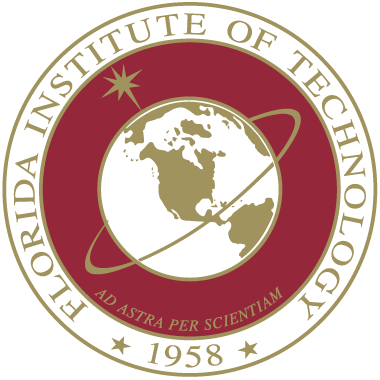
**Final Project Report**

**Effects of Noise on a Communication Channel:**

**Encoding & Decoding Buried Intelligence and LabVIEW Animation**



ECE 4342 – 1

Virtual Instrumentation Laboratory

Student Name: Xiaodong Zheng

Student ID: 902733276

Instructor: Dr. Syed Hussain Murshid

GSA: Gregory Lovell

Report Submitted: April 25, 2017

**Effects of Noise on a Communication Channel**

**Requirements:**

Realize the VI that can divide a 200 by 200-pixel picture into three color channels (Red, Green, and Blue), and add at least four different types of noises to the picture. Then, use a sub VI that includes corresponding filters to filter the noise added picture.

**Purpose:**

This subpart requires we add noises and filters to a 200 by 200-pixel picture, which can be done through image processing. The lab encourages us to extract three color channels’ data and recombines them to three pictures. Then, add four type noises to these pictures. Student should know corresponding knowledge of image processing and accomplish the VI.

**Software used:**

* LabVIEW 2014
* LabVIEW 2016

**Procedure:**

**Front Panel Design:**

A front panel was designed to show three different color channels of the original picture. The front panel includes a two-page tab control. The first page shows three different color channels and the original picture. A path control was applied to choose a picture, and a Boolean switch was used to choose the type of the selected picture. Two readable picture types are JEPG and PNG. The second page shows three channels after noises and filters added. Also, three graphs were added in to show the original single color picture and the differences of the pictures that before noises added in.

**Step by step procedures:**

1. Created a tab control in the front panel.
2. In the first page of the Tab control, created a path control, three intensity graphs, and named intensity graphs as Red, Green, and Blue, respectively.
3. Right clicked on the color panel of the Red intensity graph, chose maker color, set the Red channel value as 255, Blue and Green as 0.
4. Repeated last step for the Green intensity graph and Blue intensity graph.
5. Placed a Boolean switch on the first page of the Tab control.
6. In the second page, copy the intensity graphs of the first page, place two copies in the second page, aligned them into three columns and left some room in the gap.
7. Placed radio button controls between different columns for each intensity graph.
8. Right clicked on every radio button control, and chose “Add Radio Button,” increased the number of the radio button to five, renamed the radio buttons.
9. Placed a “Text Ring” control and an indictor in the second page respectively.
10. Right clicked on the Ring control, chose properties, in the Edit Items option, added four items, use the item name as their values. The Items are Uniform, Gaussian, Random, and No noise.
11. Created a Ring indictor to show the chosen item of the Ring control.
12. Placed six numeric indictors and renamed them as Noise difference.

Block Diagram Design:

The block diagram includes ten subVIs, to allow users choosing and corrupting with four noises for each channel, multiple case structure was applied. To calculate the total time that run once the program, the whole VI was included in a sequence structure and was divided into two parts.

Step by step procedures:

1. Created a Read JPEG File VI and a Read PNG File VI.
2. Created a case structure, put JPEG File.VI into the true case and the PNG File VI into the false case.
3. Connected the button to the case selector.
4. Created a Draw Flattened Pixmap VI.
5. Wired the image data terminal from JPEG VI and PNG VI to Draw Flattened Pixmap VI.
6. Created a subVI named Channel Spliter, the following steps describe how to create this subVI.
7. Created an Unbundle by Name.
8. Created a Decimate 1D Array.
9. Created three Reshape Array functions.
10. Created three Reserve 1D Array functions.
11. Set the Unbundle by Name to three items: image, Rectangle.right, and Rectangle.bottom.
12. Created indictors for the components.
13. Connected the components as Figure XX shown.
14. In the front panel, assign terminals for each indictor.

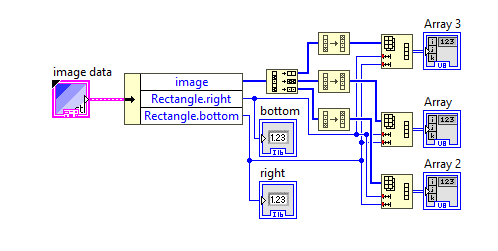


Figure 1:Channel Spliter subVI

1. Connected the output terminals: Array 1, Array 2, and Array 3 to three intensity graphs, respectively.
2. Created “Add Noise” subVI:
   1. Created a case structure, right clicked on the edge of the case structure, chose “Add Case After” to add cases.
   2. The case items should be same as radio buttons.
   3. Added four noise subVIs to corresponding case.
   4. Created a text ring control named “Choose case”.
   5. Connected the text ring control to the case selector.
   6. Created corresponding indictors in the front panel.
   7. In the front panel, assigned terminals to each indictor and controls.

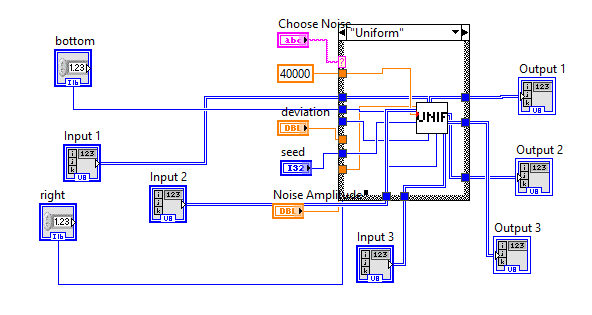


Figure 2:Add Noise subVI Block Diagram

1. Created noise subVIs:
   1. Selected a Gaussian White VI, placed a multiply function and a Reshape Array function.
   2. Multiply the Gaussian White VI with a numeric control via the multiply function.
   3. Connected the output of the multiply function to a Reshape function, connected the Rectangle.Right to the first-dimension size and Rectangle.Bottom to the second dimension of the Reshape function.
   4. Placed a To Byte Integer, connected the output of Reshape function to To Byte Integer function.
   5. Placed three multiply functions, created three inputs and outputs.
   6. Multiplied the output of To Byte Integer function with input by using the multiply functions.
   7. Created the rest three noise subVIs by using same method mentioned above, the only difference is that the noise VI should be changed correspondingly.
2. Created a case structure, added four cases: Red channel, Blue channel, All Channels, and Green channel.
3. Placed the “Add Noise” subVI into a case structure.
4. In different case, connected different output terminals of the “Add Noise” subVI to outside of the case structure, for example, if the case was “Blue Channel”, then connected only the terminal “input3” of the “Add Noise” VI to Blue channel. For Red and Green channel, just connected them to the output side of the case structure, so that in each case only one channel could be added noise.
5. Created Filter subVI:
   1. Placed a case structure and add five cases.
   2. Created a Radio Button control and add six radio buttons.
   3. Connected the Radio button control to the case selector of the case structure in the block diagram.
   4. Add corresponding cases for the case structure.
   5. Placed different filter VIs into the case structure and renamed the case structure items.
   6. Placed two Reshape Array functions and one constant with the value 40000.
   7. Expanded the second Reshape Array function and connected them as the diagram XX shown.

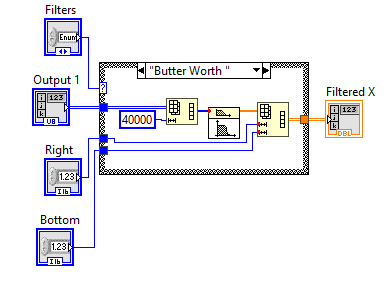


Figure 3 Add Filter subVI

1. Created “Recombined&noise” subVI
   1. Created three Reshape Array, three Reserve Array, and a interleave 1D Array.
   2. Created a Bundle by Name and an Unbundle by Name.
   3. Set the parameters of Bundle by Name and Unbundle Name as shown below.
   4. Created a Picture to Pixmap and a Drawn Flattened Pixmap.
   5. Connected the components as shown below.

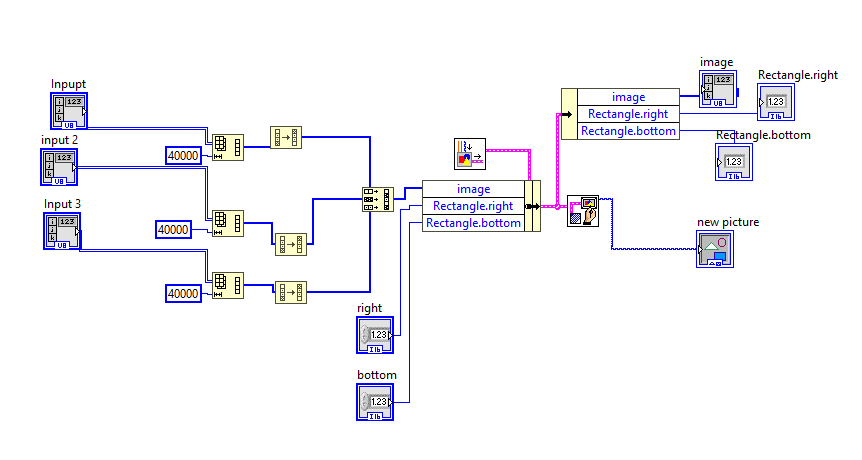


Figure 4: Recombined & noise subVI Block Diagram

1. Created Noise Difference subVI.
   1. Created two index Array functions.
   2. Expanded the number of inputs terminal Index Array to two.
   3. Added two zero constants to the index(row) and index(column) of the Index Array functions.
   4. Connected the rest components as shown below.

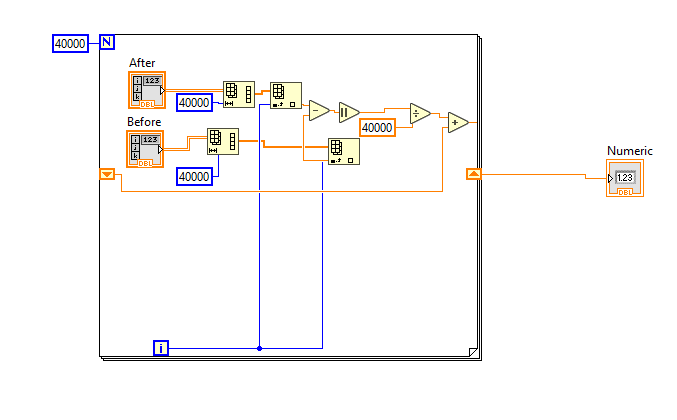


Figure 5:Noise Difference subVI Block Diagram

1. Created record noise difference subVI.
   1. Created Concatenate Strings function.
   2. Created string constants and set appropriate format.
   3. Created Open/Create/Replace File function, and any other functions that shown in below figures.
   4. Connected all components as shown below.

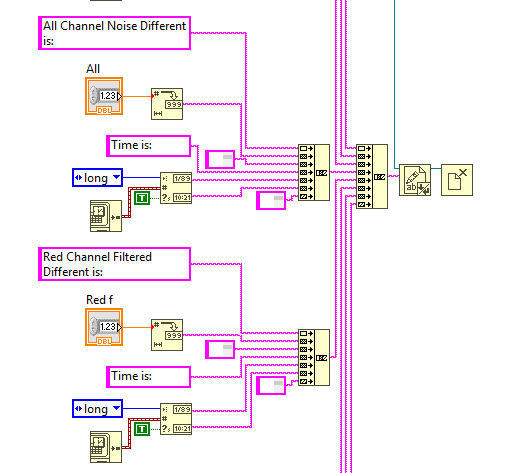


Figure 6: Record Data subVI part 1

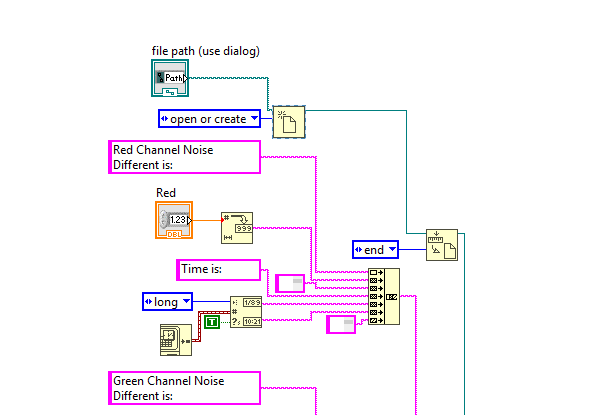


Figure 7: Record Data subVI part 2

1. Created sequence structure.
2. Right clicked on the edge of the sequence structure chose“Add Frame After”.
3. Added a “Tick Count” function in the first frame, added another one in the second frame..
4. Placed a minus function, connected the second Tick Count to the first input and the first Tick Count to the second input of the minus function.
5. Created a “Format into String” function in the second frame.
6. Created a “One Button Dialog” function in the second frame, created two string constant with the values as “last run time is %d ms” and “Got it”.
7. Connected all components as shown below.

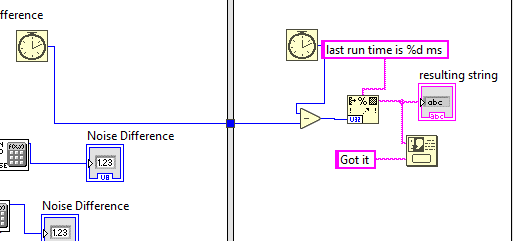


Figure 8:Time calculating and Dialog Part

**Results**

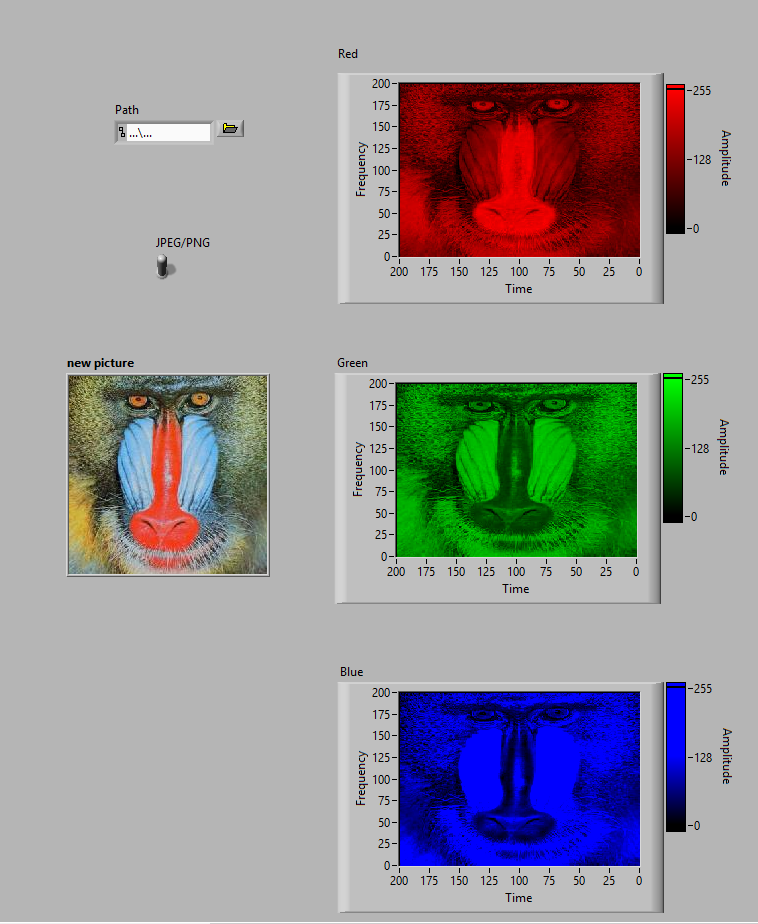


Figure 9: Image Divided Into Three Different Color Channel

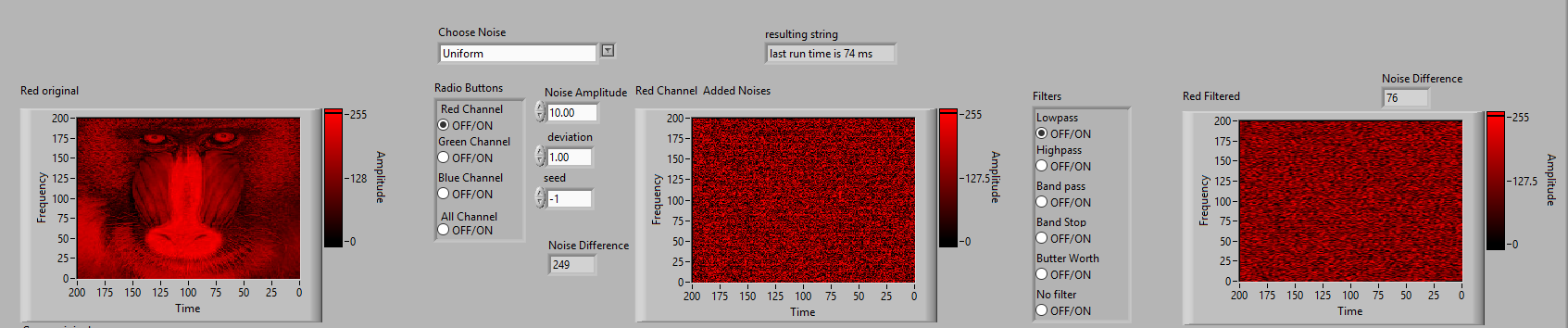


Figure 10:Red Channel-Uniform noise-Low Pass Filter

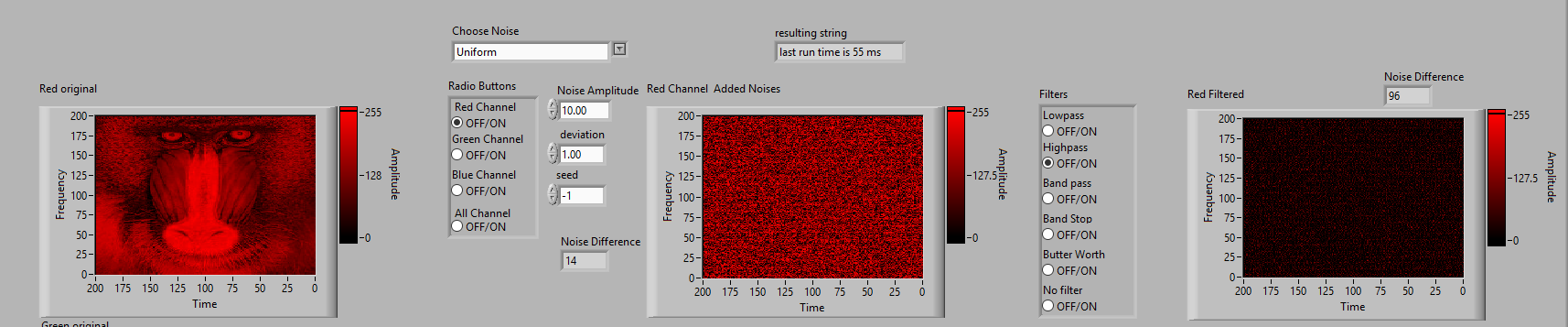


Figure 11: Red Channel-Uniform noise-Low Pass Filter

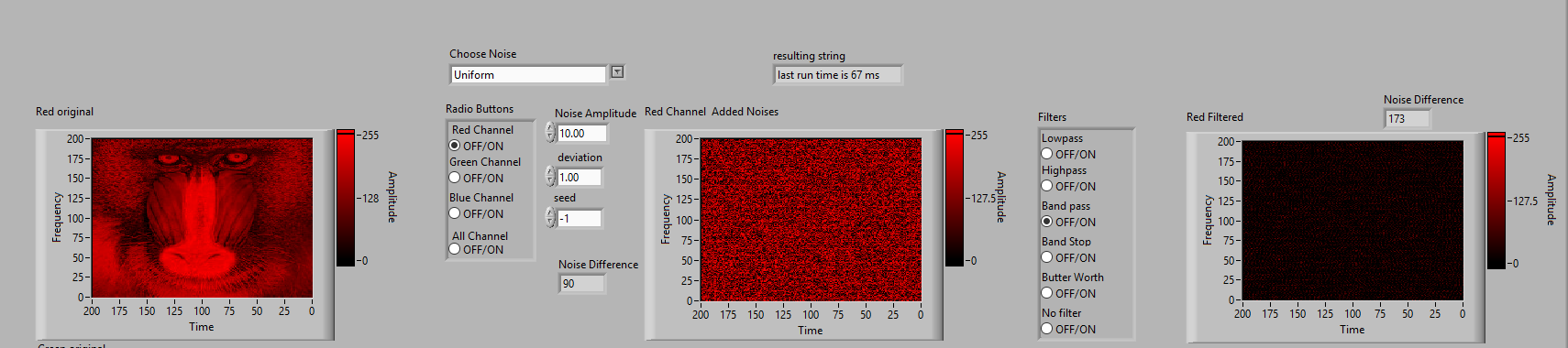


Figure 12: Red Channel-Uniform noise-Band Pass Filter

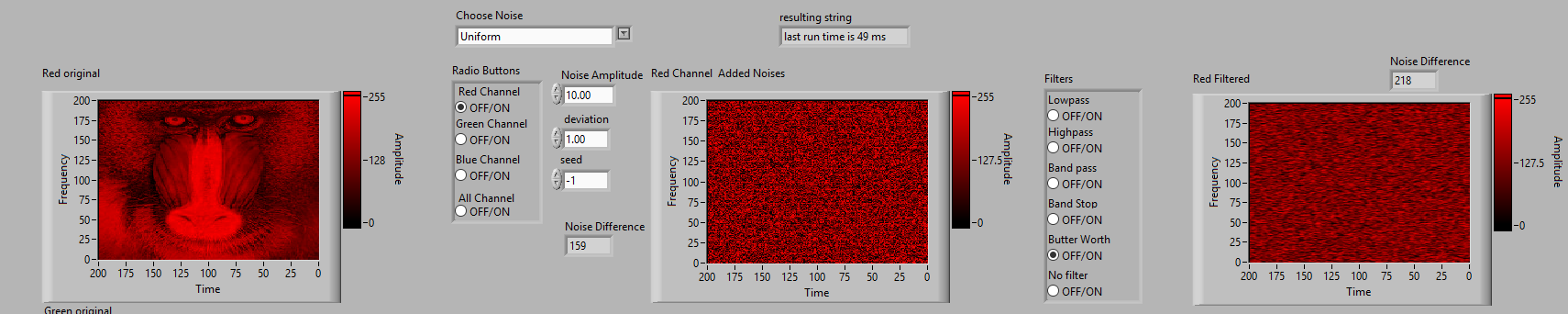


Figure 13: Red Channel-Uniform noise-Band Pass Filter

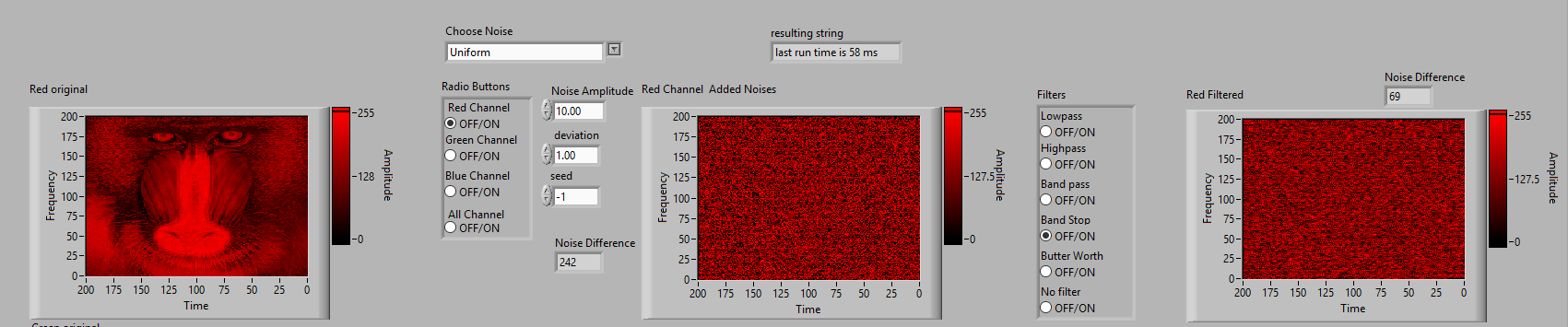


Figure 14: Red Channel-Uniform noise-Band Stop Filter

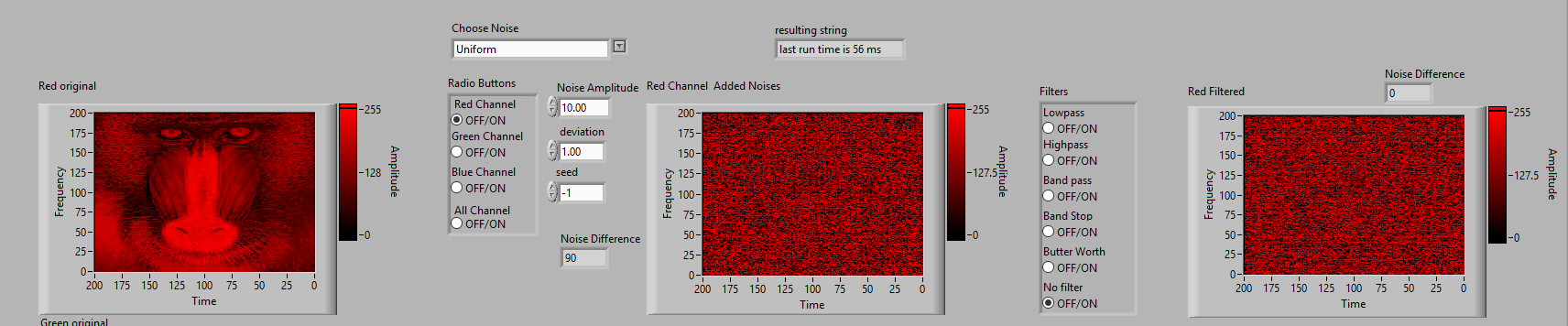


Figure 15: Red Channel-Uniform noise-No Filter

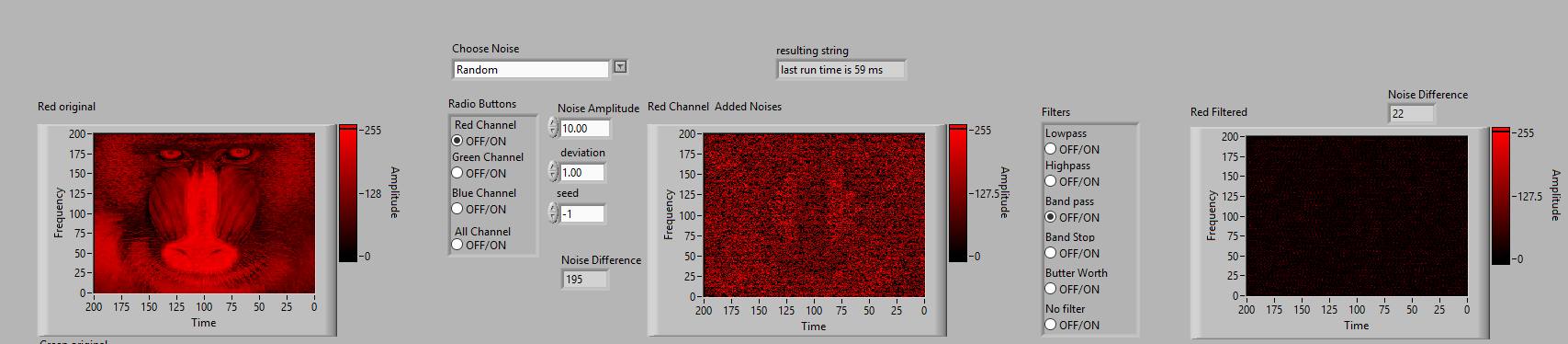


Figure 16: Red Channel-Periodic Random noise-Band Pass Filter

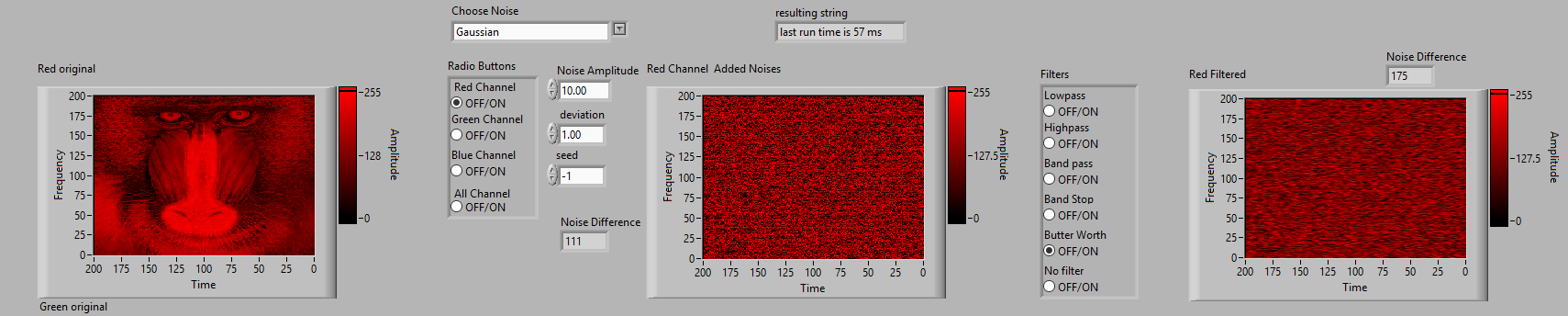


Figure 17: Red Channel-Gaussian noise-Butter Worth Filter

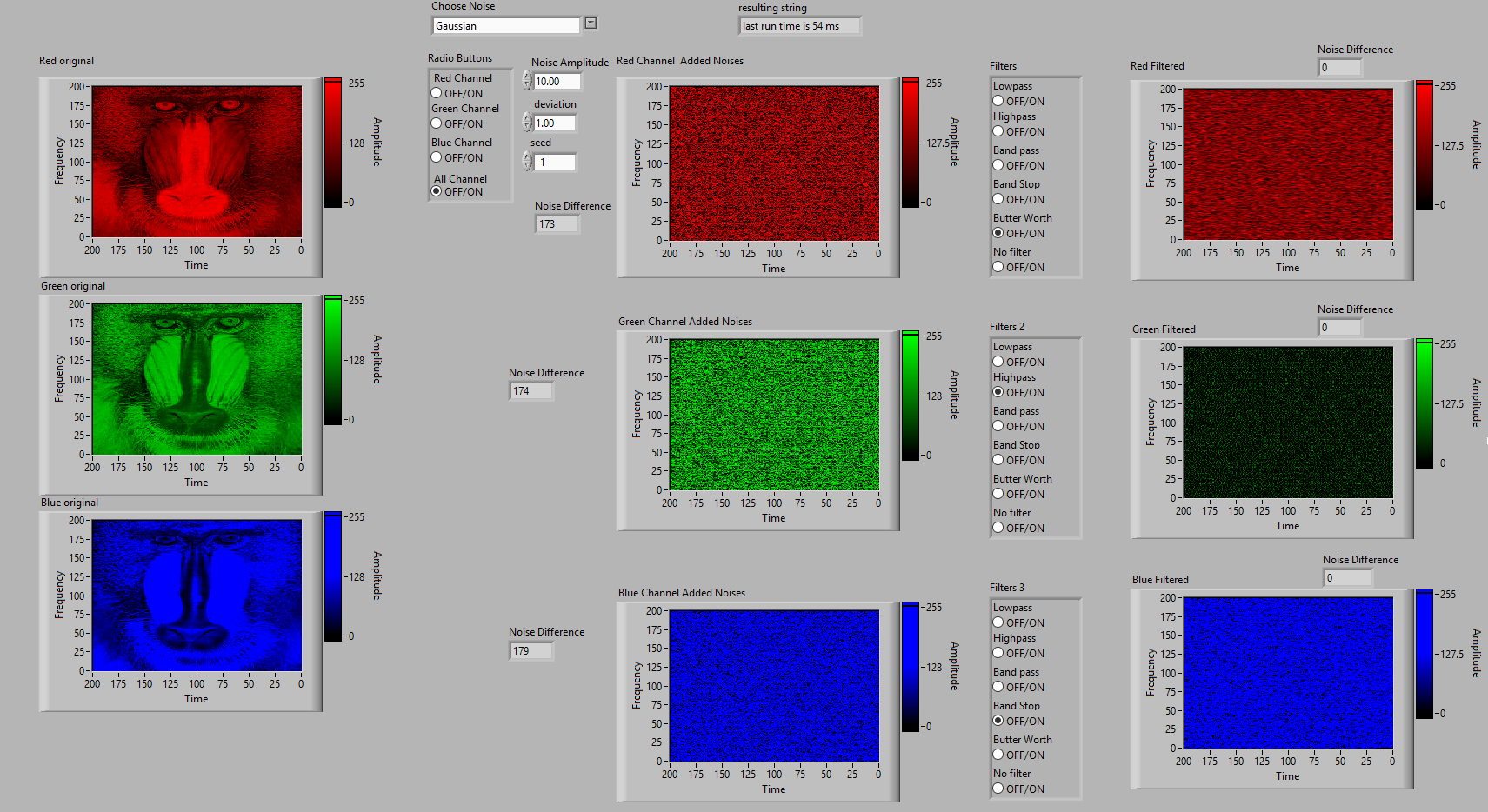


Figure 18:All Channel Gaussian Noise with Different Filter applied



Figure 19:Time to Process

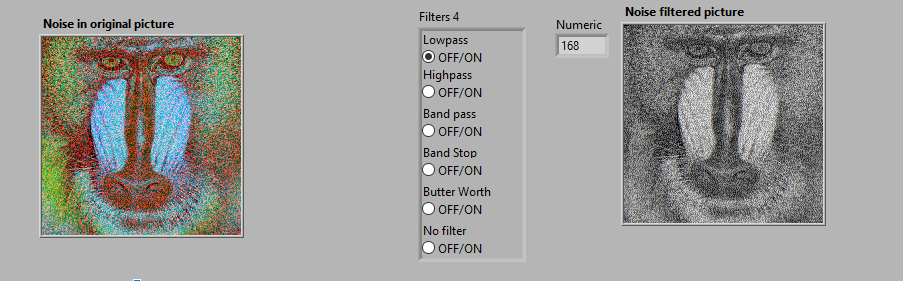


Figure 20:Uniform Noise with Lowpass Filter

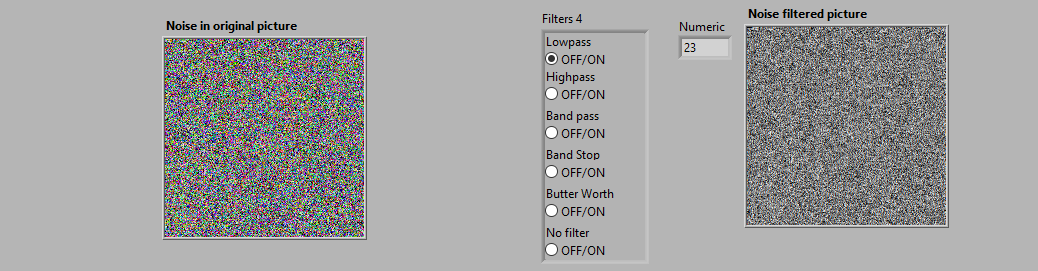


Figure 21:All Channel Added Periodic Random Noise Shown in One Picture and Lowpass Filter Applied

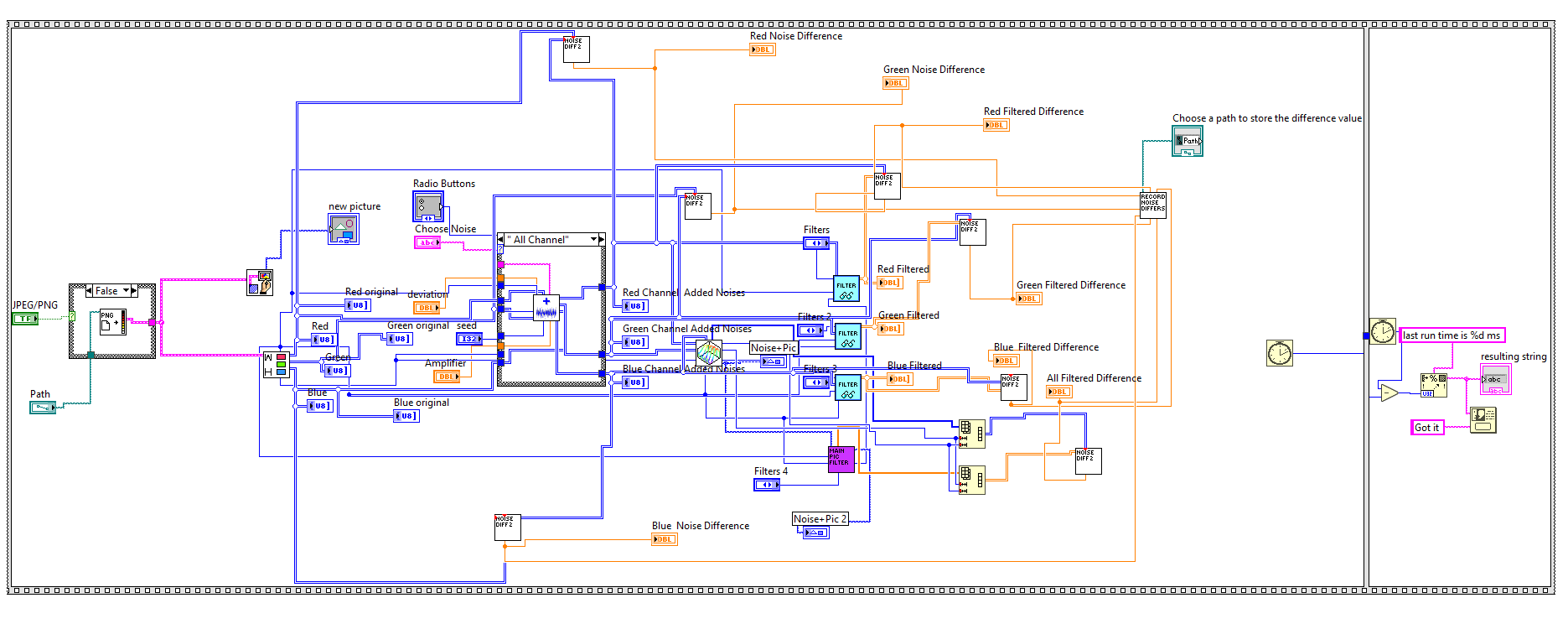


Figure 22:Complete VI Block Diagram

**Calculation:**

**Noise Difference:**

N: total pixels number

PN: noisy pixel

PO: original pixel

**Discussion:**

In this part, I did many subVIs and cost a lot of time to figure out the structures. The image is 200 by 200 pixels, which means the total number of pixels is 40000. To divide the image into three channel, I first used Unbundle by Name to transform the image data to array data, and also get the picture width and height by using Rectangle.right and Rechangle .bottom items. Then, I used a Decimate 1D Array function to process the image array. Because the image data is made up of pixels, and every pixel includes three 8 bit number. To divide the image into three channel, we need decimate the data into three independent 1D array, and then use reshape function to recover the picture. To adjust the direction of the picture, I added reverse 1D array function to each channel, and adjusted the axes direction of the intensity graph on the front panel. Also , because I connected the rectangle.right attribute to the first “dimension size” option of the Reshape Array, so I need to transpose the array. Another thing worth to mention is that the intensity graph, at the beginning, I did not know how to set the intensity graph, and I got all pictures shown in blue color. After I searched many resources and asked help from my friends Lin, finally I knew I had to set the maker color. For red channel, the red color number should be 255 while the rest two color numbers should be set as 0, and similar things should be done in green and blue channel.

The noise part is the most complex part for this VI, I did a “Add noise” VI to involve all the noise subVIs, and put it into a case structure. Also, every single noise VI is involved in a case structure. In this kind of structure, the noise can be added in every single channel every time the program runs. The hierarchy of the “Add Noise” subVI is shown below.

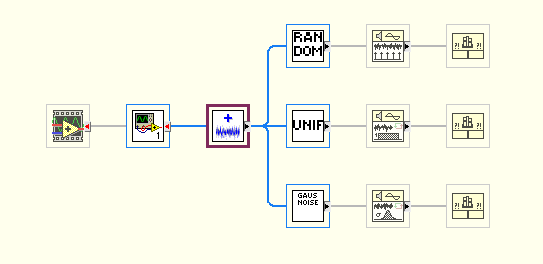


Figure 23:Hierarchy of the Add Noise subVI

The “Recombined&noise” subVI can recombine three noise-added-channels to a picture and show it on the front panel. This VI is very similar to the “Channel Spliter” subV. It uses “Bundle by Name” to recombine the data of the image, and then converts the image data to picture data.

The “Add Filter” subVI can add a specific filter to the input data. This subVI has five filters: Low-pass, High-pass, Band-pass, Band-stop, and Butter Worth filter. The Low/High/ Band pass filter allow specific frequency signal to pass. The Butterworth filter, according to its definition, “is a type of signal processing filter designed to have as flat a frequency response as possible in the passband(Wikipedia).”

For the noise difference subVI, I used the given equation of the lecture slides to calculate the noise difference. I used a shift register and a for loop to realize this equation. The results are very big because of square sign in the equation. To store the result, I did a subVI named “Record Noise Difference” the subVI can record every noise difference result into a text file, and record the time. To use this function, the file path should be assigned in the front panel.

To calculate how much time it takes for this VI to run, I used a sequence structure and two tick count functions. The first tick count function will pass the start time in milliseconds to second sequence structure, and the second tick count function will record the end time. The difference of two values is the time costed for this VI to run.

**Encoding & Decoding Buried Intelligence**

**Requirements:**

1. Read in user Text file for buried intelligence
2. Encode intelligence using the least significant bit of any channel (RGB) of the image
3. Decode & display text as Scrolling Marquee on front panel
4. Build an executable (.EXE) of this VI.

**Purpose:**

This part requires student to make a VI that can encode a paragraph words into a picture, and depress the words from the picture. The image processing technics will be applied to this VI, and the VI should show the words by using a scrolling marquee.

**Software used:**

* LabVIEW 2014
* LabVIEW 2016

**Procedure:**

**Front Panel Design:**

in this section, the front panel of the VI mostly depend on the block diagram, for the subVIs .

1. Created a Tab control in the front panel.
2. Created four File Path control in the front panel.
3. Created three string indictors.
4. Created a slider control in the front panel.
5. Configured the components as the picture shown below.



Figure 24: Front Panel of Buried Intelligence

**Block Diagram Design:**

1. “ReadPic&OutputPicData”subVI
   1. Created a case structure.
   2. Created all the components shown in the diagram below.
   3. Put the Read PNG File into False case, Read JPEG File into true case.
   4. Connected all other components as shown in the block diagram below.

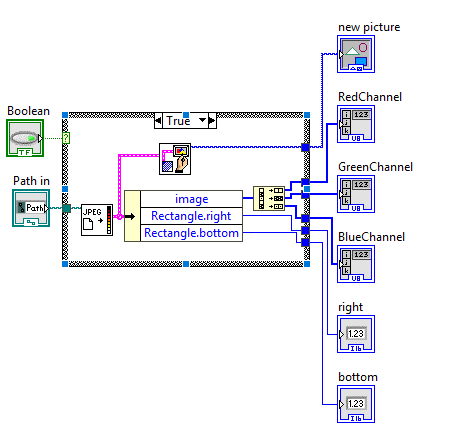
:

Figure 25:“ReadPic&OutputPicData”subVI block diagram

1. “ShowAfterBuried”subVI
   1. Created all the components shown in the diagram below.
   2. Connected all other components as shown in the block diagram below.
   3. Move the “new picture” component into appropriate position in front panel.

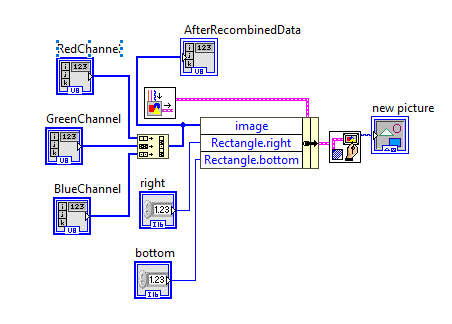


Figure 26: “ShowAfterBuried”subVI

1. “Insert Text to Pic” subVI
   1. Created a Read from Text File function.
   2. Created the rest components.
   3. Connected all components as shown in the below diagram.

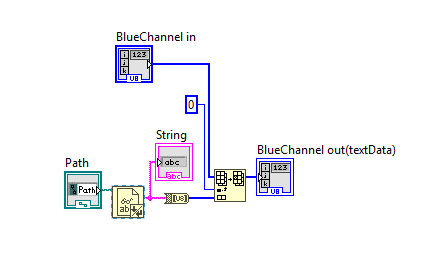


Figure 27: Insert Text to Pic subVI

1. “Decode Words from Picture” subVI.
   1. Created components as shown below.
   2. Connected all components.

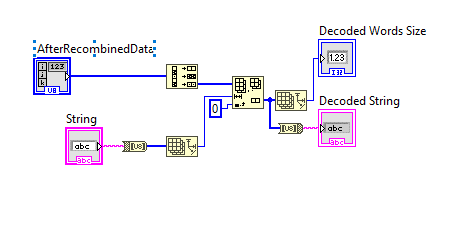


Figure 28: “Decode Words from Picture” subVI.

1. “GetPointString”subVI
   1. Selected Draw Text at Point function.
   2. Created a Unbundle by Name function.
   3. Expanded the Unbundle by Name to four items.
   4. Created two Unbundle functions.
   5. Created a invoke node and a property node.
   6. Right clicked on the invoke node, chose link to option, chose “new picture”.
   7. Clicked on the property node, chose DrawAreaSize option.
   8. Connected all other components as shown below.

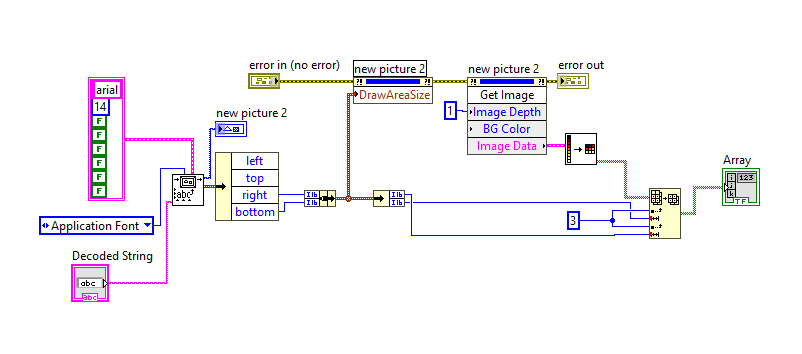


Figure 29: “GetPointString” subVI

1. “Save to file”subVI
   1. Created all the components shown in the diagram below.
   2. Connected all other components as shown in the block diagram below.

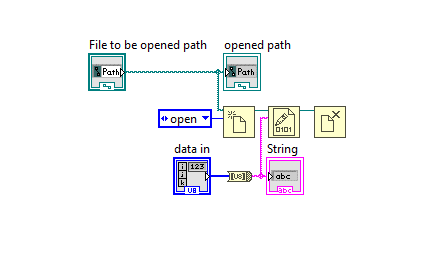


Figure 30: “Save to file” subVI

1. Created a while loop, inside the while loop, created a case structure.
2. Created all the components shown in the below diagram, connected all components.

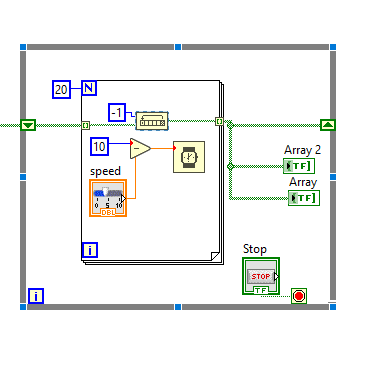


Figure 31: Show String Part

1. Whole VI block diagram

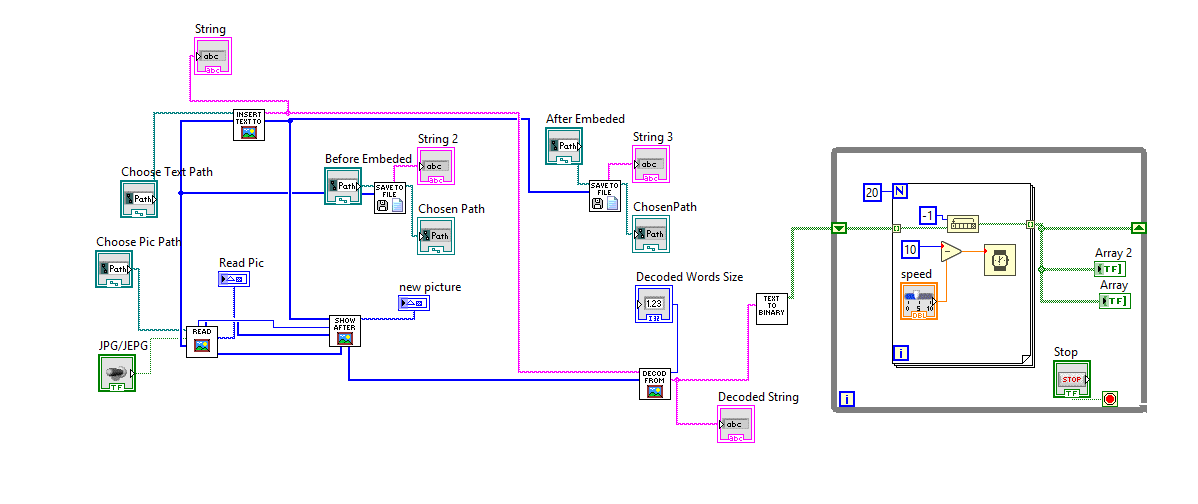


Figure 32:Whole VI block diagram

**Results:**

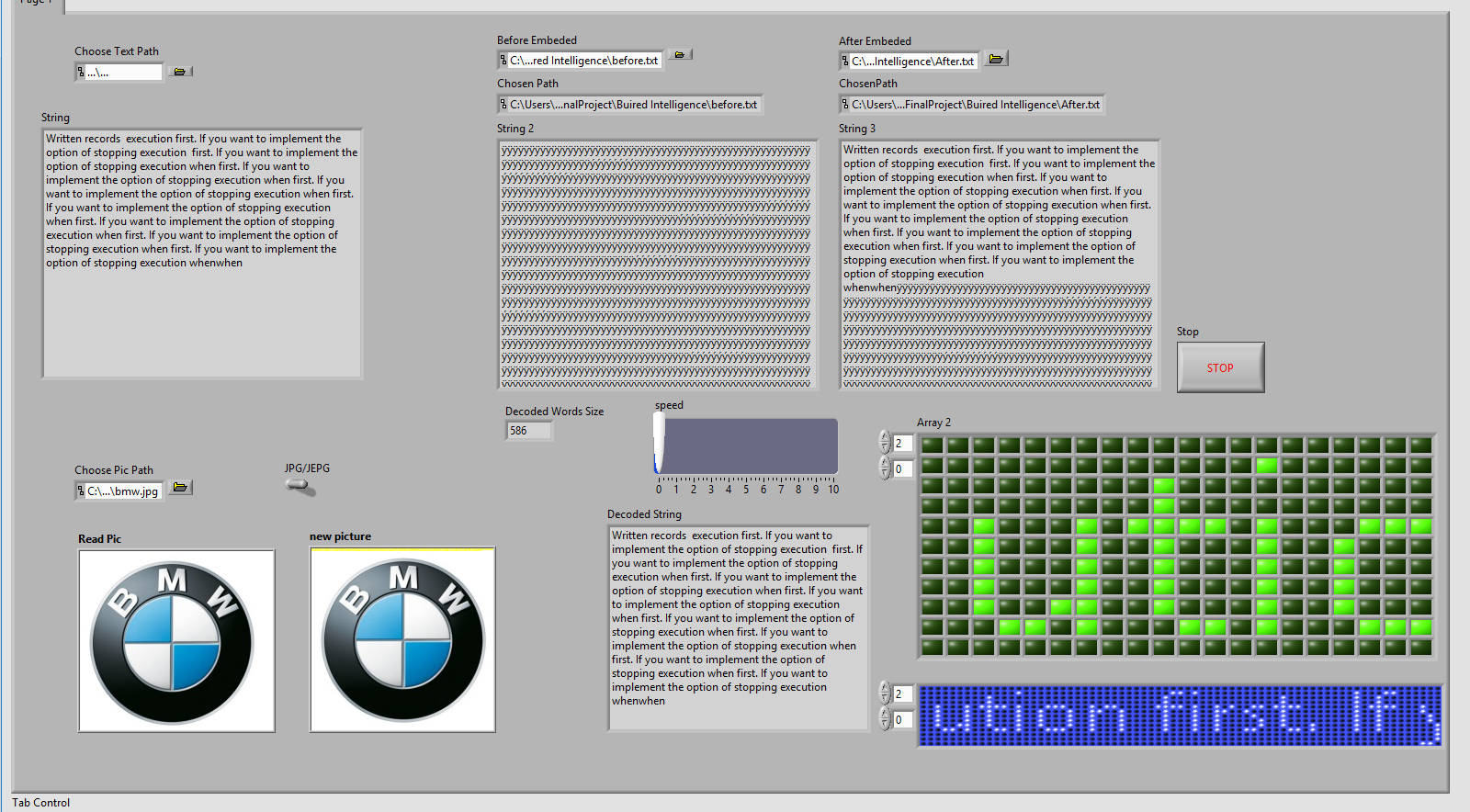
****

Figure 33: Buried Intelligence VI

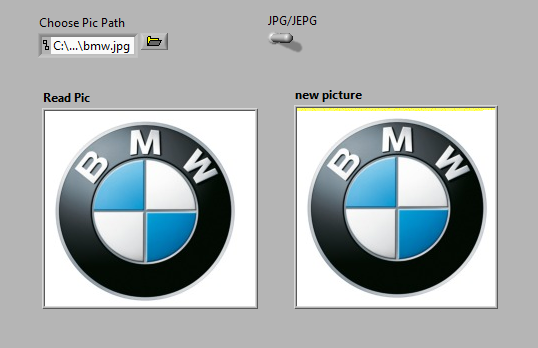


Figure 34:Choose Picture Part and Encoded Image

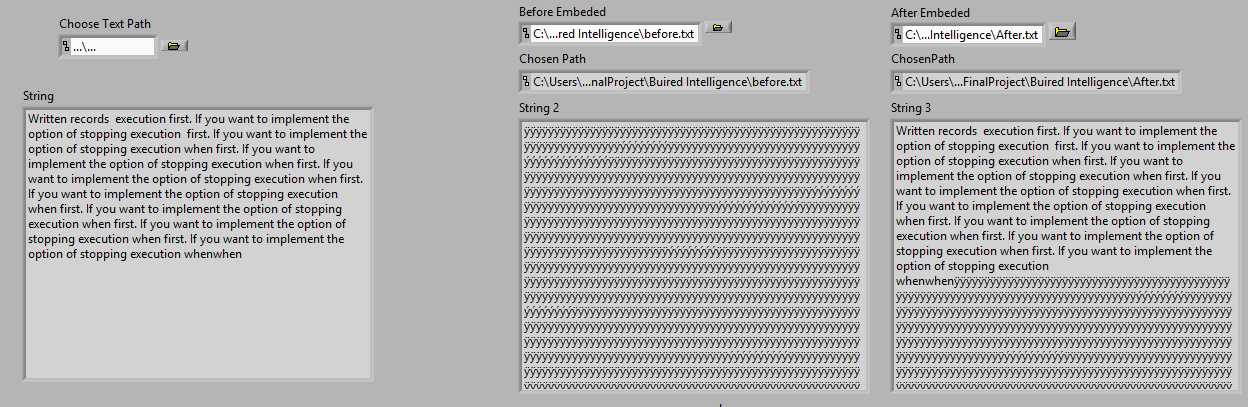


Figure 35:Choose Text and Processed String

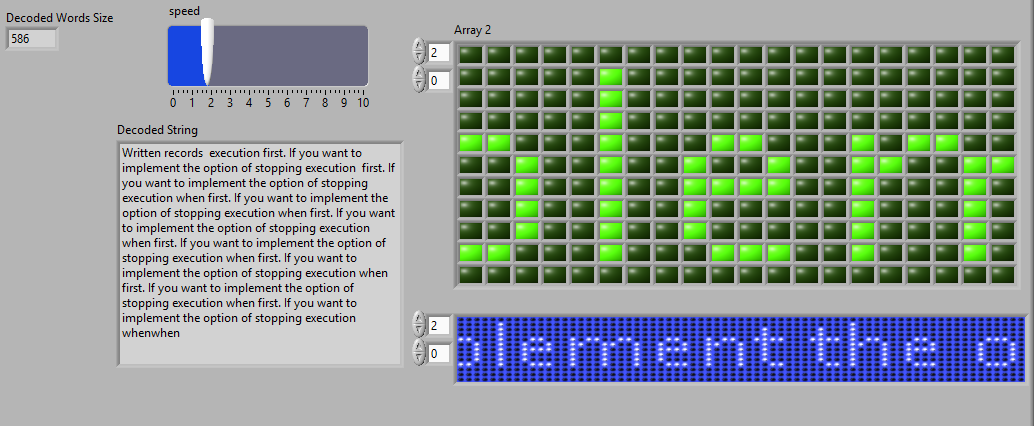


Figure 36:Decoded Text and Scrolling Marquee

**Discussion:**

In this VI, I used a lot of subVIs to make the structure clearer. The Read subVI is used to read the picture from system, user can choose JPG or JEPG picture to use. To use the least significant bit of any channel, I used similar structure as the RBG VI, which is by using Unbundle by Name and Decimate 1D array to divide the image data into three channels, and I used the blue channel data to bury the text data. To bury the text data into the picture, I just simply used replace array function replaced the data of blue channel with text data. The text string data can be changed to byte data via String to Byte array.

The most hard part of this VI is how to show the text in the form of a scrolling marquee. Honestly, I checked many resources by using internet, and learned many useful components and skills to handle this problem.

I used the “Draw Text at Point” to make the decoded string become a picture, then, I used the property node to control the picture size appropriate, and used the invoke node to collect the information of the new created picture. After that, an unflatten Pixmap was applied to converts the image data cluster into a 2D 1-bit array, which means the array only has the elements of true and false. To make the string scrolls like a scrolling marquee, I created a for loop inside a while loop. Inside the For loop, a rotate 1D array function was applied to move the bit. By using this kind of structure, the string can be moving like a scrolling marquee.

**LABVIEW ANIMATION**

**Requirements:**

1. Create Animation on front panel using a series of at least 10 picture frames. User should be able to control the speed of the animation.
2. Include Play, Pause, and Stop buttons.
3. Embed each frame of the animation inside the VI.
4. Make the VI password protected. Have the user enter correct password before allowing VI to run. If wrong password is entered it should display a warning message.
5. Set up Animation VI to allow user to remotely control said VI from a separate computer.
6. This can be done over the internet or over the network.

**Purpose:**

This VI requires students realize an animation player by using LabVIEW. The animation actually consist of serval pictures, when the playing speed exceeds 16 pictures /second, human eyes will produce illusion so that the animation effect produced.

**Procedure:**

**Front Panel Design:**

The front panel of this VI is very simple; the picture shown below shows the structure and layout of the front panel.

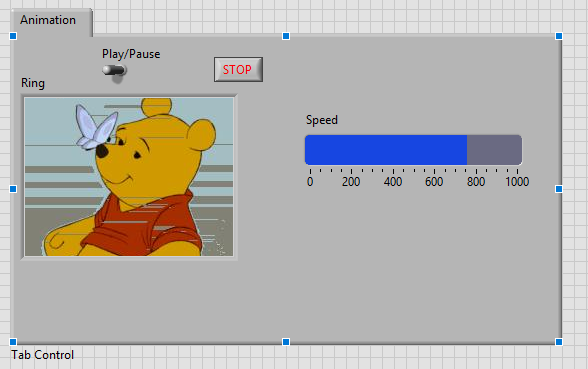


Figure 37:Animation VI Front Panel

The Picture Ring is used to show the animation pictures, and the slider is applied to control the play speed. A button called Play/Pause is used to play the animation or pause the animation.

**Block diagram:**

The block diagram is basically divided into two parts; the first part is used to check the password while the second part is used to play the animation.

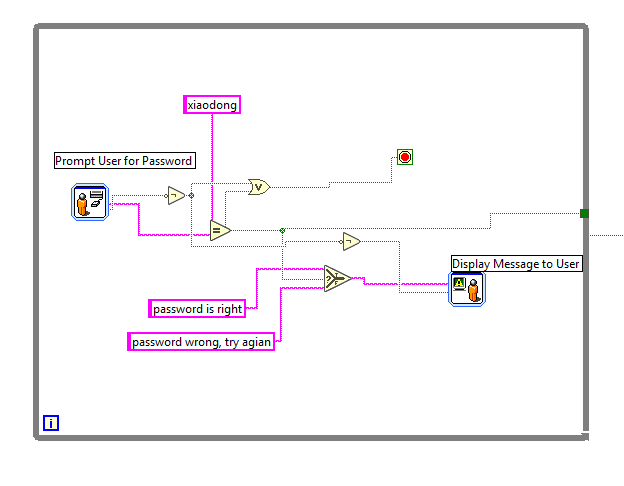


Figure 38: Password Process loop

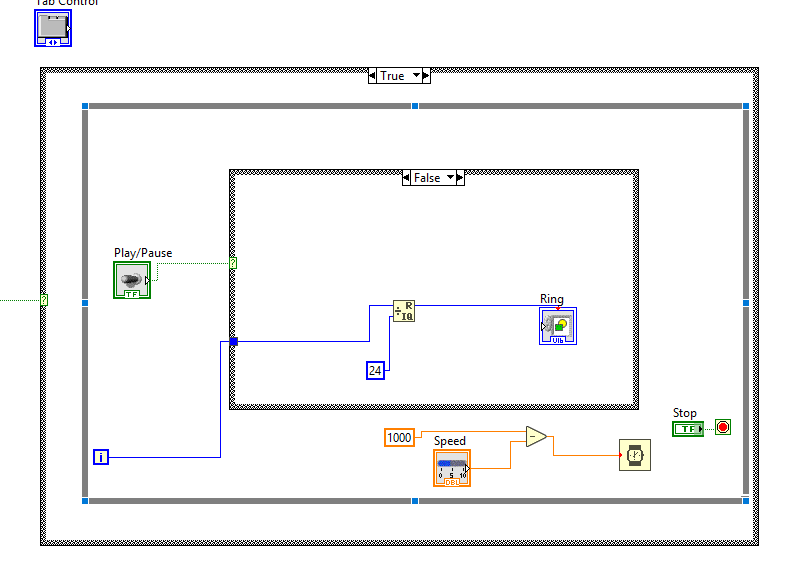


Figure 39: Animation Playing Part

**Results:**

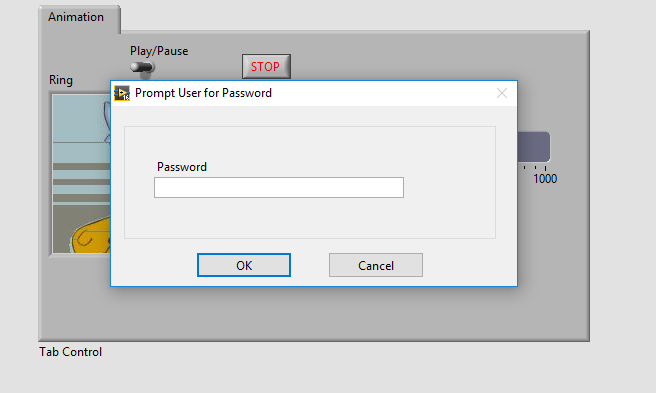


Figure 40: VI Password Protected

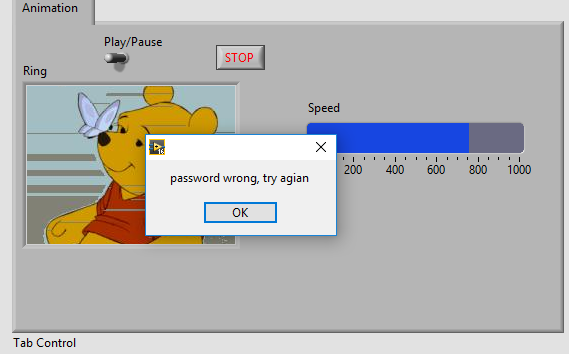


Figure 41:Wrong Password

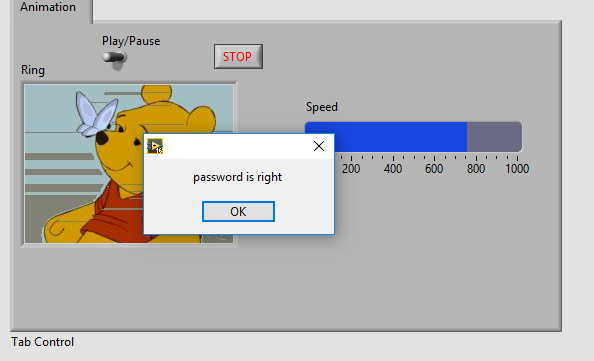


Figure 42:Right Password

**Discussion:**

This VI comparing with the other two Vis is really easy, the password could be realized by using Prompt User for Password. The hardest part of this VI is how to pause the animation while it is running. A case sturecture is applied to realize this function; in the false case, a Quotient & Remainder used to control the number of picutres to be playing. Outside the case structure, a while loop is applied to play the picutres in a very fast speed so that they look like animation. The second case structure is in the outside of the while loop. The selector of the second case structure is connected to the first part so that the program will be running only in the true case. To control the playing speed, I placed a slider into the while loop and connected it to a wait function.

**Conclusion:**

In this project, I learned many useful knowledge of the image processing. This project is pretty hard for me, I spent a lot of time on it. The RGB VI is the most hard one, I built up more than ten subVIs to make the structure more clear, however, I noticed too many wires making the VI looks very messy. I used the given formular to calculate the noise analyzation. Moreover, I found that some function of the LabVIEW sometimes will cause unconvenience for users. For example, if you changed a control’ representation of a subVI in a main VI, the labVIEW will autochange all the representations of the components that connected to the control.

Through this course, I learned how to use LabVIEW software. However, the most important thing is that I have learned how to think inpendently and linearly. The labview helped me get more understanding to programming, and the structure of softwares, which is the biggest fortune I have gotten.