Autorouting Process Flow Documentation

This documentation describes the flow of the **autorouting process** within the system. It explains what happens after the **"Autoroute" button** is pressed in the user interface (UI) and how the autorouting instance is created and executed, followed by optimization, error handling, and final feedback to the user.

**1. User Clicks the "Autoroute" Button**

* **File**: BoardToolbar.java (lines 92 to 114)
* **Description**: When the user clicks the **Autoroute** button in the toolbar, an action listener is triggered in the BoardToolbar.java class. This button is associated with an event handler that will begin the autorouting process.

**Flow**:

* + The **Autoroute button** is defined in the toolbar.
  + The button’s event listener listens for user interaction and starts the **autorouting process** when clicked.

**2. Trigger the Autorouting Process**

* **File**: BoardHandling.java
* **Function**: start\_autorouter\_and\_route\_optimizer()
* **Description**: After the button is clicked, the start\_autorouter\_and\_route\_optimizer() method is invoked in BoardHandling.java. This method is responsible for starting a new thread to handle the autorouting and optimization process.

**Flow**:

* + The **start\_autorouter\_and\_route\_optimizer()** function prepares the environment for autorouting.
  + It initializes the autorouting process and calls the appropriate instance that manages autorouting and post-route optimization.

**3. Creating Autorouter and Optimizer Thread**

* **File**: AutorouterAndRouteOptimizerThread.java
* **Function**: Constructor (AutorouterAndRouteOptimizerThread())
* **Description**: The function get\_autorouter\_and\_route\_optimizer\_instance() (likely in the BoardHandling.java file) calls the constructor of the AutorouterAndRouteOptimizerThread class. This constructor is the **main point** where the autorouter and optimizer instance is created.

**Flow**:

* + An instance of AutorouterAndRouteOptimizerThread is created.
  + The class manages both **autorouting** and **post-routing optimization** processes.
  + The constructor is responsible for preparing any necessary settings or parameters related to the autorouting algorithm and initializing the thread.

**4. Executing the Autorouting Algorithm**

* **File**: BatchAutorouter.java
* **Function**: autoroute\_passes()
* **Description**: Within the AutorouterAndRouteOptimizerThread class, the autoroute\_passes() method in BatchAutorouter.java is invoked. This method is the core of the autorouting algorithm, managing the iterative process of routing different board components over multiple passes.

**Flow**:

* **Start Autorouting Passes**: **autoroute\_passes()** initiates a sequence of routing passes over the board, attempting to route connections between components according to design rules and constraints.
* **Loop Through Each Item**:
  + For each pass, it iterates through items that need to be routed, checking connectivity and any remaining unrouted connections.
* **Skip Already Routed or Redundant Connections**:
  + If a specific board state has already been evaluated in a previous pass, the algorithm avoids redundant processing by recognizing previously routed paths.

**4.1. Routing Individual Items**

* **File**: BatchAutorouter.java
* **Function**: autoroute\_item()
* **Description**: During each pass in autoroute\_passes, the autoroute\_item() method is called for individual items. This method checks if an item is already fully connected and, if not, attempts to establish connections within its assigned net.

**Flow:**

* **Identify Unconnected Sets**:
  + autoroute\_item() identifies the unconnected and connected sets of the item in question, determining start and destination points for routing.
* **Configure Autoroute Control**:
  + It then configures AutorouteControl settings specific to the item's net, trace width, and via costs, preparing the routing environment.
* **Execute Routing for Item**:
  + If the item is not fully connected, autoroute\_item() attempts to establish connections by invoking the autoroute\_connection() method within AutorouteEngine.

**4.2. Establishing Connections Using Maze Search**

* **File**: AutorouteEngine.java
* **Function**: autoroute\_connection()
* **Description**: The autoroute\_connection() function in AutorouteEngine performs the actual routing attempt between the start and destination sets of the item by initializing a maze search algorithm.

**Flow:**

* **Initialize Maze Search:**
  + autoroute\_connection() uses MazeSearchAlgo.get\_instance() to set up a new MazeSearchAlgo instance. This instance initializes the maze search algorithm, which attempts to find a path between the start and destination items.
* **Run Maze Search Algorithm:**
  + File: MazeSearchAlgo.java
  + Function: find\_connection()
  + If the initialization succeeds, find\_connection() is called, which uses a priority-based search (maze\_expansion\_list) to find a viable path between the start and destination items.
* **Evaluate Routing Path:**
  + If a route is found, find\_connection() returns a result that can then be further processed for insertion into the board.

**4.3. Inserting Found Connections**

* **File:** LocateFoundConnectionAlgo.java, InsertFoundConnectionAlgo.java
* **Functions:** LocateFoundConnectionAlgo.get\_instance(), InsertFoundConnectionAlgo.get\_instance()
* **Description:** If a path is found by MazeSearchAlgo, the path is analyzed and then inserted into the board. The algorithms check if the path is valid and, if so, commit it to the board layout.

**Flow:**

* **Validate and Insert Connection:**
  + LocateFoundConnectionAlgo.get\_instance() evaluates the found path, and InsertFoundConnectionAlgo.get\_instance() performs the actual insertion, updating the board layout.
* **Handle Ripped Connections:**
  + During this process, autoroute\_connection may remove ("rip up") existing connections to make room for the new route, updating p\_ripped\_item\_list to track these changes.

**4.4. Intermediate Checks and GUI Updates**

* **File:** BoardHandling.java
* **Function:** set\_batch\_autoroute\_info()
* **Description:** After each pass, intermediate status updates are sent to the user interface to reflect the current progress of the autorouting process.

**Flow:**

* **Display Progress Updates:**
  + The autorouting progress (e.g., number of items routed, remaining items) is displayed to the user, offering real-time feedback on each pass’s status.
* **Save Intermediate Stages (if enabled):**
  + If the save\_intermediate\_stages flag is enabled, the current state of the board is saved after each pass, allowing the user to access intermediate results.

**4.5. Finalization of Routing Passes**

* **File:** BatchAutorouter.java
* **Function:** autoroute\_passes()
* **Description:** After routing all items within each pass, the function concludes the routing pass. If all items are routed or certain stopping conditions are met (e.g., minimal improvements), autoroute\_passes exits the loop.

**Flow**:

* **Finalize Pass:**
  + Once the final routing pass is completed, any unconnected vias are removed if remove\_unconnected\_vias is enabled, and the board is prepared for optional post-routing optimization.
* **Clear State for Next Steps:**
  + Temporary states, like already\_checked\_board\_hashes, are cleared, ensuring the autorouter is ready for any subsequent steps.

**5. Post-Routing Optimization (Optional)**

* **File**: BatchOptRoute.java or BatchOptRouteMT.java
* **Function**: optimize\_board()
* **Description**: Once autorouting is complete, the system checks if post-routing optimization is enabled. If so, the optimize\_board() method is called to improve the layout further by reducing trace lengths and optimizing the number of vias.

**Flow**:

* + Post-routing optimization is an optional step, and it only runs if enabled.
  + The optimize\_board() function is responsible for refining the routing to improve the board’s manufacturability and performance.
  + It optimizes by minimizing the number of **vias**, reducing total **trace length**, and improving overall **signal integrity**.

**6. Status Updates and User Feedback**

* **File**: BoardHandling.java
* **Function**: set\_status\_message()
* **Description**: Throughout the autorouting and optimization process, status messages are displayed to the user. The method set\_status\_message() is used to provide real-time feedback on the progress of the routing.

**Flow**:

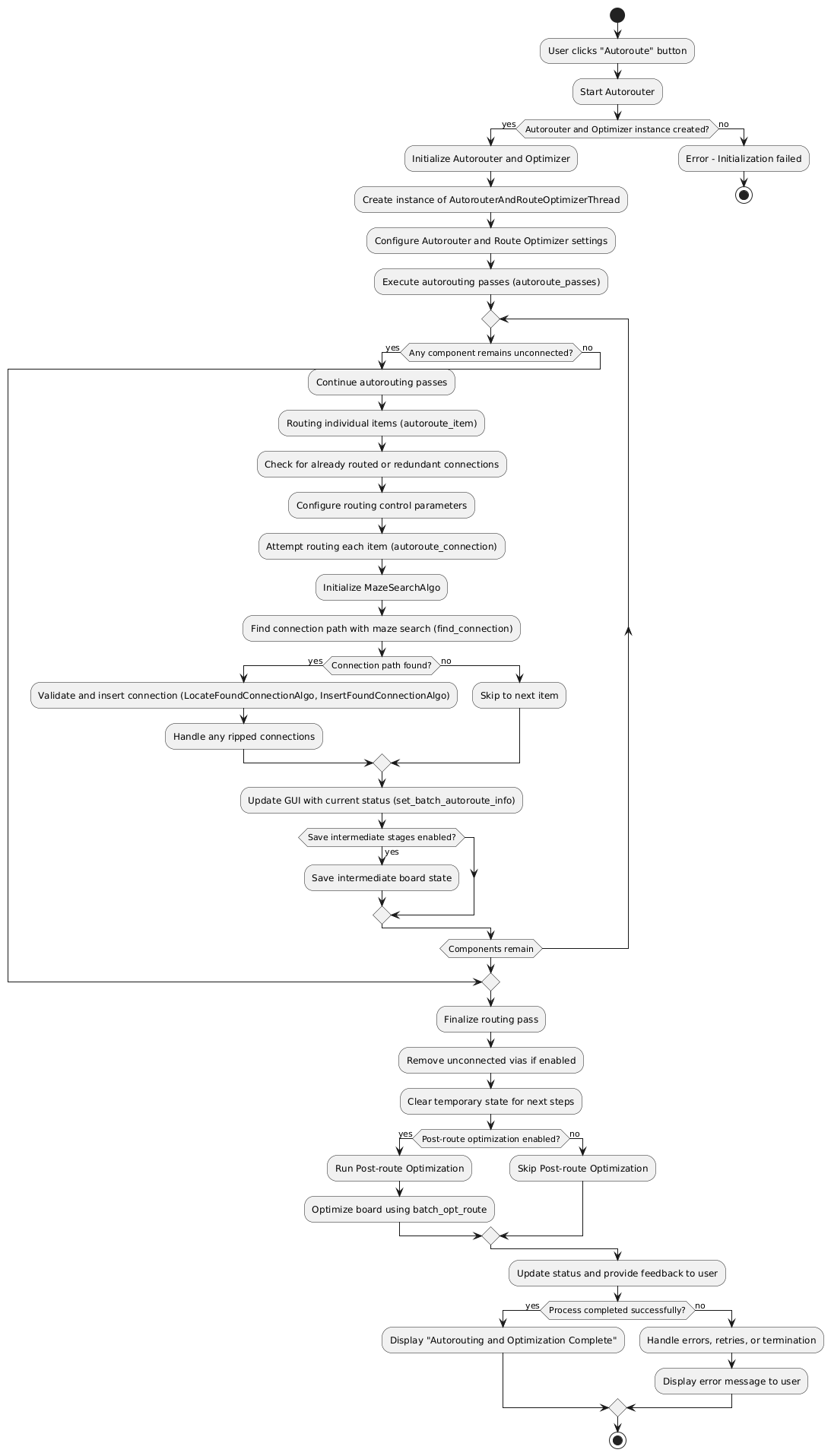
* + Status messages are updated at key points in the process: autorouting start, autorouting passes, optimization start, and completion.
  + The user is kept informed about what’s happening, including any errors that occur during the process.

**7. Process Completion**

* **File**: AutorouterAndRouteOptimizerThread.java
* **Description**: After autorouting and (optional) post-routing optimization are complete, the final status is presented to the user. The autorouter thread terminates, and any generated routing files or intermediate results are saved.

**Flow**:

* + The autorouting process terminates.
  + The board's routing and optimization are completed, and the user is informed.
  + The system may save the result (either automatically or upon user request) for further analysis or production.



Autorouting Process Flow Diagram