20CYS312 - Principles of Programming Languages
Exploring Programming Paradigms

Assignment-01

Presented by Mukesh SA
CB.EN.U4CYS21046
TIFAC-CORE in Cyber Security
Amrita Vishwa Vidyapeetham, Coimbatore Campus

Feb 2024

AMRITA
VISHWA VIDYAPEETHAM

Outline

DataFlow Paradigm

- DataFlow programming paradigm is characterized by the flow of data through a network of processing nodes, where computations are triggered as data becomes available.
- In this paradigm, emphasis is placed on the dynamic movement and transformation of data, enabling parallelism and responsiveness in the execution of programs.

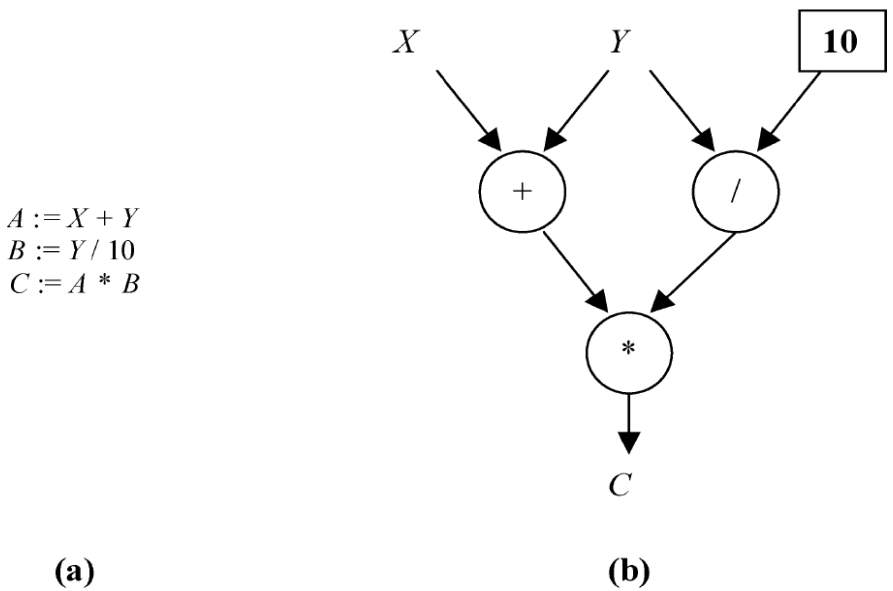

Figure: Basic Idea of DataFlow

DataFlow Paradigm

- How it works:
  - Execution is based on data movement, with programs structured as directed graphs. Nodes represent operations, and edges denote the flow of data. Execution occurs when data becomes available, not explicitly through control flow.

DataFlow Paradigm

- Features:
  - Parallelism: Tasks execute concurrently as soon as their input data is ready.
  - Implicit Synchronization: Dependencies between operations are automatically managed, reducing the need for explicit synchronization.

DataFlow Paradigm

- Uses:
  - Parallel Processing: Efficient for tasks breakable into parallelizable subtasks, enhancing performance on multi-core systems.
  - Streaming Data Processing: Ideal for continuous data applications like real-time analytics and signal processing.

DataFlow Paradigm

- Disadvantages:
  - Complexity: Designing and debugging dataflow programs can be challenging due to the implicit nature of control flow.
  - Resource Management: Efficient allocation and management of resources can be difficult, impacting overall system performance.

LabVIEW - DataFlow Programming

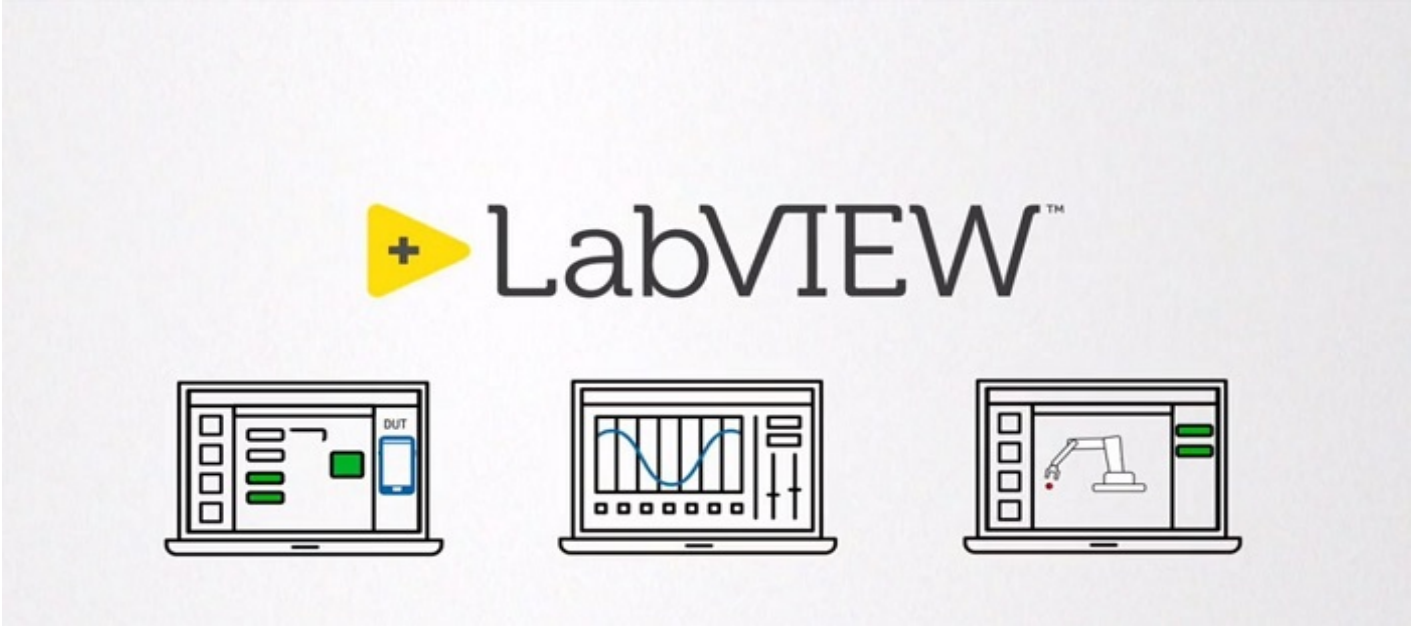

Figure: LabVIEW

LabVIEW - DataFlow Programming

1. LabVIEW - Laboratory Virtual Instrument Engineering Workbench.
2. LabVIEW is a graphical programming language developed by National Instruments.
3. LabVIEW is widely used in scientific and engineering applications for designing virtual instruments and automating measurement systems.
4. It allows users to connect functional blocks visually, making it accessible for both engineers and scientists without extensive coding backgrounds.
5. LabVIEW is versatile, supporting a wide range of hardware platforms and devices, making it a popular choice for automation, control, and data acquisition in various industries.

LabVIEW - DataFlow Programming

Components:

1. Block diagram: The code that controls the program. The block diagram window contains terminals, constants, functions, SubVIs, structure, and wires that connect data from one object to another.
2. Connector pane: Represents the VI in the block diagrams of others, calling VIs.

LabVIEW - DataFlow Programming

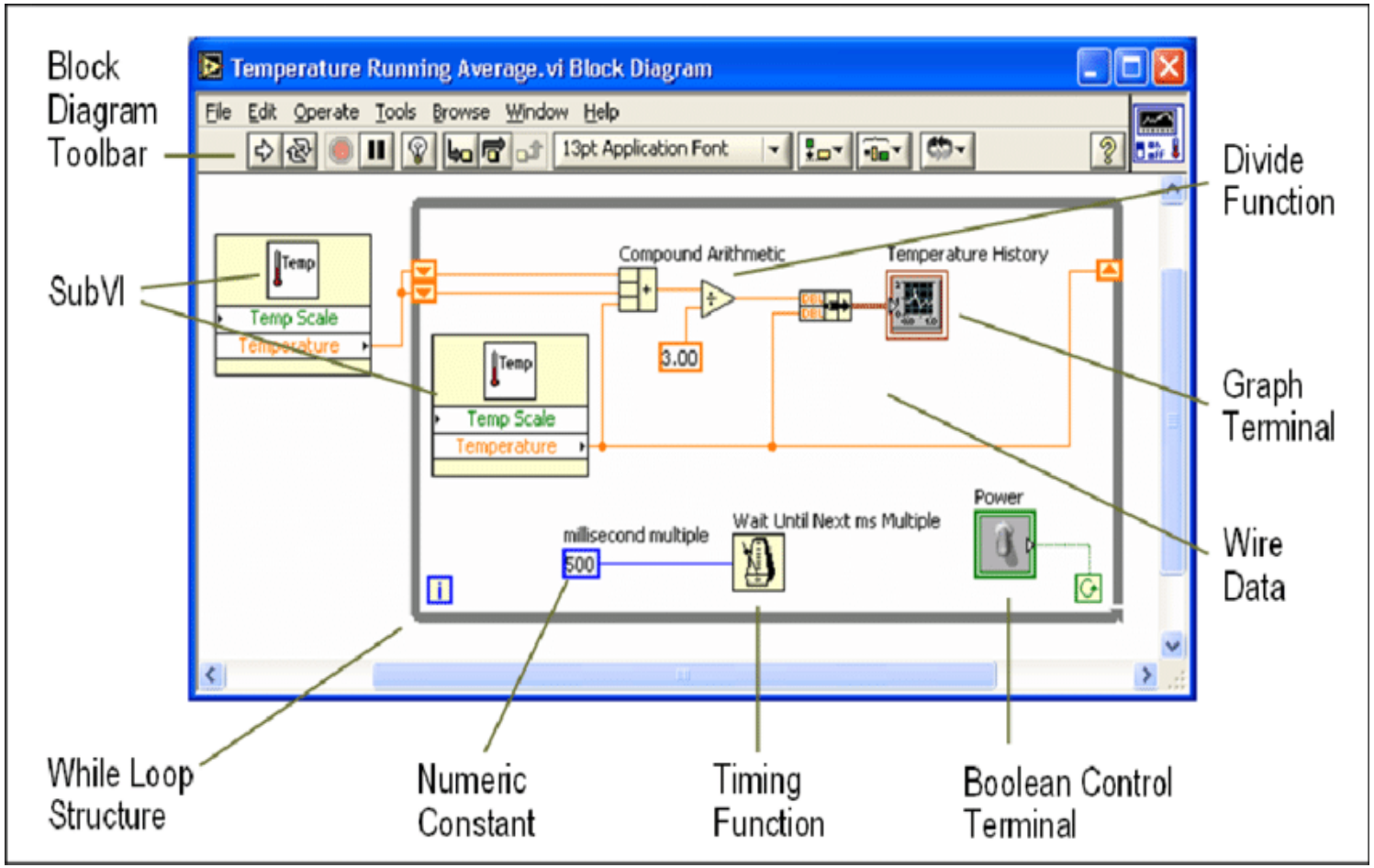

Figure: LabVIEW Block Diagram

LabVIEW - DataFlow Programming

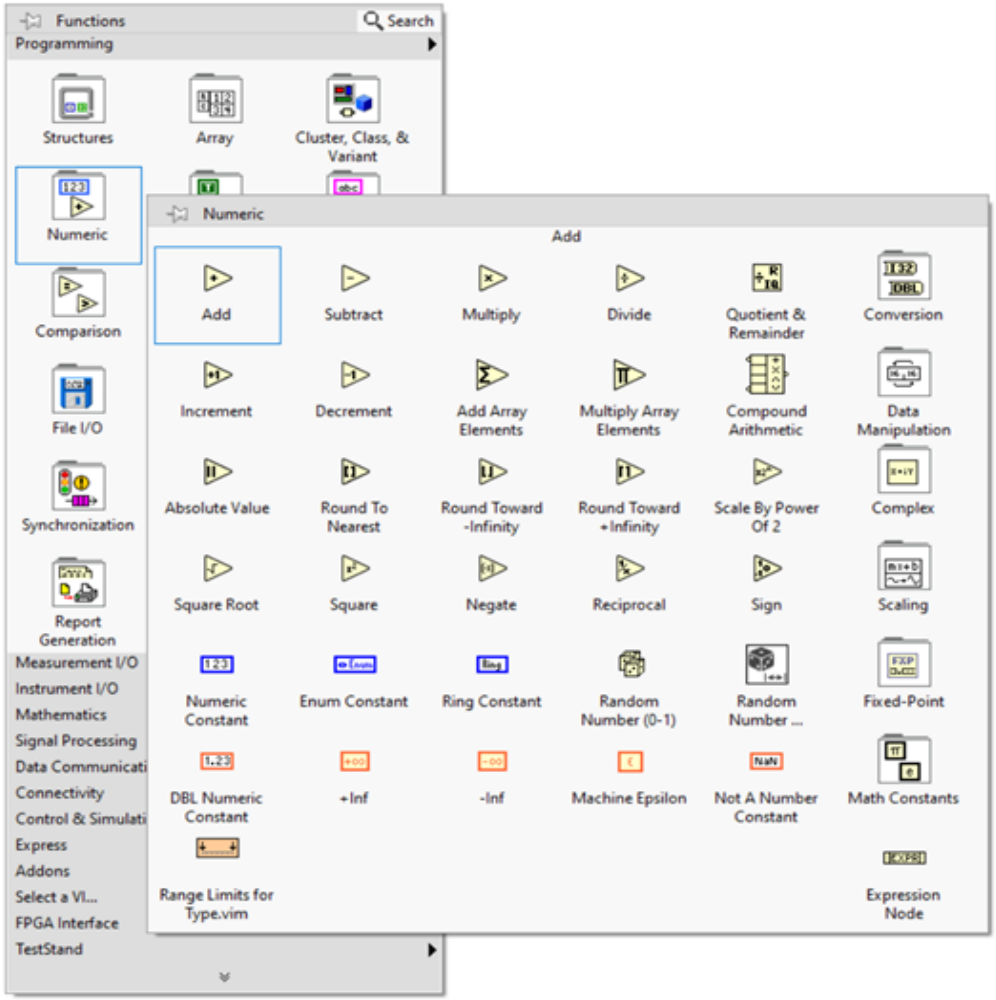

Figure: LabVIEW Components

LabVIEW - DataFlow Programming

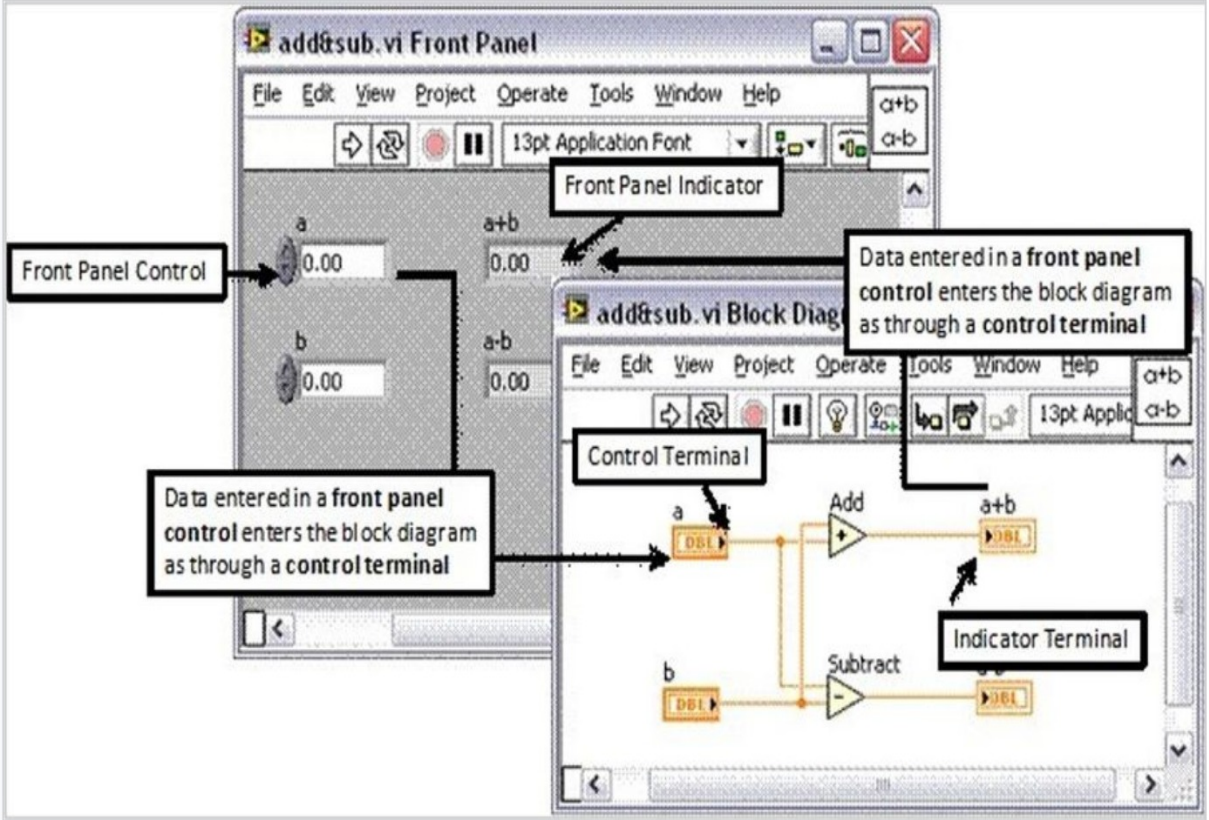

Figure 4: Control and Indicator Terminals

Figure: LabVIEW Assemble

LabVIEW - DataFlow Programming

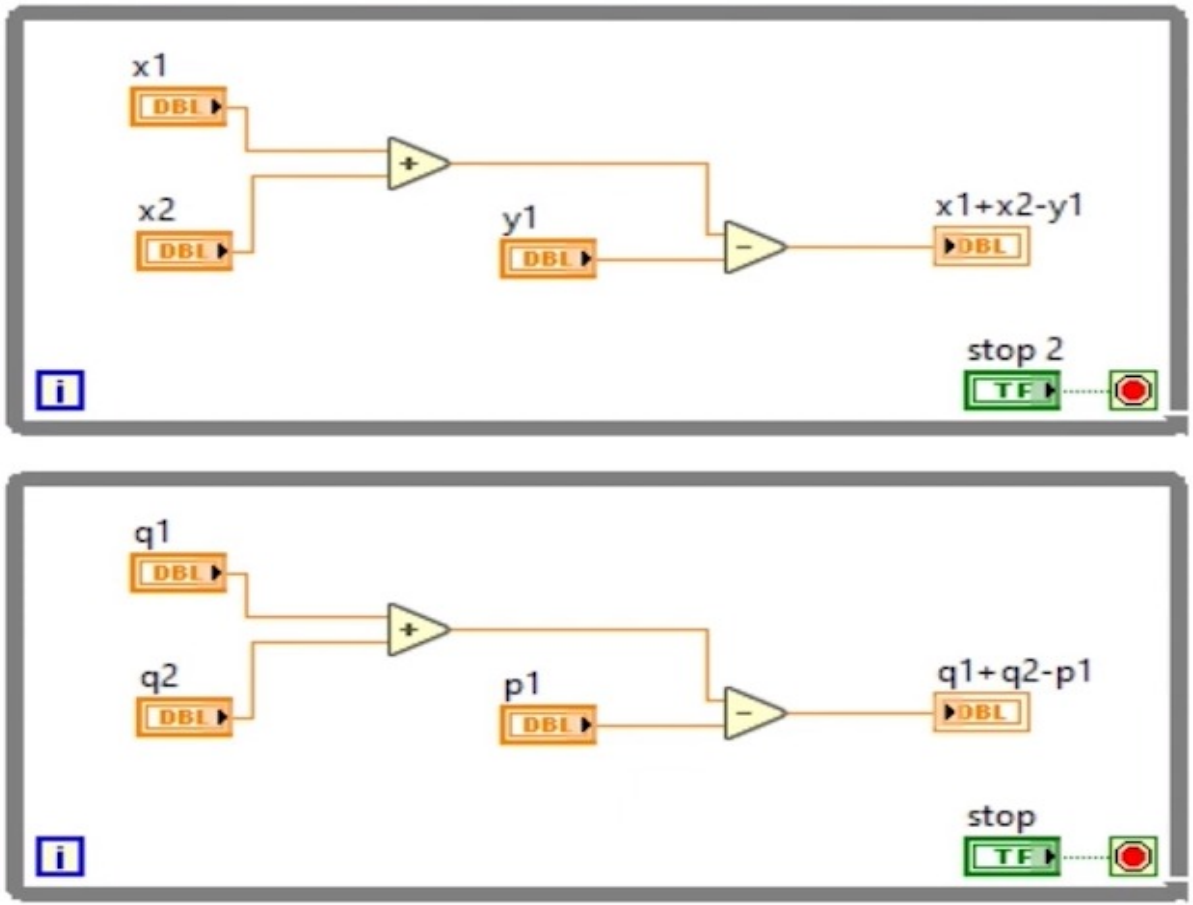

Figure: Parallel Processing in LabVIEW

LabVIEW - DataFlow Programming

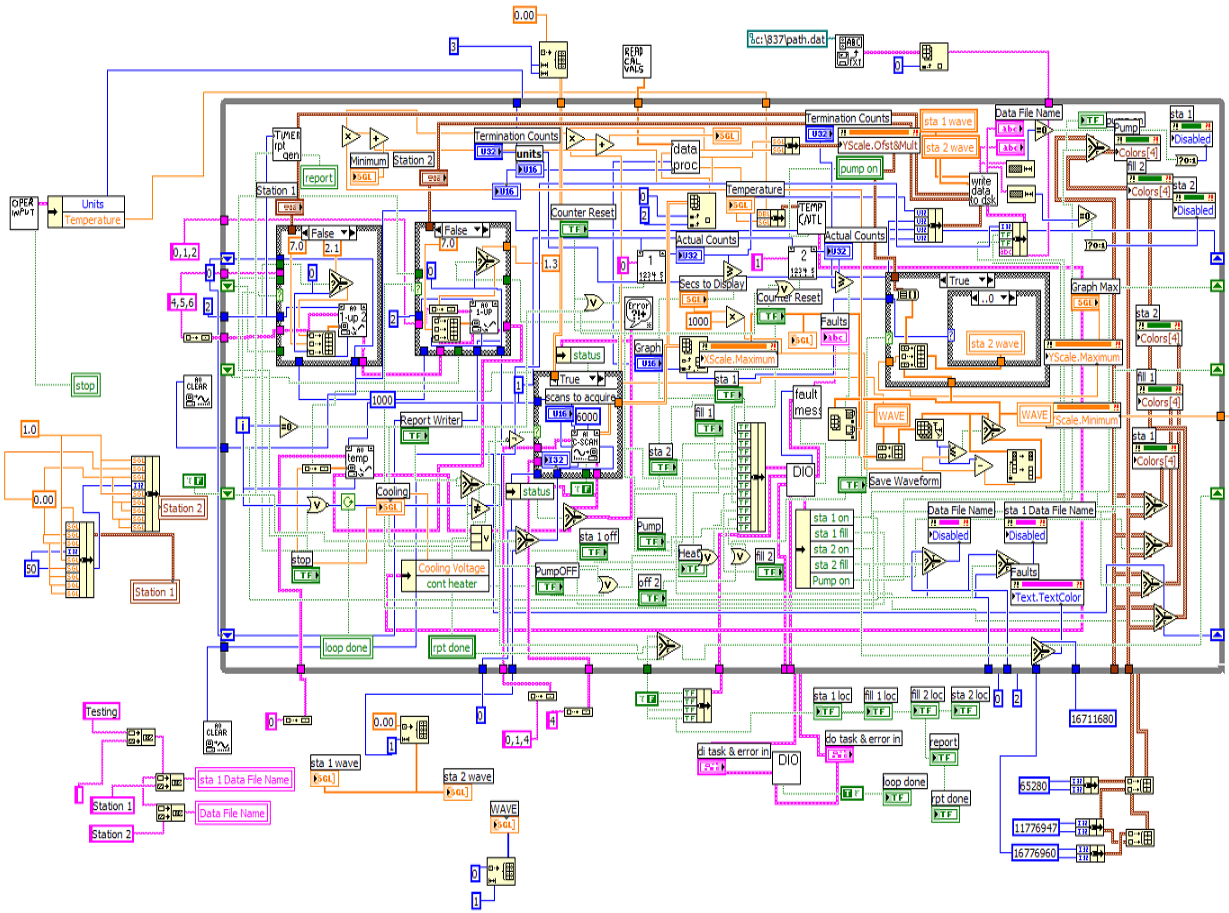

Figure: To Explain Disadvantage

Reactive Programming Paradigm

- Reactive programming is like setting up a chain reaction in software. Instead of writing code that explicitly tells the computer what to do step by step, you define how your program should react to changes or events.
- It's as if you're saying, "Hey, if this data changes, do something automatically without me having to specifically instruct you each time."

Figure: Rx Java

RxJava in Reactive Programming

- Reactive Programming Paradigm: JavaRx, short for Reactive Extensions for the Java Virtual Machine, introduces a powerful reactive programming paradigm. It enables developers to work with asynchronous and event-driven programming effortlessly.
- Composable and Declarative: JavaRx promotes a composable and declarative style of coding, allowing developers to express complex asynchronous operations in a concise and readable manner. This makes it easier to manage and reason about reactive code.

RxJava in Reactive Programming

- Observer Pattern on Steroids: At its core, JavaRx implements the observer pattern, but with added functionalities and enhancements. It provides a rich set of operators for transforming, filtering, and combining streams of data, making it a versatile tool for handling real-time events.
- Cross-platform Compatibility: One of JavaRx's strengths lies in its cross-platform compatibility. Developers can leverage its reactive capabilities not only in Java applications but also in other JVM languages, making it a valuable tool for building robust and responsive systems.
- Wide Industry Adoption: JavaRx has gained widespread adoption in various industries, from enterprise applications to mobile development. Its ability to handle concurrency, manage complex event flows, and improve responsiveness has positioned it as a go-to solution for building scalable and reactive software.

Comparing DataFlow and Reactive Paradigms

- Execution Model:
  - DataFlow: Execution is triggered by the availability of data. Nodes represent operations, and the program's structure is a directed graph. Parallelism is inherent, and tasks execute as soon as input data is ready.
  - Reactive: Execution is event-driven, reacting to changes or events. It follows the observer pattern, where components (observers) react to changes in the state of the subject. It provides a declarative and composable way to express asynchronous operations.

Comparing DataFlow and Reactive Paradigms

- Concurrency and Parallelism:
  - DataFlow: Designed for parallelism, tasks can execute concurrently as data becomes available. Implicit synchronization manages dependencies between operations.
  - Reactive: Supports concurrency by handling asynchronous events. Reactive systems can handle multiple events concurrently, making it suitable for building responsive and scalable applications.

Comparing DataFlow and Reactive Paradigms

- Programming Model:
  - DataFlow: Programs are structured as directed graphs where nodes represent operations. Execution is based on the dynamic flow of data through the graph.
  - Reactive: Programs are written to react to changes or events. It emphasizes a more declarative and expressive style, making it easier to handle asynchronous operations.
- Use Cases -
  - DataFlow: Well-suited for tasks that can be broken into parallelizable subtasks, enhancing performance on multi-core systems. Ideal for streaming data processing in real-time applications.
  - Reactive: Particularly effective for applications with dynamic and changing data, where responsiveness to events is crucial. Commonly used in user interfaces, real-time analytics, and systems with frequent state changes.

References

- ni.com
- baeldung rx-java
- reactivex.io
- geeksforgeeks
- youtube
- wikipedia