Exercise 1:

For Array A we use random generated number % by 60001 then minus the number by 30000 to make sure the range of the value is between -300,000 to +30,000. For Array B we use the same method to make sure the value is in the correct range. (-1000 to +1000) to compute C, first we divided C array to 4 part(top_left, top_right,bottom_left,bottom_right). We do dot product and update the value to the part of C array. After all the multiplication was done, we merged the value to one array and the post the value to mailbox. In the end we get the merged value and print it out.

Exercise 2:

In this exercise after getting the greyscale image by changing the write task for R G and B along with the equation for Y val, I had to create to increase the NUM_LINE to 7 or something higher since R_val_two along with new G and B was added. I have decided to create a new task for second read queue for the second image that is being read, which is a duplicate of the first read queue without the readimagewidth being posted again as its not needed/no point to duplicate as once the mailbox has posted and pended. All the computation for our implementation was done in the compute Y task after the data has been feed. In the loop starting with j equal to 0 till it reaches the image width which is 640. The whole line is done first as R G B is being computed we use module %2 = 1 so only 320 columns of the first image is taken, with only odd pixels. Whereas for the second image we use the condition %2 = 0 where y_val[image height/2 [even++] the even pixels are taken starting from 320 instead of 0, so the second image can display from half where again only 320 pixels will be taken of the 640, where the odd is discarded.

Exercise 3:

For this exercise, we came up with more small problems then expected as we realized its much more simpler than we expected. As to note our group member Ridwan collectively collaborated with Vincent who started late but contributed to coming up this the implementation together as a pair for this exercise and debugging. We went from the our understanding from experiment 3 and the step up and followed step by step from the PDF of the lab. Push button was initialized and it was dereferenced in the IRQ to be enabled when to pressed in CPU1 to generate the array (out issue was we had NULL in the parameters which stopped us from entering the while loop inside mutex since push button was not updated), then maximum sub as followed by document. In the hardware mutex when we went to flag from the shared buffer to go to or in the first case send the array to CPU2, like the experiment we use a while loop we lock the mutex then set the have the shared buffer flag wait for an arbitrary flag such as CPU2 wait flag, where from CPU2 in the while loop it will wait for the CPU2 wait flag till it detects it and perform the execution which for cpu2 at first was to received the array send in twos to the cpu by cpu2. We follow the same lock and unlock processor so overwritten on message cannot occur, as only one CPU can excess the resources according to the mutex. For our implementation, we divided the workload where ultimately 75% is done by CPU2 AS 25% is done by CPU1 as we divide the array size in half and start indexing form the middle of the array to the end to find the longest sequence. From countless debugging and synchronization issues, when we finally reached a match in subsequence from both CPU, we calculated our speed up time (as the CPU1 search alone cc divided by concurrent run as said) to be 4.01 approximately.