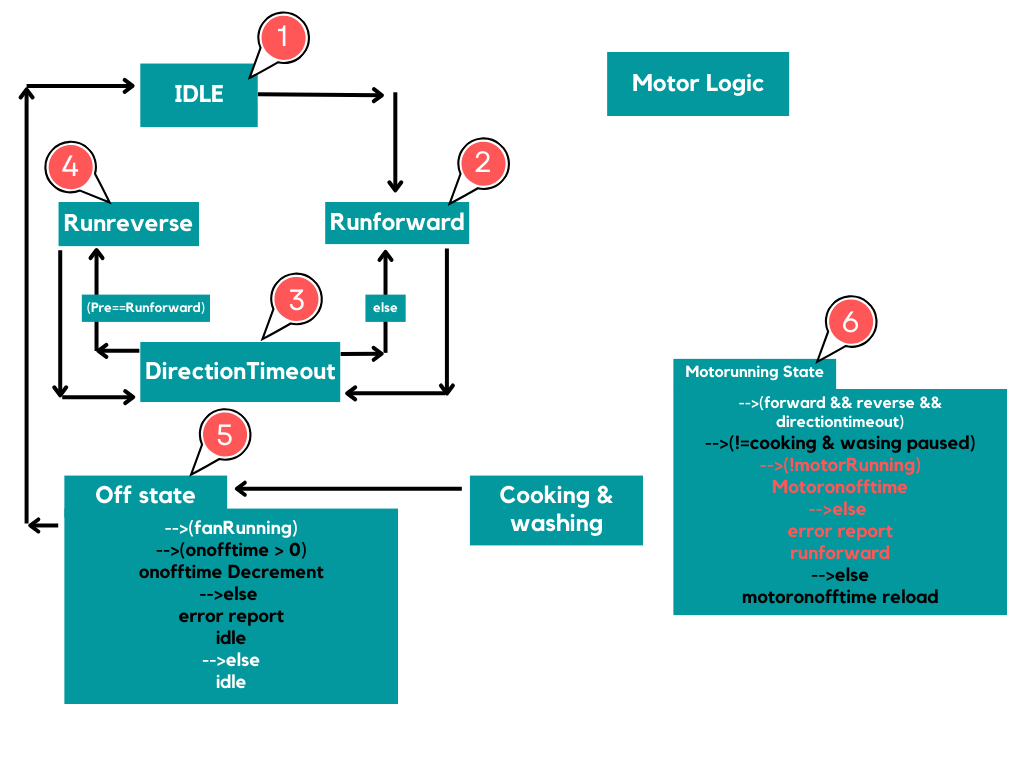
**Fan Motor logic**



**Header Inclusions:**

* This section includes various header files that provide essential functionalities and definitions for the program.
* "**IOIF.h**" likely contains functions and macros related to input and output operations.
* "**Includes.h**" may include general-purpose libraries or system-specific definitions.
* "**DEBUG\_H**" may provide debugging utilities or macros.
* "**DAC\_H**" might contain functions and macros for controlling the DAC (Digital-to-Analog Converter).
* "**DIAG\_H**" could include diagnostics-related functions and macros.
* "**DIAGIDS\_H**" might define unique identifiers for diagnostic events

**Macro Definitions:**

* Macros in this section define constants used throughout the code, making it easier to configure and adjust the program's behaviour.
* **MOTOR\_REVERSE\_SWITCHING\_WAITIME** sets the timeout for motor reverse switching to 30 seconds.
* **MOTOR\_DIRECTION\_SWITCH\_TIMEOUT** configures the timeout for motor direction switching to 5 minutes.
* **MOTOR\_ON\_OFF\_TIMEOUT** to check the on/off operation wait time Check counter timeout (15 seconds).
* **MOTOR\_SWITCHED\_TIMEOUT** specifies the timeout for motor switching (3 seconds).

**Global Variable Declarations:**

* In this section, global variables are declared, which store important information about the motor control and its state.
* **fanMotorControl** and **prevMotorControlState** are enumerated types representing the current and previous motor control states, respectively.
* **fanMotorRunningStatus** and **PrvRunstatus** are Boolean flags indicating whether the fan motor is currently running. They are used for tracking the motor's status.
* **motorDirectionSwitch**, **motorOnOffTimeout**, and **motorSwitchingTimeout** are counters used for managing timeouts related to motor control operations.

**Function Declarations:**

* This section declares function prototypes for two functions:
  + **updateMotorSpeed**: This function calculates and sets the motor speed based on a threshold level and input data using a DAC.
  + **motorControl**: The main motor control function responsible for managing the fan motor's operation.

**Flow of states**

**🡪Motor \_ Idle**

🡪**Motor\_Forward**

🡪**Motor\_DirectionTimeout**

🡪**Motor\_Reverse**

**🡪Motor\_DirectionTimeout**

🡪 **Motor\_Off**

**E\_fanMotorIdle** :

* This state represents an idle or inactive state for the fan motor.
* No specific actions related to motor control are taken in this state.

**E\_fanMotorRunForward**:

* This state is represented by the case **E\_fanMotorRunForward** in the switch statement.
* The code within this state focuses on running the fan motor in the forward direction.
* It checks a timeout condition (**motorSwitchingTimeout**) to ensure that the motor switch changes are managed properly.
* When **motorSwitchingTimeout** is Equal to **MOTOR\_SWITCHED\_TIMEOUT**, it turns off the reverse motor switch (**TURN\_REVERSE\_MOTOR\_OFF()**) and it wait for timeout then turns on the forward motor switch (**TURN\_FORWARD\_MOTOR\_ON()**).
* It updates various variables to keep track of the motor's state and sets a timeout for motor direction switching (**motorDirectionSwitch**).
* Finally, it transitions to the state **E\_fanMotorDirectionTimout** for further motor control.

**E\_fanMotorRunReverse:**

* This state is represented by the case **E\_fanMotorRunReverse** in the switch statement.
* Similar to the forward state, it manages the motor's direction, but this time for running in reverse.
* It also checks the **motorSwitchingTimeout** and, when necessary, turns off the forward motor switch and turns on the reverse motor switch.
* Like the forward state, it updates variables and sets a timeout for direction switching before transitioning to **E\_fanMotorDirectionTimout**.

**E\_fanMotorDirectionTimout**:

* This state is represented by the case **E\_fanMotorDirectionTimout** in the switch statement.
* It handles the periodic switching of the motor's direction (forward to reverse or reverse to forward).
* It checks a timeout condition (**motorDirectionSwitch**) and decrements it.
* When this timeout reaches zero, it changes the motor's direction based on the previous state (forward or reverse).
* After the direction switch, it resets the direction switch timeout (**motorDirectionSwitch**) for the next cycle.

**E\_fanMotor\_OFF:**

* This state is represented by the case **E\_fanMotor\_OFF** in the switch statement.
* It is responsible for turning off the fan motor.
* It ensures both the forward and reverse motor switches are turned off (**TURN\_REVERSE\_MOTOR\_OFF()** and **TURN\_FORWARD\_MOTOR\_OFF()**).
* There's a check to ensure that the motor has actually turned off. If it hasn't, it logs an error and transitions back to **E\_fanMotorIdle**.
* It also handles a timeout (**motorOnOffTimeout**) to avoid instant error logging in case the motor doesn't turn off immediately.

**Motor\_running\_state check**

**🡪((fanMotorControl == E\_fanMotorRunForward) || (fanMotorControl == E\_fanMotorRunReverse) || (fanMotorControl == E\_fanMotorDirectionTimout))**

🡪That checks if the **fanMotorControl** variable is set to either **E\_fanMotorRunForward**, **E\_fanMotorRunReverse**, or **E\_fanMotorDirectionTimout**. These states Ensures that the fan motor is in running state.

**🡪((MS->MachineStaus != MS\_cookingPaused) && (MS->MachineStaus != MS\_washingPaused))**

**🡪,**It checks if the machine status (MS->MachineStaus) is not in a paused state.

🡪**(!IS\_FAN\_MOTOR\_RUNNING())**

🡪If the motor is not running, it proceeds to reports an error using Diag\_ReportEvent().