)  Again, this code should be familiar. We are merely searching for the face in our captured frame.  ifcv2.waitKey(1)&0xFF==ord('q'):  break  We wait for the ‘q’ key to be pressed. If it is, we exit the script.  # When everything is done, release the capture  video\_capture.release()  cv2.destroyAllWindows()  **Object Classification**  **Colour Threshold**  Plastic Play Pit Ball as Object to Detect  I opted to detect the plastic ball by colour so I need to set the colour range that I can use to classify each coloured ball.   |  |  | | --- | --- | | 1 | blue=(104,117,222,121,255,255) |   I use an array to hold the values of the lower and upper colour threshold. The colour threshold uses the HSV (Hue Saturation Value) colour profile.  **Colour Conversion**   |  |  | | --- | --- | | 1 | frameHSV=cv2.cvtColor(frame,cv2.COLOR\_BGR2HSV) |   Before I can use the HSV profile values that define the coloured ball, I must convert the captured webcam frame to the HSV colour profile.  **Create a Mask**  Object Detection Mask   |  |  | | --- | --- | | 1  2  3 | colorLow=np.array([lowHue,lowSat,lowVal])  colorHigh=np.array([highHue,highSat,highVal])  mask=cv2.inRange(frameHSV,colorLow,colorHigh) |  * Here I use the colour range I set for the ball to create a mask.  The mask will make it more efficient to find contours around the detected object. * The white area is the area of interest that was found within the colour range set above. I can use various image filters to improve the image mask. However, applying filters to get the perfect mask can be expensive in regards to processing power.  **OpenCV Contours** **Find Contours**   |  |  | | --- | --- | | 1 | im2,contours,hierarchy=cv2.findContours(mask,cv2.RETR\_TREE,cv2.CHAIN\_APPROX\_SIMPLE) |   Now that I have an image mask to work with I can proceed with finding contours.  The third parameter, contour approximation method, will collect only the endpoint coordinates of straight lines. All the white blobs in the mask will have contours applied. The array list of found contours will be in the contours variable.  **(Optional) Draw all Contours**  OpenCV Find Contours Example   |  |  | | --- | --- | | 1 | cv2.drawContours(frame,contours,-1,(0,255,0),3) |   Two objects are detected, and some of the darker blue colour is out of range of the threshold set earlier. If the quality of the mask is noisy, there is going to be a lot of contours – many small contours. To get useful object detection, I need to remove the contours I do not need.  The image above shows a contour circling the plastic ball; I will aim to isolate that in the next step.  **(Optional) Finding The Largest Contour**  Finding Largest Contour Example – OpenCV Method   |  |  | | --- | --- | | 1  2  3 | contour\_sizes=[(cv2.contourArea(contour),contour)forcontour incontours]  biggest\_contour=max(contour\_sizes,key=lambdax:x[0])[1]  cv2.drawContours(frame,biggest\_contour,-1,(0,255,0),3) |   The above OpenCV Python code finds the biggest contour out of all the contours found. And then draw the biggest contour on to the original image. Since the ball is the largest blue object, I can reliably detect the ball. However, the ball must remain the dominant blue object and remain in focus to be tracked reliably.  So at this point, I was able to improve OpenCV object detection. I am now only detecting one item. The quality of the object detection is very good in this case. | | | |