# Digital Communication using GNU Radio

**Prof. Kumar Appaiah**

**Department of Electrical Engineering**

**Indian Institute of Technology Bombay**

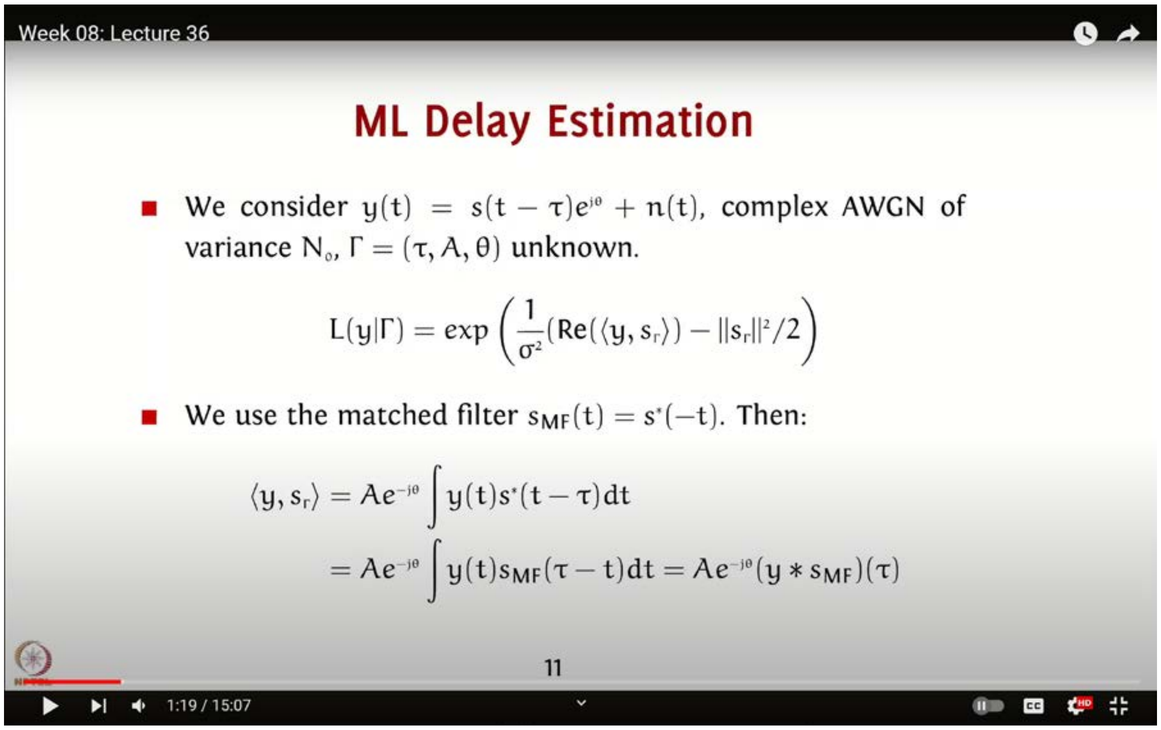
**Week-08**

**Lecture-36**

**Maximum Likelihood Delay Estimate for a Single Symbol in GNU Radio**

Welcome to this lecture on Digital Communication using GNU Radio. My name is Kumar Appiah from the Department of Electrical Engineering at IIT Bombay. In this session, we will explore an initial example of how to estimate delay.

(Refer Slide Time: 01:19)



Week 08: Lecture 36

## ML Delay Estimation

- We consider $y(t) = s(t-\tau)e^{j\theta} + n(t)$, complex AWGN of variance $N_0$, $\Gamma = (\tau, A, \theta)$ unknown.

$$L(y|\Gamma) = \exp\left(\frac{1}{\sigma^2}(\mathrm{Re}(\langle y, s_r\rangle) - \|s_r\|^2/2\right)$$

- We use the matched filter $s_{MF}(t) = s^*(-t)$. Then:

$$\begin{aligned}\langle y, s_r\rangle &= Ae^{-j\theta}\int y(t)s^*(t-\tau)dt \\ &= Ae^{-j\theta}\int y(t)s_{MF}(\tau-t)dt = Ae^{-j\theta}(y * s_{MF})(\tau)\end{aligned}$$

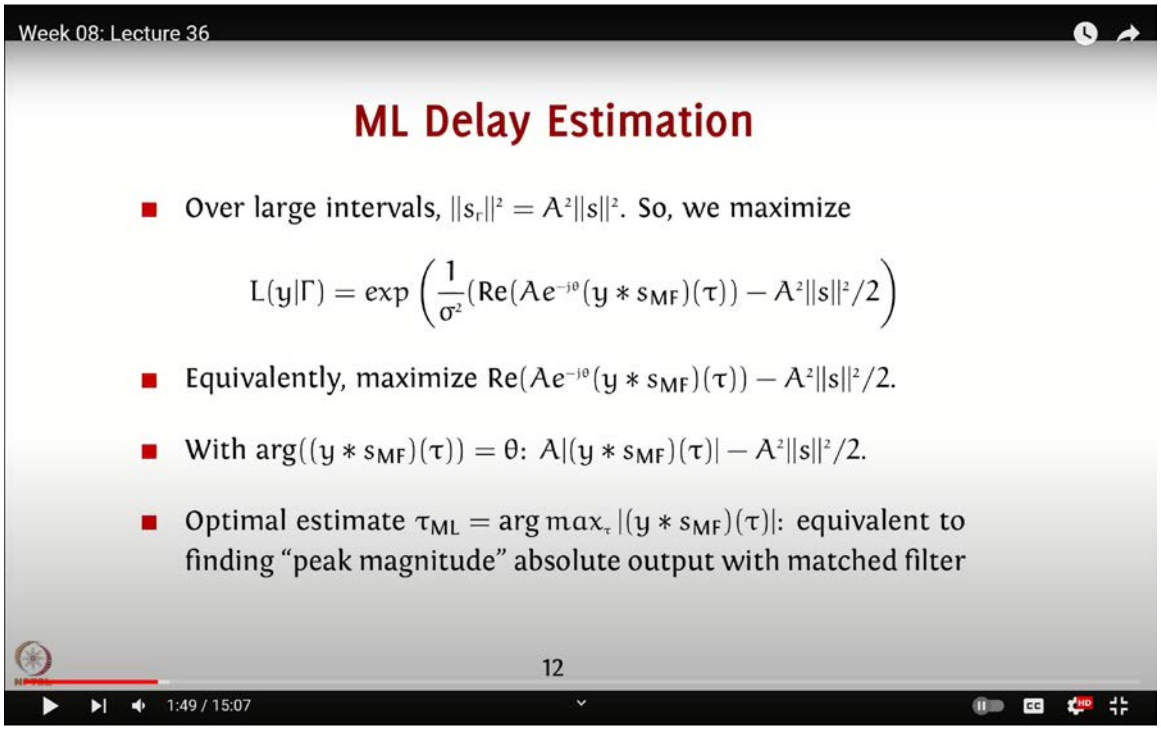
11

1:19 / 15:07

Recall from our class discussions that estimating the delay of a signal s(t) involves maximizing the absolute value of the convolution of the received signal with the matched filter. We will use a simple template function or a set of straightforward template functions to illustrate how the peak of this convolution can provide an accurate estimate of the delay.

To validate this approach, we will perform tests with random delays and ensure that our estimates consistently align with the maximum likelihood estimation approach we have studied.

(Refer Slide Time: 01:49)



Week 08: Lecture 36

## ML Delay Estimation

- Over large intervals, $||s_r||^2 = A^2||s||^2$. So, we maximize

$$L(y|\Gamma) = \exp\left(\frac{1}{\sigma^2}(\mathrm{Re}(Ae^{-j\theta}(y * s_{MF})(\tau)) - A^2||s||^2/2\right)$$

- Equivalently, maximize $\mathrm{Re}(Ae^{-j\theta}(y * s_{MF})(\tau)) - A^2||s||^2/2$.
- With $\arg((y * s_{MF})(\tau)) = \theta$: $A|(y * s_{MF})(\tau)| - A^2||s||^2/2$.
- Optimal estimate $\tau_{ML} = \arg\max_\tau |(y * s_{MF})(\tau)|$: equivalent to finding "peak magnitude" absolute output with matched filter

12

1:49 / 15:07

Our focus in this lecture will be on Delay Estimation, specifically the Maximum Likelihood Delay Estimation method. As covered in class, by overlapping s(t) with the template function and varying parameters such as τ, a, and θ, we can demonstrate that the final solution involves maximizing the magnitude of y convolved with the matched filter and finding the value of τ that maximizes this modulus. To start, we will construct a simple example and set the sampling rate to 192,000.

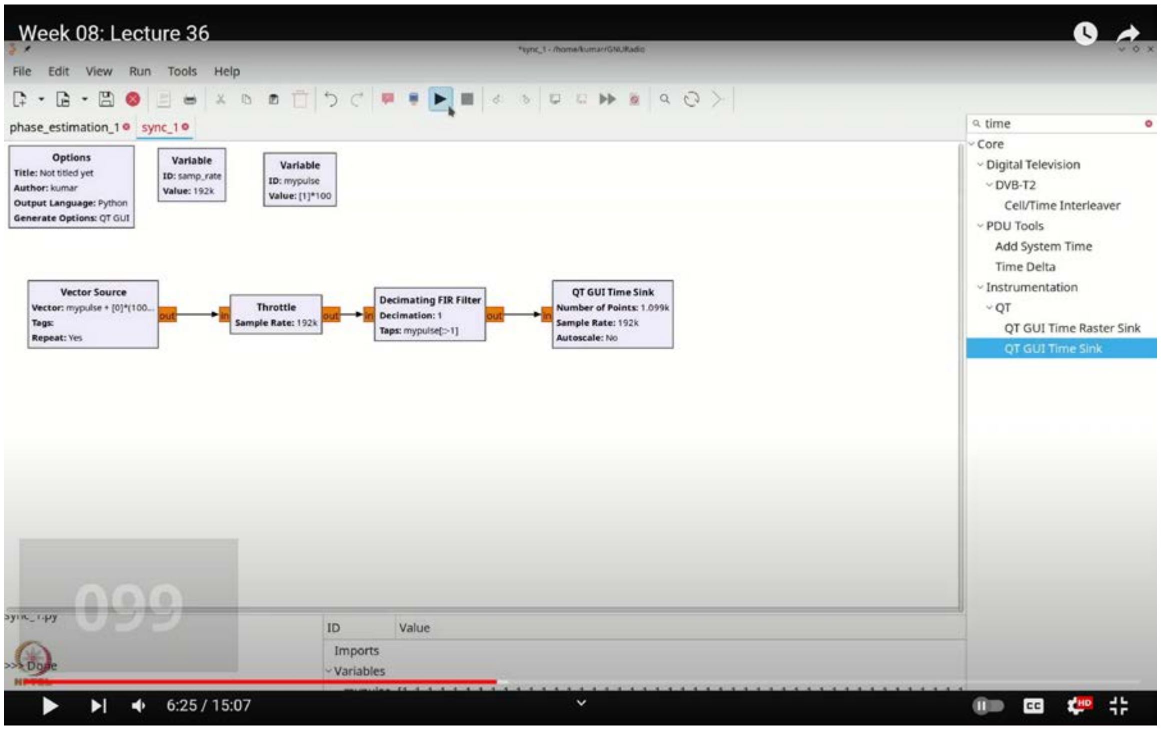
Now, let's start by creating a simple pulse and examining how delay affects it. We'll begin by setting up our flow graph. I'll first add a vector source to the graph. To do this, I'll press **`Ctrl + F`** or **`Command + F`**, type "vect", and select the vector source. Next, I'll place it on the flow graph.

We need to configure the vector source with a specific filter. For our example, we want the vector source to generate about 1,000 samples. So, we'll create a variable to hold our filter data. I'll name this variable `**my_pulse**` and set its length to 1,000. We'll initialize `**my_pulse**` with a list of 0s and ensure its length is 1,000. We achieve this by setting `**my_pulse**` to `**[0] * (1000 - len(my_pulse))**`.

The reason for this is that `**my_pulse**` will represent the pulse s(t), and we'll use it to search for delays in our example. To populate `**my_pulse**`, I'll create a new variable. Press `**Ctrl + F**` or `**Command + F**`, search for "variable", select it, and name it `**my_pulse**`. Initially, we'll set this variable to 100 ones, so the command will be `**[1] * 100**`, followed by 900 zeros. This setup is ideal because we can use it for match filtering.

Finally, let's add a throttle to the flow graph. Before doing so, convert the data type of our pulse to float, as complex numbers are unnecessary for this step. To add the throttle, press `**Ctrl + F**`, search for "throttle", and add it to the graph.

(Refer Slide Time: 06:25)



Options
Title: Not titled yet
Author: kumar
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 192k

Variable
ID: mypulse
Value: [1]*100

Vector Source
Vector: mypulse + [0]*(100...
Tags:
Repeat: Yes

Throttle
Sample Rate: 192k

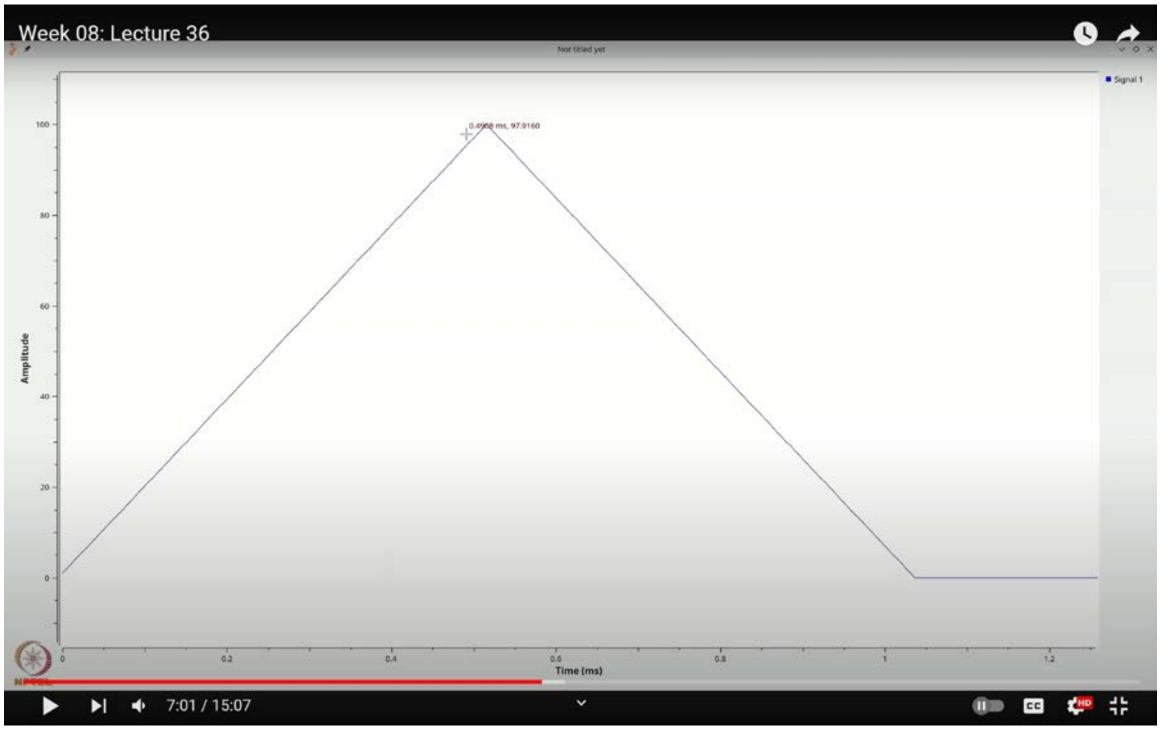
Decimating FIR Filter
Decimation: 1
Taps: mypulse[::-1]

QT GUI Time Sink
Number of Points: 1.099k
Sample Rate: 192k
Autoscale: No

We start by adding the throttle to our flow graph, ensuring it's set to float. This throttle controls the data rate of our signal. The pulse we are working with is the template pulse that will be convolved with the matched filter at the receiver. To create the matched filter, we simply reverse the pulse. Even though the pulse is symmetric, it’s good practice to reverse it explicitly. To do this, press **`Ctrl + F`** or **`Command + F`**, search for "decimating FIR filter," select it, and set it to float with real taps. Connect this filter and configure the taps to use **`my_pulse`**, applying a reversal with **`[::-1]`** to ensure it's reversed, despite its symmetry.

Next, add a QT GUI Time Sink to visualize the results. Press **`Ctrl + F`** or **`Command + F`**, search for "time," and select the QT GUI Time Sink. Double-click it to make it float, and then connect it to your flow graph. We need to set the number of points for this time sink. Since we have a 1,000-sample pulse and a 100-sample signal, the ideal length for the time sink should be around 1,100 points. Although 1,099 points would be precise, setting it to 1,100 provides a cleaner, static display.

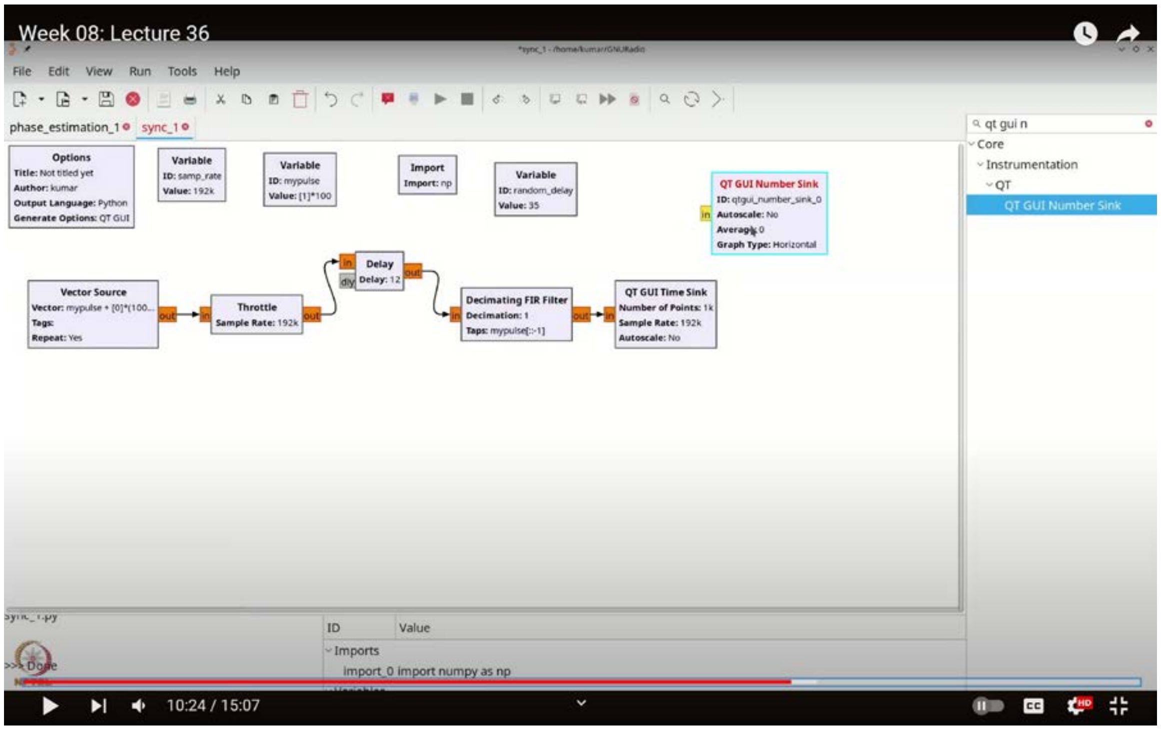
(Refer Slide Time: 07:01)



Run the flow graph. If the display seems to shift, it might be due to an incorrect length setting. Given a 1,000-sample pulse followed by 100 samples, 1,000 points should be accurate. Adjusting to 1,000 should resolve any shifting issues. When you execute the flow graph, you should observe a clear shape, with the peak ideally at 0.514 milliseconds.

To verify, let’s perform a quick calculation. For 0.514 milliseconds, using a sample rate of 192,000 samples per second, we calculate 0.514 ms × 192,000 samples/second, which equals approximately 99 samples. This corresponds to the 100th sample, consistent with the length of the filter being 100.

(Refer Slide Time: 10:24)



Options
Title: Not titled yet
Author: kumar
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 192k

Variable
ID: mypulse
Value: [1]*100

Import
Import: np

Variable
ID: random_delay
Value: 35

QT GUI Number Sink
ID: qtgui_number_sink_0
Autoscale: No
Average: 0
Graph Type: Horizontal

Vector Source
Vector: mypulse + [0]*(100...
Tags:
Repeat: Yes

Throttle
Sample Rate: 192k

Delay
Delay: 12

Decimating FIR Filter
Decimation: 1
Taps: mypulse[::-1]

QT GUI Time Sink
Number of Points: 1k
Sample Rate: 192k
Autoscale: No

| ID | Value |
|---|---|
| Imports | |
| import_0 | import numpy as np |

If you introduce any additional delay, the shift in the peak’s position will indicate the amount of delay, providing an accurate measurement of the sampling point. To illustrate this, let’s add a random delay between the signal and the decimating FIR filter to observe how the peak shift reflects the delay. For now, I'll remove this element.

I’m going to press `**Ctrl + F**` (or `**Command + F**` on a Mac) and search for "delay" to add the delay element to our flow graph. We’ll configure it to be a float and name this delay variable `**random_delay**`. To generate this delay randomly, I'll press `**Ctrl + F**` (or `**Command + F**`), search for "variable," and drag the variable element into the graph.

Double-click it to rename it to **`random_delay`**.

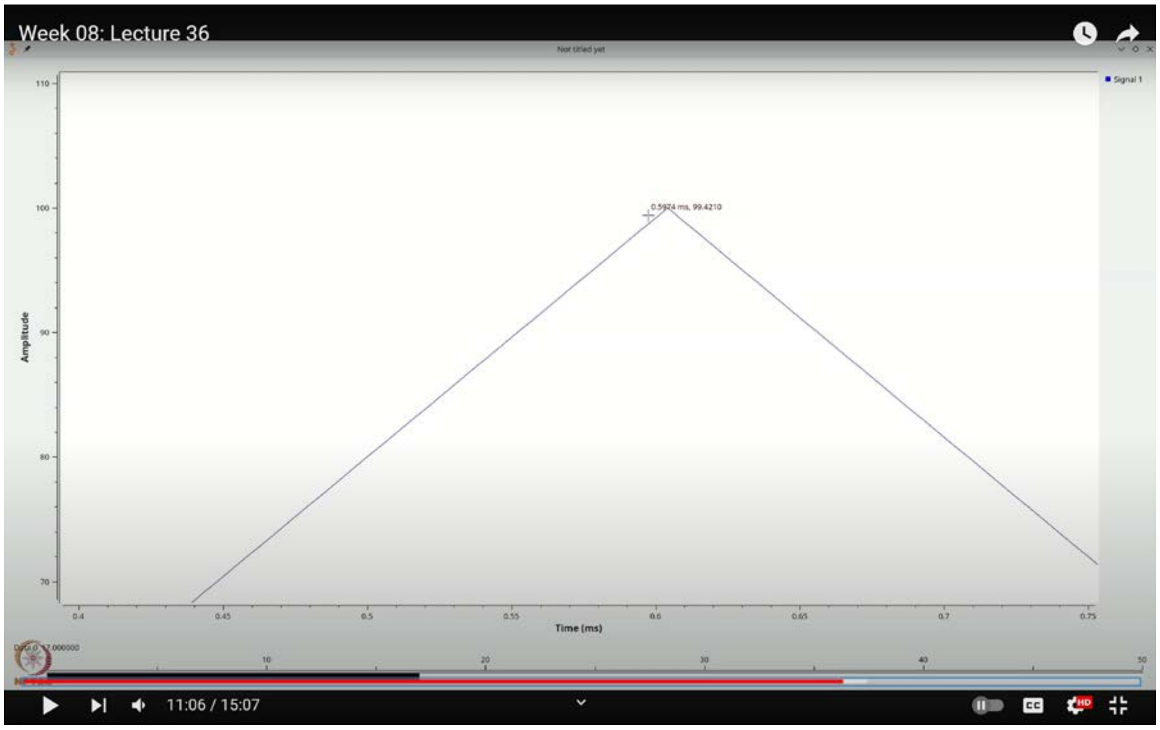
Next, we'll use NumPy to create a random delay. For this, I'll enter **`np.random.randint(0, 50)`** to generate a random integer between 0 and 49. Remember to import NumPy by pressing **`Ctrl + F`** (or **`Command + F`**) and adding the import block: **`import numpy as np`**.

Now we're all set. Every time you run this flow graph, a new random delay will be generated. Running the flow graph repeatedly will yield different random delays each time.

To confirm our results, let's add a way to view this delay. Press **`Ctrl + F`** (or **`Command + F`**) and search for "qt GUI number sync." This element will display a number representing our delay. Set the number sync to a short type, with inputs ranging from 0 to 50.

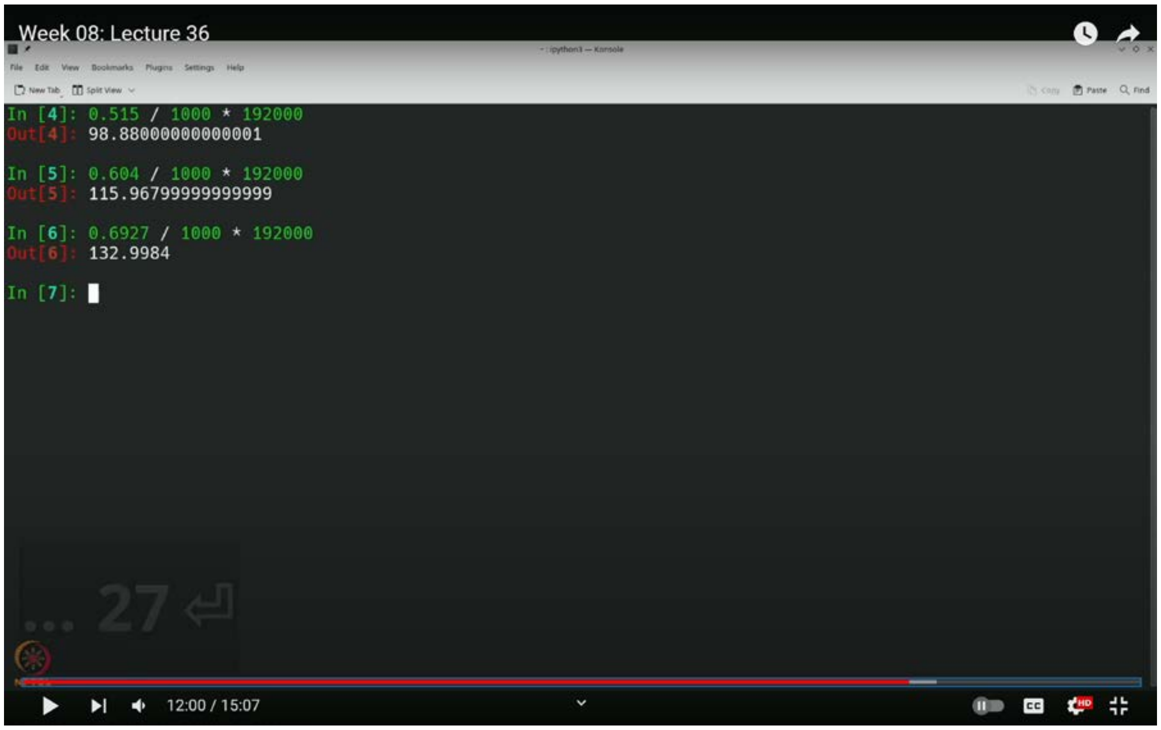
Next, add a "constant source" by pressing **`Ctrl + F`** (or **`Command + F`**) and searching for "constant source." Ensure it's of type short and set it to represent our **`random_delay`**.

(Refer Slide Time: 11:06)



Execute the flow graph, and you should see that the random delay is, for example, 17. To verify this, note the time at which the peak occurs, which might be 0.604 milliseconds. Converting this to seconds (0.604 / 1000) and multiplying by the sample rate (192,000) gives 116. This corresponds to about 117 samples of delay. When you take this delay modulo 50, it matches the random delay of 17 samples we observed. This confirms that our delay calculation is correct.

(Refer Slide Time: 12:00)



```
In [4]: 0.515 / 1000 * 192000
Out[4]: 98.88000000000001

In [5]: 0.604 / 1000 * 192000
Out[5]: 115.96799999999999

In [6]: 0.6927 / 1000 * 192000
Out[6]: 132.9984

In [7]:
```

Let’s run the experiment one more time. This time, instead of focusing on the details below, I’ll just observe the peak value. For instance, if the peak value is 0.6927 milliseconds, then converting this to samples, we get:

$$\frac{0.6927 \text{ ms}}{1000} \times 192{,}000 \approx 133 \text{ samples}$$

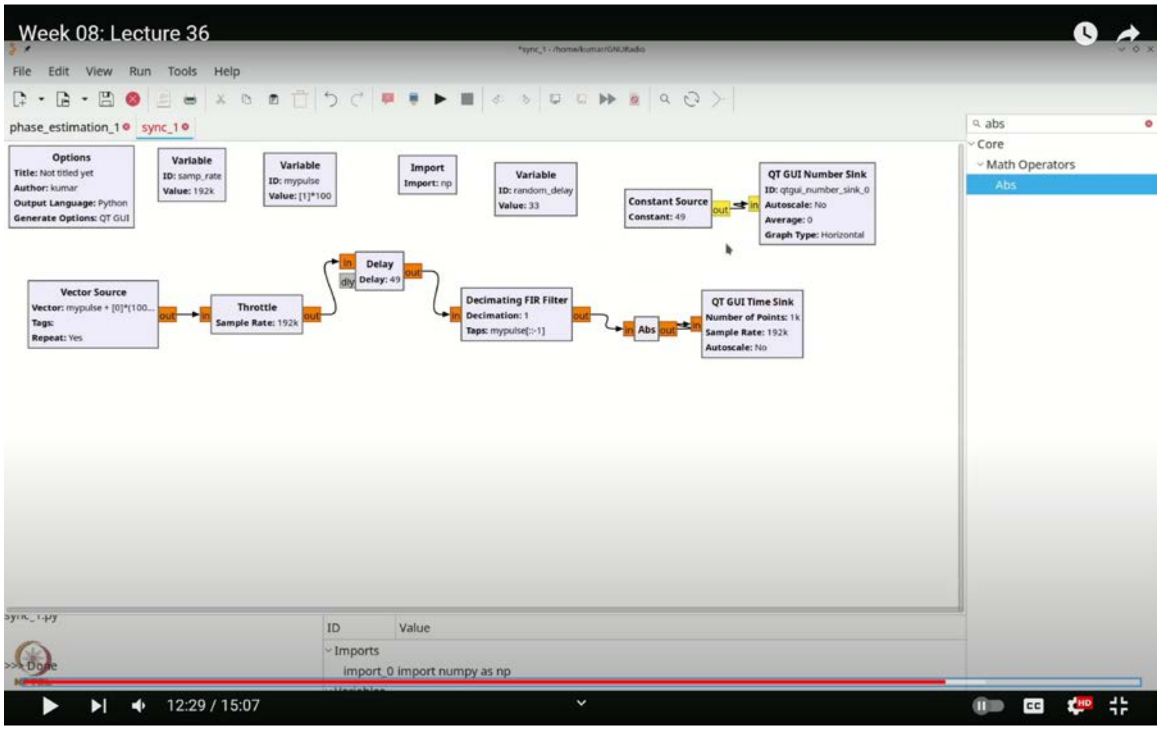
So, for this case, the modulo operation should yield 34. This confirms that our approach is correct.

However, a crucial point to note is that we should take the absolute value of the delay before performing any modulo operation. I had initially overlooked this step, so let me correct it by adding the absolute value function.

To do this, press `**Ctrl + F**` (or `**Command + F**`), search for "abs," and select the absolute value function. Connect this function to the relevant part of the flow graph.

Next, I'll modify the filter. Instead of using the current setup, let's switch to a ramp function by using `**np.arange**` to generate an array from -50 to 50, maintaining a length of 100. To handle the list appropriately, let's ensure it's treated as a list.

(Refer Slide Time: 12:29)



Options
Title: Not titled yet
Author: kumar
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 192k

Variable
ID: mypulse
Value: [1]*100

Import
Import: np

Variable
ID: random_delay
Value: 33

Constant Source
Constant: 49

QT GUI Number Sink
ID: qtgui_number_sink_0
Autoscale: No
Average: 0
Graph Type: Horizontal

Vector Source
Vector: mypulse + [0]*(100...
Tags:
Repeat: Yes

Throttle
Sample Rate: 192k

Delay
Delay: 49

Decimating FIR Filter
Decimation: 1
Taps: mypulse[::-1]

Abs

QT GUI Time Sink
Number of Points: 1k
Sample Rate: 192k
Autoscale: No

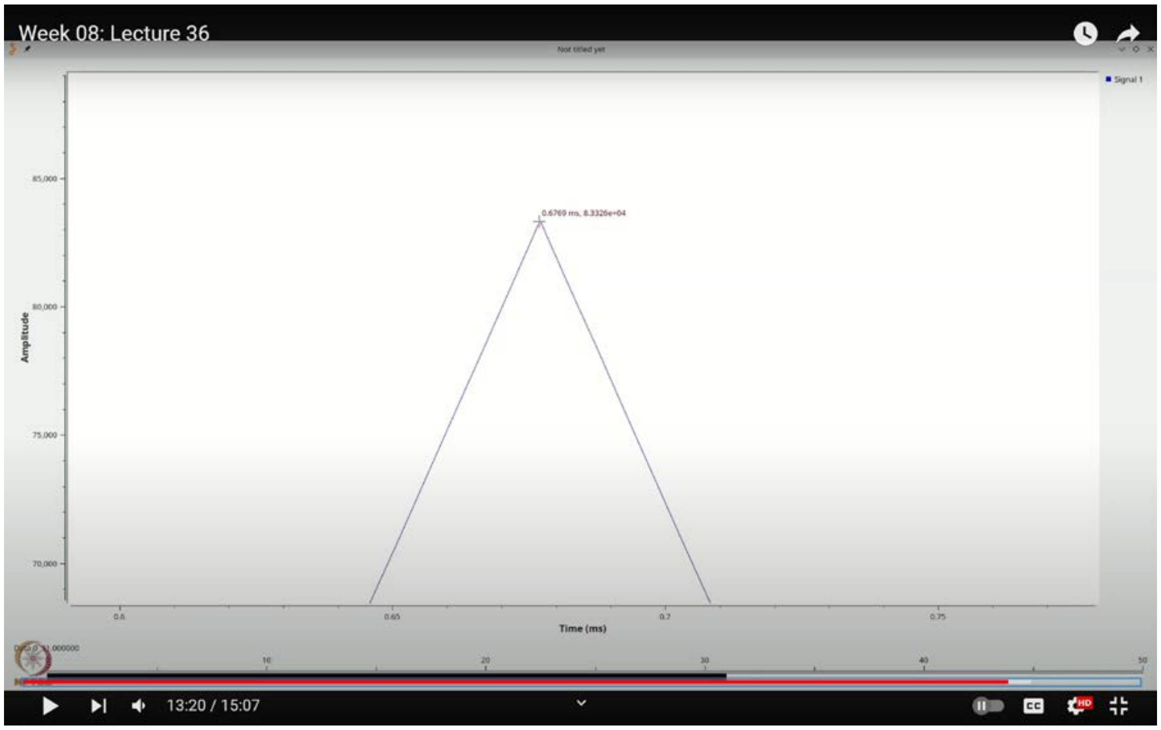
| ID | Value |
|---|---|
| Imports | |
| import_0 | import numpy as np |

Let's consider the peak value, which is 0.6767 milliseconds. Converting this to samples:

$$\frac{0.6767\text{ ms}}{1000} \times 192{,}000 \approx 130\text{ samples}$$

So, the expected result should be 131, and 131 mod 50 yields 31. This confirms that our method is accurate.

(Refer Slide Time: 13:20)



The approach discussed in the slides proves effective for estimating delays, but it's important to see how it performs in practical scenarios, especially with interpolation and pulse shapes closer to a root-raised cosine filter.

In this lecture, we’ve used GNU Radio to simulate the effect of a delay and applied the Maximum Likelihood (ML) delay estimation technique to appropriately characterize this delay.

A limitation of our current method is that it uses a single pulse, which aligns with our classroom example. However, practical applications often involve more complex signals with multiple data sequences. In the next lecture, we will integrate these aspects into a simulation to evaluate its effectiveness and determine any necessary adjustments for such scenarios. Thank you.