COVERSHEET

THE UNIVERSITY OF WESTERN AUSTRALIA

Assignment, Report & Laboratory Coversheet for Individual & Group Assignment

SUBMITTING STUDENT			
SURNAME Leveridge-Smith	GIVEN NAMES Jayden	STUDENT NUMBER 22274262	
UNIT NAME Process Instrumentation and Control		UNIT CODE ELEC5506	
TITLE/TOPIC OF ASSIGNMENT PIC Design Project		NAME OF LECTURER/TUTOR Brett Nener	
DATE/TIME DUE 15/05/2025, 11:59PM		DATE/TIME SUBMITTED 15/05/2025	

HONOURS STUDENTS ONLY	OFFICE USE ONLY
By signing this document, I further assert that the length (word count) of my dissertation is within the maximum allowed length governed by the project unit I am enrolled in. Penalties, as outlined on this website, will be applied for over length dissertations.	

FOR GROUP ASSIGNMENTS ONLY NAME	STUDENT NUMBER
1. Jayden Leveridge-Smith	22274262
2. Danielle Peralta	22891379
3. Elyney Ou	24260533
4. Kate Miller	22965308
5. Dhrumil Ghanshyambhai Bhanderi	24047059
6.	
7.	
8.	

Unless other arrangements have been made it will be assumed that all group members have contributed equally to group assignments/laboratory reports

DECLARATION

I/We are aware of the University's policy on academic conduct (see over) and I/We declare that this assignment/project is my own/my group's work entirely and that suitable acknowledgement has been made for any sources of information used in preparing it. I/We have retained a hard copy for my/our own records.

SIGN: J.Leveridge-Smith	SIGN: D.Ghanshyambhai Bhanderi
SIGN: D.Peralta	SIGN:
SIGN: E.Ou	SIGN:
SIGN: K.Miller	SIGN:

NOTE: No assignment will be accepted without the declaration above being signed and dated

SEE OVER FOR INFORMATION ON UNIVERSITY POLICY (UP07/21)

ELEC5506:
Process Instrumentation Controls

PIC Design Project Report

Group 1:

Dhrumil Ghanshyambhai Bhanderi: 24047059
Jayden Leveridge-Smith: 22274262
Kate Miller: 22965308
Elyney Ou: 24260533
Danielle Peralta: 22891379

16th May 2025

Table of Contents

Table of Figures

Table of Tables

Abbreviation List

Abbreviation	Meaning
Cmd	Command
EOC	End of Conveyor
FBD	Function Block Diagram
PLC	Programmable Logic Controller
ST	Structured Text
SOC	Start of Conveyor

1. Project Overview-

The project focuses on the development and implementation of a ST model in CodeSys, with the goal of controlling a production line to fill a $1m^3$ box with polystyrene chips. The CODESYS platform and Structured Text Language give a path to manage the sensor's level (acoustic, proximity, level switch), conveyor belt, and hopper gate along with to robot arms at the start and end of the conveyor belt.

1.1 Project Objectives

The objective of this report is to describe the high-level hardware of the conveyor system and explain the design of its operation and control based on the PackML standard. The report describes how the program has been implemented using Structured Text and gives insight to how to operator/user can modify the code to achieve their desired functionality.
A user manual has also been developed to ensure that the operator understands how the system can be controlled and tested within a virtual environment.

1.2 Description of Conveyor System

An overview of conveyor system is illustrated in Figure 1. The conveyor system is one individual cell apart of a larger production line. Figure 1 is only responsible for the filling component of the box. The designed cell will control the robot arm at the start of the conveyor where it will pick up pre-made empty boxes and place them within the system. Once the box has been filled, a signal will be sent to the next cell in the production line to pickup the full box where it is ready for the next stage of processing. The scope of this project is limited to the cell responsible for filling the empty boxes.

1.2.1 Physical Configuration

1.2.1.1 Robot arms

Robot arms are located at the start and end of the conveyor at a fixed position. The robot arms will help to pick up and drop filled boxes from one work-cell to the next work-cell. The cell which this report looks into is resposbile for controlling the arm at the start of the conveyor. However, to control the robot arm at the end of the conveyor. An internal variable has been used which will be sent to the next cell, indicating that the box is ready to be picked up.

1.2.1.2Hopper

The hopper is filled with polystyrene chips to fill the box. A solenoid is used to open and close the hopper to dispense the polystyrene material.

1.2.1.3 Sensors

Proximity sensors are used to detect when a box has arrived at the hopper or EOC. While a level sensor has been implemented to indicate when a box is full and can move onto the next stage of processing.

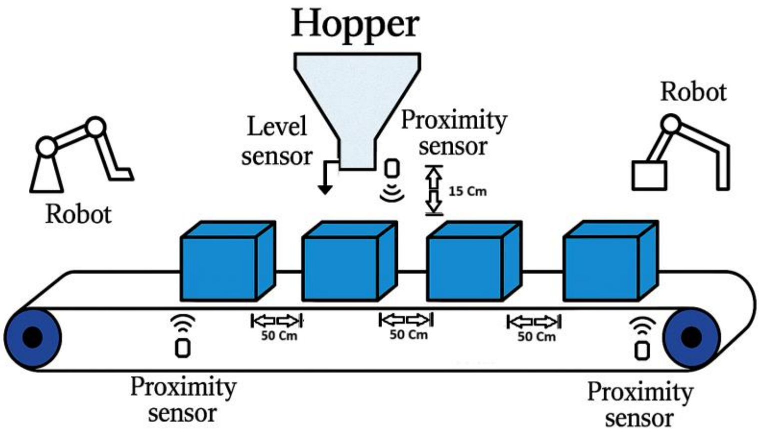

Figure 1. High-level overview of conveyor system

1.3 Assumptions

Throughout the design process the team has made numerous assumptions which were required to refine the projects scope and ensure the code works as intended. The various assumptions are outlined below and explained in detail. They apply to all aspects of the design

1) **Box Capacity and Size**

   All boxes are 1 m^3. A total of 4 boxes may fit on the conveyor at one time. Boxes are placed on the conveyor such that there is a 50 cm distance between each box. The number of boxes on the conveyor can be modified within the ST code to meet users' industry requirements.

2) **Sensors**

   The proximity sensor must be placed directly under the hopper to register a "HIGH" signal. It is expected that the sensor will only provide an accurate signal for filling when the box is positioned correctly in the centre. Once the box is fully filled, the acoustic level sensor will indicate a "HIGH" value. It is thought that the level sensor is designed to emit only a HIGH signal after a long period of detection of objects. This function prevents temporary interruptions from being accidentally activated by the sensor, such as pieces of polystyrene falling through the air. This calibration allows accurate detection of the filling level by eliminating false readings.

3) **Conveyor Belt**

   The belt only moves in the forward direction and operates without manual intervention. The total length will be according to the total number of boxes that can fit on the conveyor. Here, we take 10 boxes with 5 cm spacing between each box. Therefore,

$$Conveyor\ Length\ =\ (Box\ Length \cdot Total\ boxes)\ +\ \big(Spacing\ \cdot (Total\ Boxes\ -1)\big) \qquad (1)$$

$$Conveyor\ Length = 5.5\ metres$$

The conveyor runs at a speed such that it will take less than 10 seconds for an empty box to travel from the start of the conveyor to the proximity sensor.

4) Power Supply

All sensors, actuators and PLCs are powered with a 24V DC power supply. The conveyor motor is powered by a 240 VAC supply. Energy efficiency is not considered in the design of the conveyor system and the power consumption in all states of the system are within a practically safe and suitable range.

5) Hopper

The hopper uses a sliding gate which open horizontally to dispense the Styrofoam by relying on gravity. Within the model we assume that a box should completely fill within twenty seconds. However, this is dependent on the physical size of the hoppers gate and can be reconfigured by the used based on their hopper's specifications.

6) Robot Arms

The robot arms have two fixed positions – one for placing boxes and another for picking up boxes. It will only handle one box at a time. Robotic movements are assumed to finish each cycle within timed value of 4 seconds. However, this can be reconfigured by the user to meet specific factory specifications.

2. Theories and Frameworks

2.1 The Packaging Machine Language

Pack ML is an international standardised machine programming framework with the ability to operate on any automated machine [a]. The international standard defines machine/unit state types, machine/unit control modes, unit control mode management and State models, State descriptions, and transitions [b].

The benefit of a standardised framework includes ease of integration across machines from various vendors and a reduction in training costs and times. PackML describes a machine's condition. Pack ML includes 17 states, presented in ***Figure 2***, of which not all are required to be implemented. States can be described as waiting or acting.

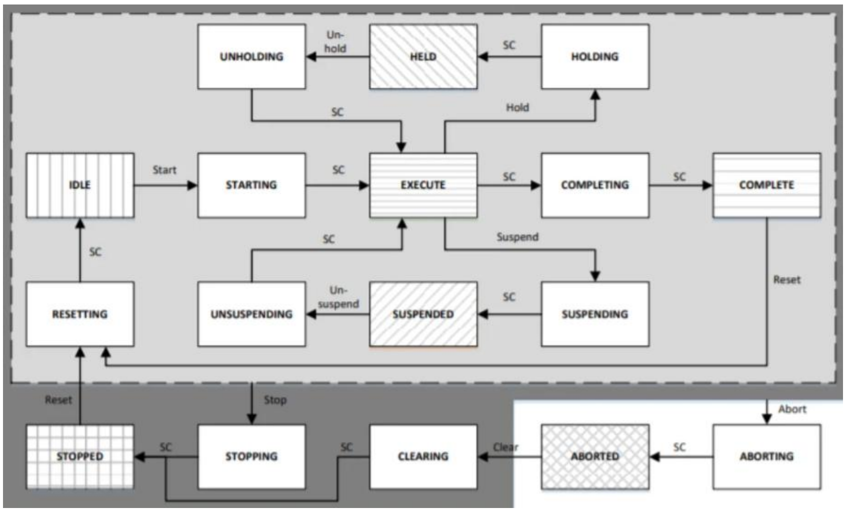

Figure 2. PackML State Diagram [1]

The seventeen PackML states are:

1. Stopped: A wait state. Reached after completion of Stopping or Clearing states. A reset command enables the exit from this state to Resetting.
2. Idle: A wait state. Occurs after the Resetting state. A start command is required to move into Starting state.
3. Starting: An acting state. Begun upon start command, a predecessor to Execute state.
4. Suspending: An acting state. Begins when external conditions do not permit the machine to continue production during Execute state, preceding Suspended.
5. Suspended: A wait state. Exited to Unsuspending upon clearing of external conditions prohibiting execution.
6. Unsuspending: An acting state. Any required actions to begin execution are initiated. Upon completion of this initiation, machine moved back to Execute.
7. Execute: An acting state where the machine carries out the designated task or process.
8. Stopping: An acting state. Entered with a stop command with machine previously in any state. Stopped follow this state.
9. Aborting: An acting state that begins from an abort command. This state occurs prior to the Aborted state.
10. Aborted: A wait state following Aborting. Leaves Aborted with a clear command to Clearing state.
11. Holding: An acting state triggered when internal machine conditions prevent the machine to continue executing. Holding can be initiated automatically or by an operator. Upon completion, the system moves to Held state.
12. Held: A wait state after Holding, prior to Unholding upon the clearing of internal conditions impeding production
13. Unholding: An acting state, where the necessary system activation actions take place prior to Execute.
14. Completing: An acting state following Execute state when the machines task is finished.
15. Complete: A wait state after Completing, indicating machine’s performed task is complete. Exit of this state occurs with a reset command which will move the system to Resetting
16. Resetting: An acting state triggered by a reset command from Stopped state or Complete State. Active alarms are cleared and safety systems are activated. Upon completion, the system transitions to Idle.
17. Clearing: An acting state. Begins from a clear command from Aborted state, any alarms triggered from abort are cleared. Prior to Stopped state.

3. Program Flowchart

The flowchart depicted below details the program logic flow to control the filling of boxes on the conveyor. The flowchart includes the 17 PackML states as described in Section 2.1, hence the program begins in the Idle state.

The Idle state transitions to the Starting state through a start command. In Starting, the robot arm before the conveyor is triggered to place a box if there is currently no box on conveyor. This state completes when the conveyor run is triggered. The Execute state begins with checking for the proximity sensor to indicate a box has arrived under the hopper. Upon arrival, the conveyor is stopped, and the hopper solenoid will open to fill box. A level sensor will trigger when the box is full, closing the hopper solenoid. At this stage a new box should be placed by the robot arm at the start of the conveyor. The conveyor will then begin to move again until the box reaches the end conveyor sensor where a robot arm will pick up the box. There are various cases within this state which can cause the system to transition to either a Holding or Suspending state. For example, if the level sensor has not triggered but the timer that starts when the hopper solenoid opens and has a preset value equivalent to the time to fill a box has been completed, the system will move to Suspending. The system will only move to Completing when the number of boxes counted in the output is equal to the desired number of filled boxes. Following the completing state, the system transitions to Complete and remains in this state until a reset command.

The Suspending state stops the conveyor, closes the hopper solenoid and stops both the pre and post conveyor robot arms. The Unsuspending state checks the suspend case, i.e. the reason the system was put into Suspending, depending on the case the process differs slightly to avoid redundant disabling of functionality of the system.

The Holding state is entered through a hold command, the state is very similar to the Suspending state where the conveyor is stopped, hopper closed and both robot arms disabled. Following this, the system is moved to the Held state. The Held state waits for an unhold command. Unholding state transitions the system back to Execute.

The Resetting state, triggered by the reset button or following the Stopped state, simply resets all internal bits and timers before moving to Idle.

The abort command and stop command will transition the system to Aborting or Stopping respectively, regardless of the current state.

The Aborting state stops the conveyor, closes the hopper and stops both robot arms before moving to Aborted which swiftly transitions to Clearing. Any power supply faults triggered are cleared before the system moves to Stopped.

The Stopping state also stops the conveyor and sets the hopper solenoid to false as well as both robot arms before moving to Stopped. From Stopped the system moves to Resetting.

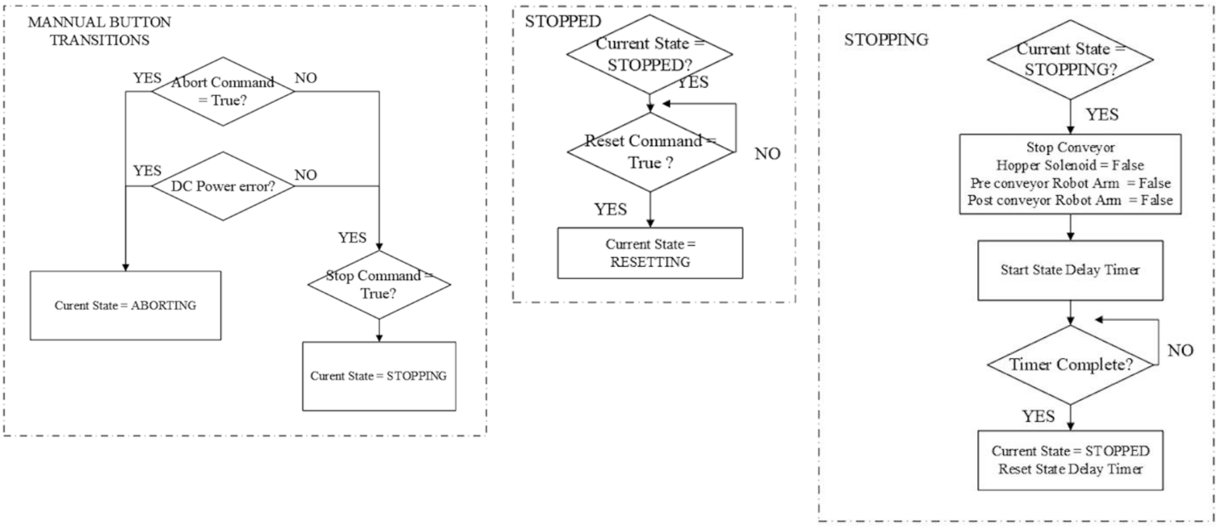

Figure 3. Stopped, Stoppping States Flow Chart

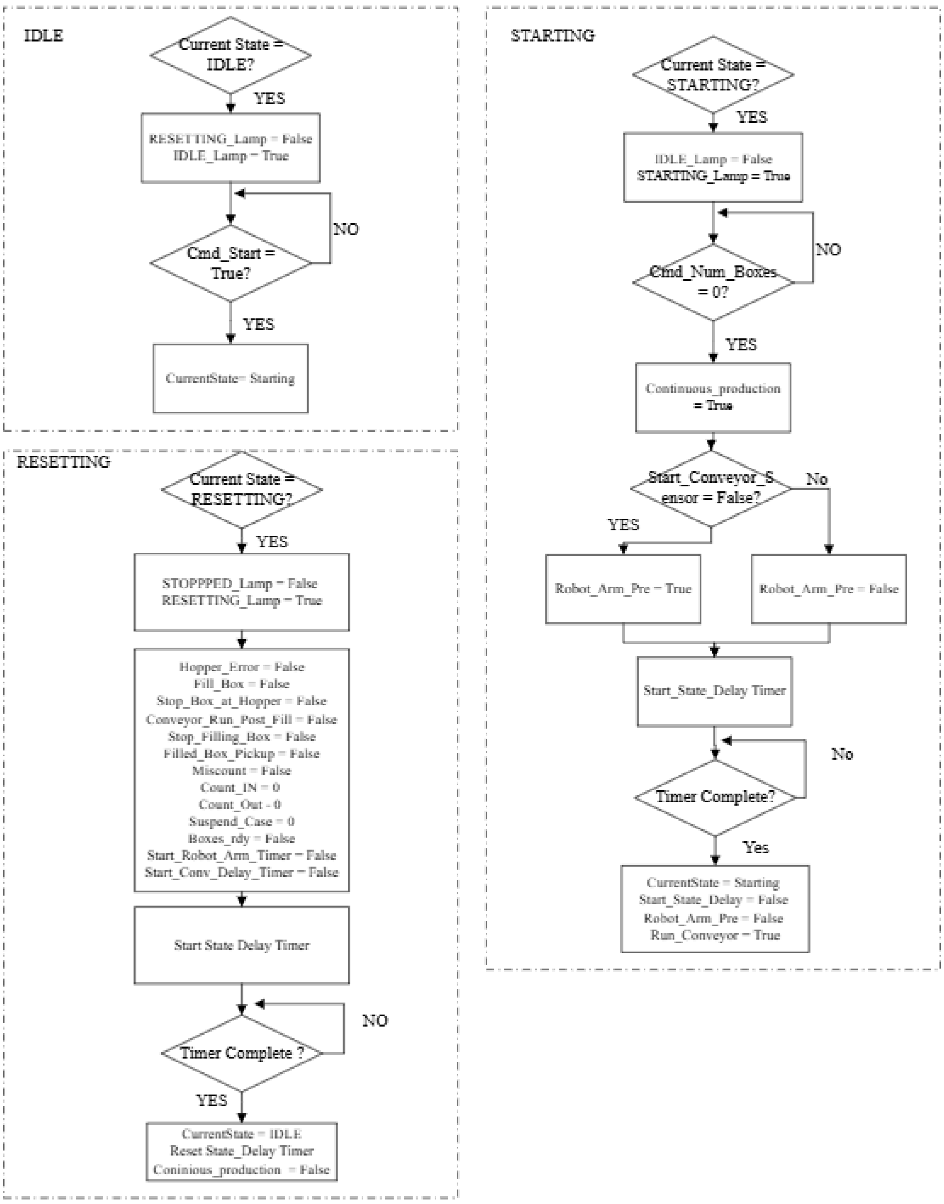

Figure 4. Idle, Starting, Resetting States Flow Chart

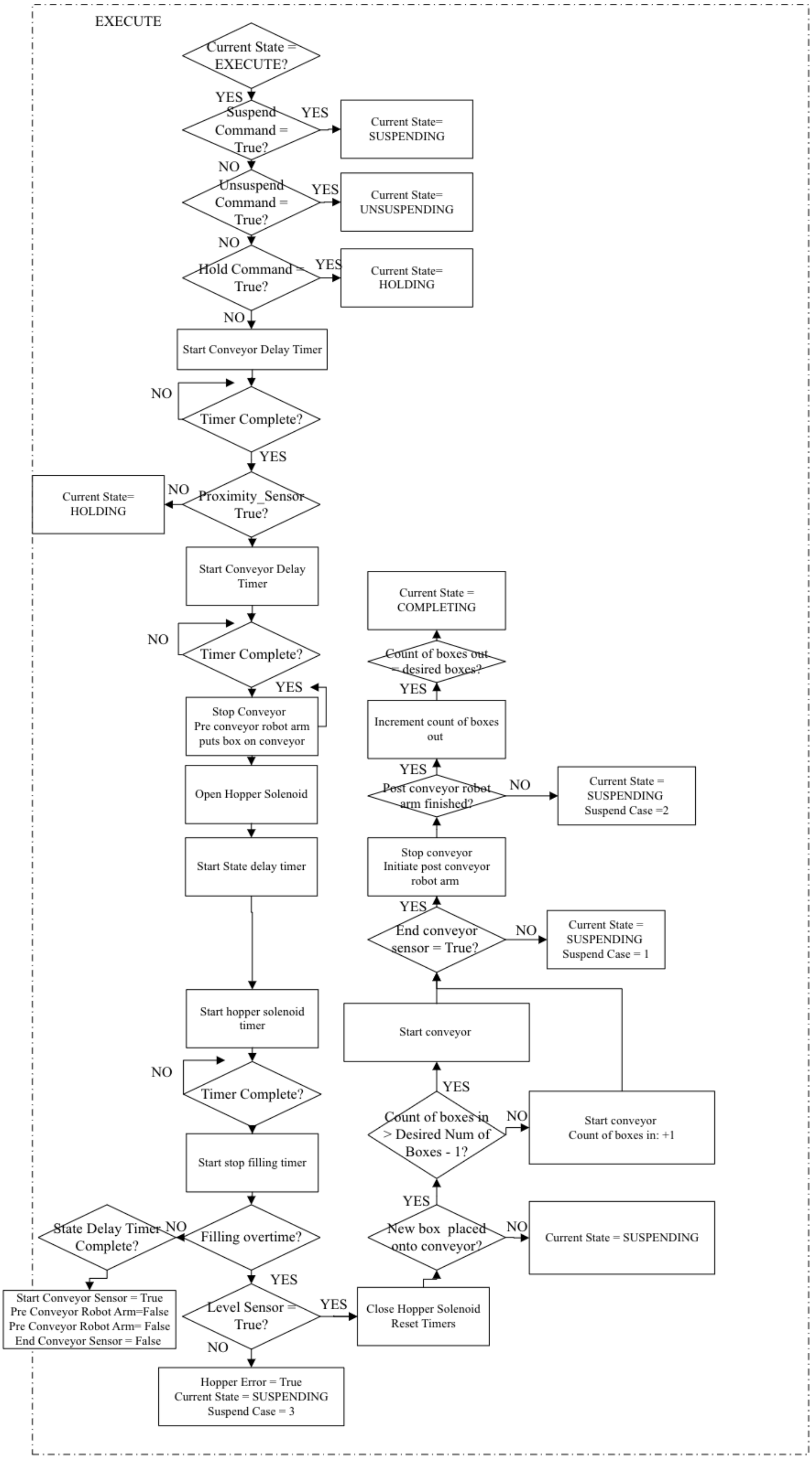

Figure 5. Execute State Flow Chart

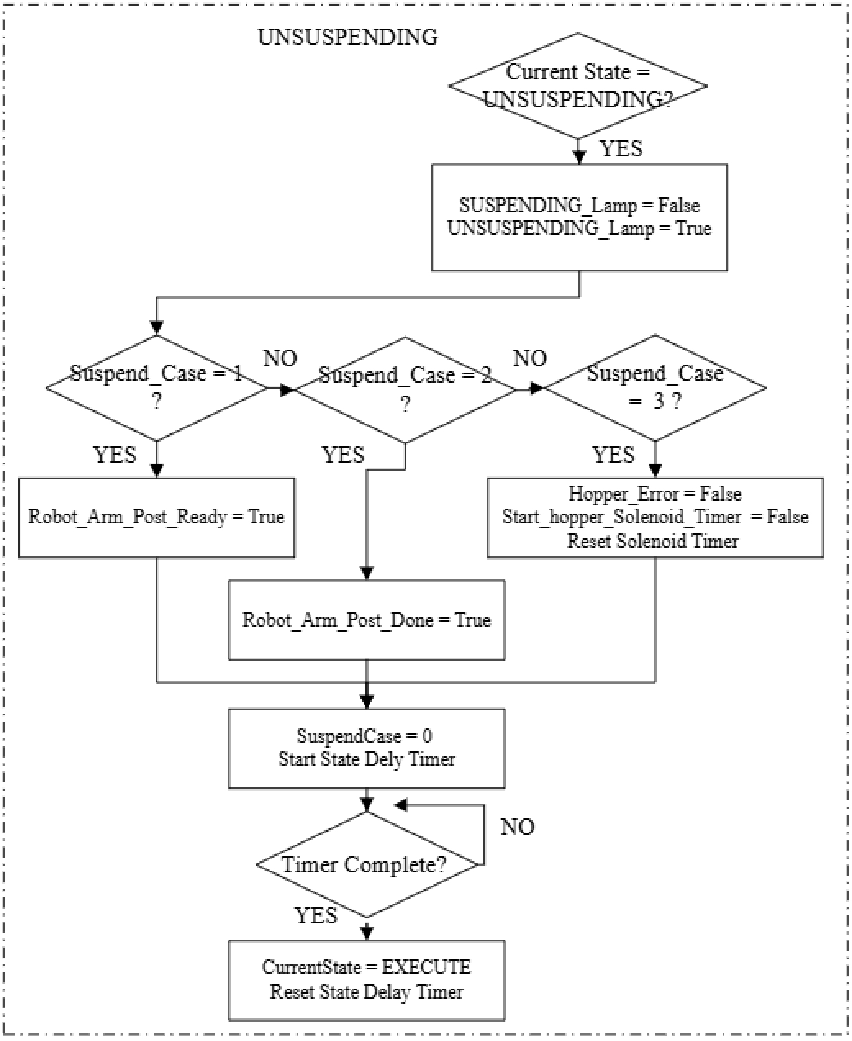

Figure 6. Unsuspeding State Flow Chart

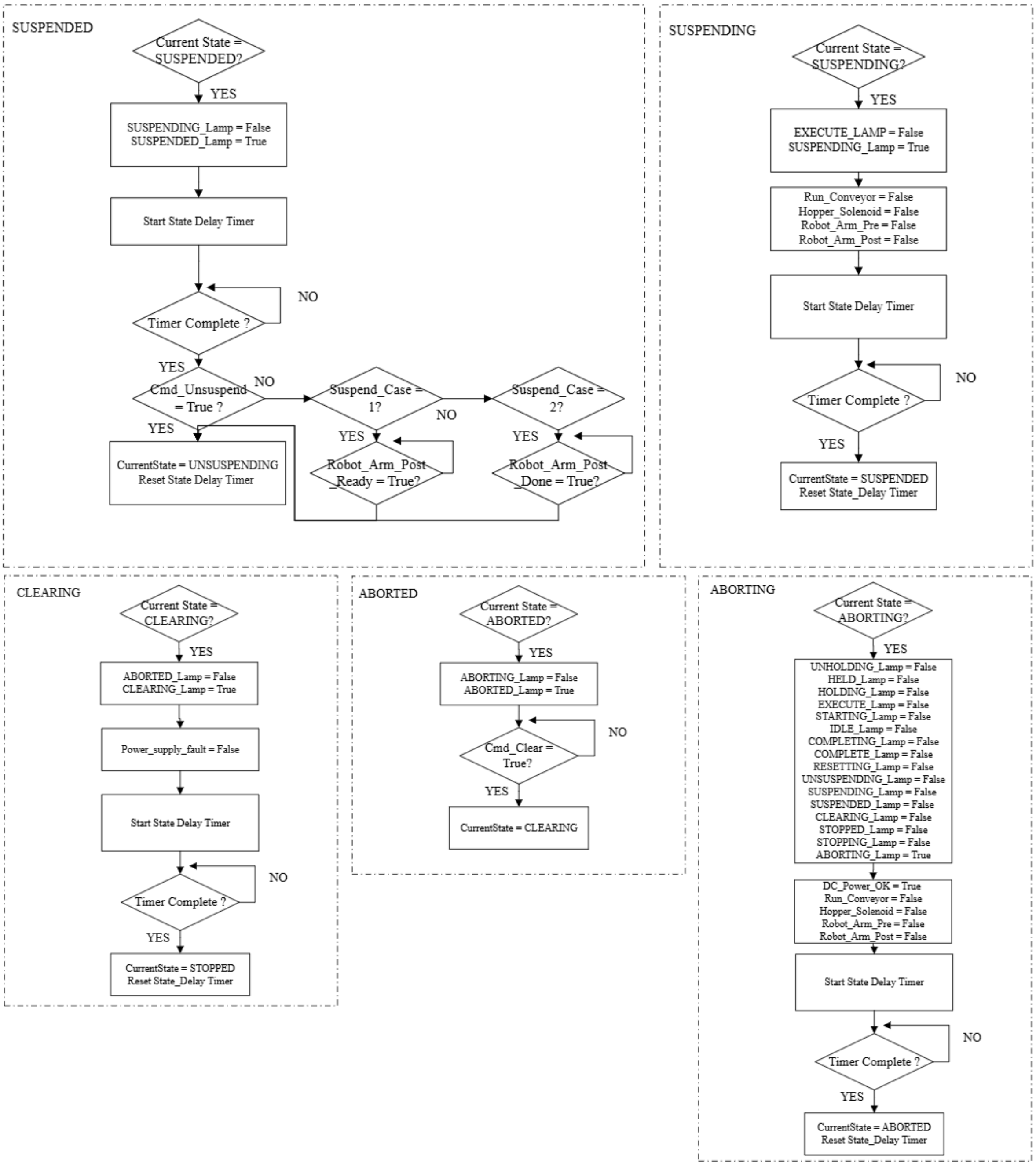

Figure 7. Suspending, Suspending, Clearing, Aborted, Aborting State Flow Chart

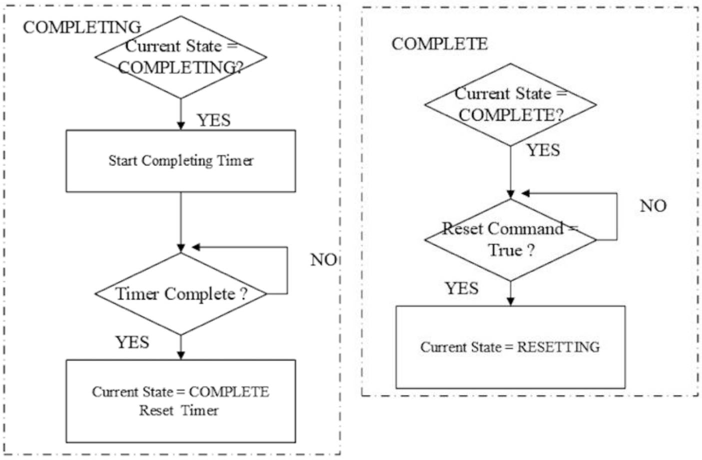

Figure 8. Completing & Complete, State Flow Chart

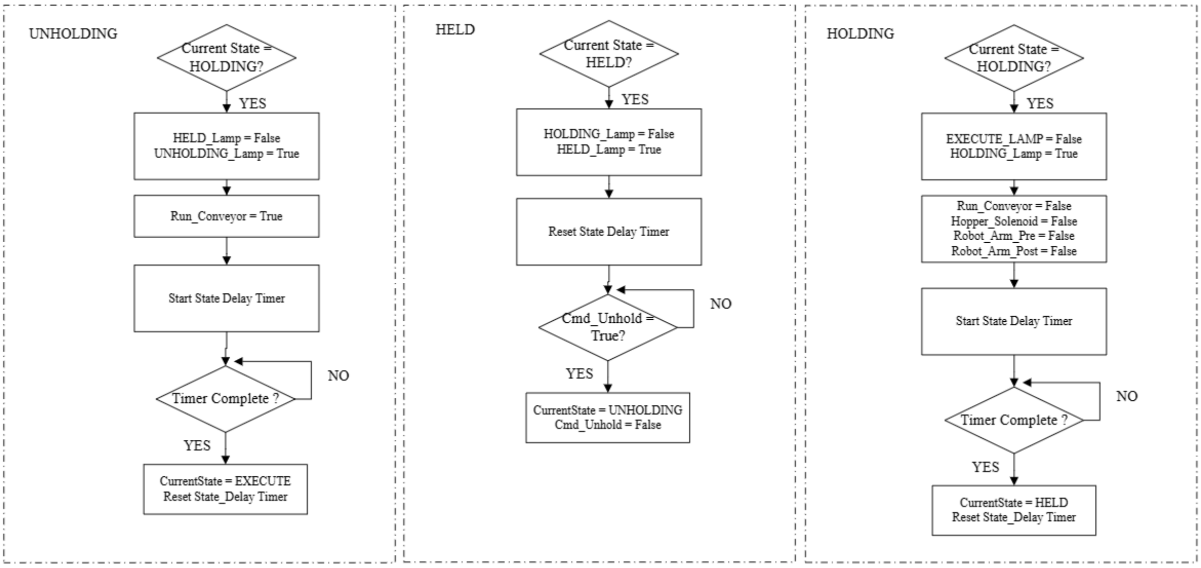

Figure 9. Unholding, Held, Holding State Flow Chart

Program Inputs and Outputs / Variables

Table 1 details the input variables used in the CodeSys program that correspond to the devices used in Figure 1. Additional Command (Cmd) variables correspond to command buttons within the system.

Table 1: Input Variables

Name	Description	
Proximity_Sensor	True: If box under hopper	False: otherwise
Level_Sensor	True: If box full	False: otherwise
End_Conveyor_Sensor	True: If box at end of conveyor	False: otherwise
Robot_Arm_Post_Ready	True: ready (signal from next cell)	False: otherwise
Robot_Arm_Post_Done	True: pick up done (signal from next cell)	False: otherwise
DC_Power_OK	True: If DC power to system is stable	False: otherwise
Start_Conveyor_Sensor	True: If start command active	False: otherwise
Cmd_Abort	True: If Abort button pressed	False: Otherwise
Cmd_Start	True: If Start button pressed	False: Otherwise
Cmd_Stop	True: If Stop button pressed	
Cmd_Reset	True: If Reset button pressed	False: Otherwise
Cmd_Hold	True: If Hold button pressed	False: Otherwise
Cmd_Unhold	True: If Unhold button pressed	False: Otherwise
Cmd_Suspend	True: If Suspend button pressed	False: Otherwise
Cmd_Unsuspend	True: If Unsuspend button pressed	False: Otherwise
Cmd_Clear	True: If Clear button pressed	False: Otherwise
Cmd_Num_Boxes	0: Continuous Production Mode	>0: Number of boxes to produce

Table 2 outlines the variables that control the system’s outputs. This includes the conveyor, hopper solenoid and robot arms.

Table 2. Output Variables

Name	Description	
Run_Conveyor	True: Conveyer running	False: otherwise
Hopper_Solenoid	True: Hopper solenoid open, filling box	False: otherwise
Robot_Arm_Pre	True: Robot Arm at start of process places box on conveyor	False: otherwise
Robot_Arm_Post	True: Robot Arm at end of process picks up box from conveyor	False: otherwise

Table 3. Timer Bits

Name	Description
Conv_Delay_Timer	Timer in Held state, conveyor delay or jam timer before transitioning to Unholding state
Hopper_Solenoid_Timer	Timer in Execute state; Timer to record box filling time
State_Delay	Timer in Stopping state for visualisation lamp transitions
State_Delay_2	Timer to fill box, the timer hoper solenoid should be open and filling the box in Execute state.
State_ Delay_3	Timer after box is full, to ensure the any remaining filling material falls from hopper falls into box whilst in Execute state.

The exhaustive list of internal variables used in the program, including those used for visualisation, are outlined in Appendix A.

4. Structured Text Code

Within this section of the report the PackML states within the ST are broken down and explained in significant detail. Each section will explain a different state and how the code will rely on the various sensors, to make decisions and act accordingly to reach the desired functionality of the overall system.

4.1 State Transitions

The ‘CurrentState’ variable has been used to implement a State Machine within the program. Each state has been assigned as an integer value as outlined in Figure 10. The variable allows the system to easily move between segmented parts of the code and operate in a safe and reliable manner.

```
CurrentState    : INT := 0;
IDLE            : INT := 0;
STARTING        : INT := 1;
EXECUTE         : INT := 2;
STOPPING        : INT := 3;
ABORTING        : INT := 4;
ABORTED         : INT := 5;
RESETTING       : INT := 6;
HOLDING         : INT := 7;
HELD            : INT := 8;
UNHOLDING       : INT := 9;
SUSPENDING      : INT := 10;
SUSPENDED       : INT := 11;
UNSUSPENDING    : INT := 12;
CLEARING        : INT := 13;
STOPPED         : INT := 14;
COMPLETING      : INT := 15;
COMPLETE        : INT := 16;
```

Figure 10. State Transition Variables

An IF Loop is used for each PackML state, which is executed when the ‘CurrentState’ is assigned the PackML State’s corresponding integer.

Due to the high speed of program scans, state transitions that do not require a change in the input conditions by the user result in the lamp visualisations not visibly triggering. This occurs during the following transitions:

Idle	→ Starting
Starting	→ Execute
Aborting	→ Aborted
Holding	→ Held
Suspending	→ Suspended
Clearing	→ Stopped
Completing	→ Complete

As a result, a 5-second timer is implemented before these state transitions occur. The general structure of this timer is illustrated in Figure 11, which involves enabling the ‘State_Delay’ timer with ‘Start_State_Delay’, transitioning to the next state when the timer is complete with ‘State_DN’, and resetting the timer before entering the next state by setting ‘Start_State_Delay’ back to FALSE. The same format is used for all state transitions mentioned above to have the visualisation run successfully and be a more accurate representation of a real-life system.

```
        // -----VISUALISATION + State transition------
        Start_State_Delay := TRUE;
        State_Delay(IN :=Start_State_Delay, PT := State_Delay_PT);
        State_Delay_DN := State_Delay.Q;
        IF State_Delay_DN = TRUE THEN
            CurrentState := 2; //Execute State
            Start_State_Delay := FALSE;
            State_Delay(IN :=Start_State_Delay, PT := State_Delay_PT);
            Robot_Arm_Pre := FALSE;
            Run_Conveyor := TRUE; // Conveyor is running
        END_IF
    END_IF
    //xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Enable and initialise timer

Next State

Reset timer

Figure 11. CodeSys Extract – Example State Transition Timer

4.2 Idle State

The program commences with CurrentState in the idle state. When in the Idle state, the Idle lamp on the visualisation is turned on, the Resettting lamp is specified to turn off as the CurrentState can transition to the Idle state from Resetting State, as shown in Figure 12. The program will not leave the Idle state to move to Starting into input Cmd_Start is True.

```
    // --- IDLE ---------------------------------------------------
    IF CurrentState = IDLE THEN

        // --- FOR VISUALISATION ----
        RESETTING_Lamp := FALSE;
        IDLE_Lamp := TRUE;
        // -------------------------

        IF Cmd_Start THEN
            CurrentState := 1; //Starting
            END_IF
        END_IF

    //xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Figure 12. CodeSys Extract – Idle State

4.3 Starting State

Upon CurrentState being equal to Starting state, the Idle visualisation lamp is turned off and the Starting lamp is switched on in Lines 41-42. The product mode can be defined by user to either continuous filling of boxes until a command (Cmd_Num_Boxes = 0) or a predefined number of boxes to fill (Cmd_Num_Boxes = predefined number). The default case is continuous filling. The Starting state checks the Start_Conveyor_Sensor, to determine if a box is on the conveyor. If a box is on the conveyor, Start_Conveyor_Sensor is True and the output controlling the robot arm at the start of the conveyor is not iniitalised. If there is no box on the conveyor, hence Start_Conveyor_Sensor is False, the Robot_Arm_Pre is triggered.

```
37    // --- STARTING ------------------------------------------------
38    IF CurrentState = STARTING THEN
39
40        // --- FOR VISUALISATION ----
41        IDLE_Lamp := FALSE;
42        STARTING_Lamp := TRUE;
43        // -------------------------
44
45        //Decide product mode: X Number of Boxes (Cmd_Num_Boxes > 0)
46        // OR Continuous Production (Cmd_Num_Boxes = 0)
47        IF Cmd_Num_Boxes = 0 THEN
48            Continuous_production := TRUE;
49        END_IF
50
51        // check if box already @ start of the conveyor, doesnt put box if there is
52        IF Start_Conveyor_Sensor = FALSE THEN
53            Robot_Arm_Pre := TRUE;   // places box on conveyor
54        ELSE
55            Robot_Arm_Pre := FALSE;  //doesnt place box on conveyor
56        END_IF
```

Figure 13. CodeSys Extract – Starting State

Following this the state delay timer is started, upon completion the CurrentState transitions to the next state: Execute. The end of the timer also triggers the turning off of Robot_Arm_Pre and starting the Run_Conveyor.

```
        // -----VISUALISATION + State transition------
        Start_State_Delay := TRUE;
        State_Delay(IN :=Start_State_Delay, PT := State_Delay_PT);
        State_Delay_DN := State_Delay.Q;
        IF State_Delay_DN = TRUE THEN
            CurrentState := 2; //Execute State
            Start_State_Delay := FALSE;
            State_Delay(IN :=Start_State_Delay, PT := State_Delay_PT);
            Robot_Arm_Pre := FALSE;
            Run_Conveyor := TRUE; // Conveyor is running
        END_IF
    END_IF
    //xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Figure 14. CodeSys Extract - State Transition Timer

4.4 Execute State

The execute PackML state is the primary state the system will be operating in to process and fill the boxes with Styrofoam. It will be responsible for telling the robot arm at the start of the conveyor to place a box on the conveyor. Detecting when a box arrives underneath the hopper and subsequently opening and closing the hopper solenoid. Before moving the box to the end of the conveyor where it will signal the next work cell to pick up the filled box. Numerous sensors and conditional statements have been utilised within this stage to achieve the desired functionality

4.4.1 State Initialisation & Cmd Checks

As seen from Figure 15 the Execute sections code will continuously check if there is a transition into the execute state. Once this occurs the correct lamps will be initialised to demonstrate to the user the program is now in the execute stage. Once this has occurred, checks will be conducted to ensure that command buttons such as the Çmd_Syspend and Cmd_Hold aren't pressed. If they are, then the program will transition from the execute state to either the suspending, unsuspended or holding state once the state transition timer completes. An additional check has been placed from line 89 in Figure 15 which will trigger a Holding state fault.

The code will check for a potential fault and will trigger if no box is detected by the proximity sensor underneath the hopper after a ten second period. This assumes that it takes less than ten seconds for the box to move from the start of the conveyor to underneath the hopper. If the timer completes before a box arrives at the hopper then it is assumed that the box has either fallen off or gotten jammed. Resulting in the fault state to occur.

```
    // --- EXECUTE ---------------------------------------------------
    IF CurrentState = EXECUTE THEN
        // --- FOR VISUALISATION ----
        UNHOLDING_Lamp := FALSE;
        STARTING_Lamp := FALSE;
        UNSUSPENDING_Lamp := FALSE;
        EXECUTE_Lamp := TRUE;
        //---------------------------

    IF Cmd_Suspend THEN
        CurrentState := 10; //Suspending state
    ELSIF Cmd_Unsuspend THEN
        CurrentState := 12; //Unsuspend state
    ELSIF Cmd_Hold THEN
            CurrentState := 7; //holding state
    END_IF

    //HOLDING CASE: If Box isnt detected by proxy sensor within 10s -> Holding
    Start_Conv_Delay_Timer := TRUE;
    Conv_Delay_Timer(IN :=Start_Conv_Delay_Timer, PT :=Conv_Delay_Timer_PT); // Assume boxes arrive @ hopper <7s
    TimerDone_Conv_Delay_Timer := Conv_Delay_Timer.Q;

    IF TimerDone_Conv_Delay_Timer AND NOT Proximity_Sensor AND Run_Conveyor AND Continuous_production THEN
        CurrentState := 7; // Holding state
    ELSIF Proximity_Sensor THEN
        Start_Conv_Delay_Timer := FALSE;
        Conv_Delay_Timer(IN :=Start_Conv_Delay_Timer, PT :=Conv_Delay_Timer_PT);
        TimerDone_Conv_Delay_Timer := FALSE;
    END_IF;
```

Figure 15. CodeSys Extract - Execute State: State Initialisation & Cmd Check

4.4.2 Stop Box Underneath the Hopper

Once the state has been initialised and a box is moving along the conveyor. The next stage is to ensure that it stops in the correct location underneath the hopper ready to be filled. The code from Figure 18 is designed to achieve this functionality. Line 103 will continuously check that the conveyor is running and wait for the proximity sensor under the hopper to be triggered. Once triggered, the conveyor will stop and the robot arm at the start of the conveyor is instructed to place another box on the conveyor. Additionally, the ten second timer which was discussed in section 4.4.1 is turned off as the box has arrived safely at the hopper (Seen from line 107).

```
102     // Stop box at hopper
103     Stop_Box_at_Hopper := Run_Conveyor AND Proximity_Sensor;
104     IF Stop_Box_at_Hopper = TRUE THEN
105
106         // Stop Holding State Timer
107         Start_Conv_Delay_Timer := FALSE;
108         Conv_Delay_Timer(IN :=Start_Conv_Delay_Timer, PT :=Conv_Delay_Timer_PT);
109
110         Run_Conveyor := FALSE; // Turn off conveyor
111         Robot_Arm_Pre := TRUE; // puts box onto the conveyor
112     END_IF
```

Figure 16. CodeSys Extract - Execute State: Stop Box Underneath Hopper

4.4.3 Fill Box Logic

Now that the box has come to a stop underneath the hopper, the next step in the process is to fill the box with the styrofoam. This section of code from Figure 20 requires specific conditions to be met before the hopper solenoid will open.

If Proximity_Sensor is true while Level_Sensor is false and Run_Conveyor is false, then the system can confirm that the box is under the hopper and ready to be filled, therefore the hopper solenoid will be activated and open. Once the solenoid has opened the code will simultaneously initiate the timer Start_State_Delay_2. This was incorporated into the design to ensure that the robot arm at the start of the conveyor finishes its task of placing a box on the conveyor before the conveyor can be started. However, it is assumed that it takes significantly less time to place a box on the conveyor than it does to fill a box at the hopper. Therefore, if the level sensor is triggered before the box is placed on the hopper the system will move into a fault state, as seen from the code in Figure 18.

```
// Fill box logic
 Fill_Box := Proximity_Sensor AND (NOT Run_Conveyor) AND (NOT Level_Sensor);
 IF Fill_Box = TRUE THEN
    Hopper_Solenoid := TRUE;

    Start_State_Delay_2 := TRUE;
    State_Delay_2(IN :=Start_State_Delay_2, PT := State_Delay_2_PT);
    State_Delay_DN_2 := State_Delay_2.Q;
```

Figure 17. CodeSys Extract - Execute State: Filling Box Logic Check

Another timer has been implemented to identify if the box is taking longer than expected to fill. Start_Hopper_Solenoid_Timer is used record the time taken to fill the box. If the hopper is open and the conveyor is stopped, but the Level_Sensor is not triggered within twenty seconds, we assume that a hopper malfunction has occurred. In that case, the system enters the Hopper Error state, sets CurrentState to ten to enter the suspend state, and assigns Suspend_Case the value 3 to help identify the cause of the suspension. If there is no timeout and the polystyrene chips fill the box normally, the timer will reset after filling.

```
//Timer to record filling time
Start_Hopper_Solenoid_Timer := TRUE;
Initalise_Stop_Filling_Timer(IN := Start_Hopper_Solenoid_Timer, PT := Solenoid_Timer_PT);
TimerDone_Solenoid_Timer := Initalise_Stop_Filling_Timer.Q;
IF Hopper_Solenoid = TRUE
      AND Initalise_Stop_Filling_Timer.Q = TRUE //filling over time
      AND NOT Level_Sensor
      AND NOT Run_Conveyor THEN
      Hopper_Error := TRUE;
      CurrentState := 10; //suspending state
      Suspend_Case := 3; // suspend case 3: Hopper error, filling overtime
END_IF
```

Figure 18. CodeSys Extract - Execute State: Suspend Case Checks

The code from Figure 19 will indicate when the task of putting another box onto the conveyor by the robot arm is complete. Variable State_Delay_DN_2 will return TRUE once the timer completes. Indicating that enough time has passed for the box to be placed on the conveyor. The length of the timer can be modified by the user in line 5 of the code to meet the factories specific requirements.

```
IF State_Delay_DN_2 = TRUE THEN
    Robot_Arm_Post          := FALSE;
    End_Conveyor_Sensor     := FALSE;
    Robot_Arm_Pre           := FALSE;
    Start_State_Delay_2     := FALSE;
    Boxes_rdy               := TRUE;
    Start_Conveyor_Sensor   := TRUE;
    State_Delay_2(IN :=Start_State_Delay_2, PT := State_Delay_2_PT);
END_IF
 ELSE
    Hopper_Solenoid := FALSE;
    State_Delay_2(IN :=Start_State_Delay_2, PT := State_Delay_2_PT);
    Start_State_Delay_2 := FALSE;
    State_Delay_DN_2    := FALSE;
END_IF
```

Figure 19. CodeSys Extract - Execute State: Robot Arm Complete Action

4.4.4 Continue Production After Box is Full

Now that the box is currently being filled, the code will need to continuously check the Level_Sensor to determine when the Hopper_Solenoid should close. The code from Figure 20 will check for this condition.

Line 153 defines the Stop_Filling_Box variable. When the conveyor is not running, the Proximity_sensor is true, and the Level_sensor is also true. It indicates that the box is positioned under the hopper and the chips have been filled. At this point, the system enters the stop-filling state and turns off the hopper solenoid.
In the Stop_Filling_Box state, when the Level_Sensor is true, the Start_Hopper_Solenoid_Timer will be reset. Otherwise, the timer may incorrectly time out when the next box arrives underneath the hopper. This would potentially trigger a hopper error and transition the system into the suspending state.

```
152     // Continue after full
153     Stop_Filling_Box := Proximity_Sensor AND (NOT Run_Conveyor) AND Level_Sensor;
154     IF Stop_Filling_Box  = TRUE THEN
155         Hopper_Solenoid     := FALSE;
156
157         //Reset the filling timer
158         IF Level_Sensor = TRUE THEN
159         Start_Hopper_Solenoid_Timer := FALSE; // once full, reset filling timer
160         Initalise_Stop_Filling_Timer(IN := Start_Hopper_Solenoid_Timer, PT := Solenoid_Timer_PT);
161         END_IF
```

Figure 20. CodeSys Extract - Execute State: Stop Filling Box Check

4.4.4.1 Production Target

While a box is being filled underneath the hopper, it provides an ideal opportunity to place another box at the start of the conveyor since the conveyor belt is stationary. The code from Figure 21 will determine if another box should be placed on the conveyor belt. This will be based on whether the system is currently operating in a continuous mode or if it seeks to reach a production quota and cease production.

In the filling process, we have Boxes_rdy to indicate if a new box is placed while filling the current box. Normally the new box is ready before the current box is full, which will make Boxes_rdy = TRUE. Since we have two production modes, we defined two separate conditions to prevent overproduction of boxes.

Line 163 corresponds to the limited production mode. It is triggered when the system enters the Stop_Filling_Box state and the current box is not the last one (Count_IN > (Cmd_Num_Boxes - 1)). In this case, the conveyor is restarted, the Boxes_rdy flag is reset, and State_Delay_DN_2 is cleared. This allows the conveyor to transport the filled box to the end while resetting the 'new box' status, preparing for the next box to enter the process.

We also set all relevant sensors (such as Proximity_Sensor, Level_Sensor, and Start_Conveyor_Sensor) to FALSE here purely for visualization simulation purposes — in reality, once the conveyor begins moving, the box is carried away, and these sensors would naturally return to FALSE.

Line 171 handles the continuous production mode, where the box quantity limit is removed. The rest of the operations are the same as in the limited production case.

```
        // Stops placing more boxes if its reached the desired # of box production
        IF Boxes_rdy = TRUE AND Stop_Filling_Box AND (NOT Continuous_production) AND (Count_IN > (Cmd_Num_Boxes - 1)) THEN
            Run_Conveyor            := TRUE;
            Proximity_Sensor        := FALSE;
            Level_Sensor            := FALSE;
            Boxes_rdy               := FALSE;
            Start_Conveyor_Sensor   := FALSE;
            State_Delay_DN_2        := FALSE;
        // Continous production
        ELSIF Boxes_rdy = TRUE AND Stop_Filling_Box THEN
            Run_Conveyor            := TRUE;
            Proximity_Sensor        := FALSE;
            Level_Sensor            := FALSE;
            Count_IN                := Count_IN + 1;
            Boxes_rdy               := FALSE;
            Start_Conveyor_Sensor   := FALSE;
            State_Delay_DN_2        := FALSE;
```

Figure 21. CodeSys Extract - Execute State: Production Target

However, if Boxes_rdy = FALSE, then that indicates that the new box was not successfully placed onto the conveyor. In this case, the conveyor should not start; otherwise, it would leave an empty spot, and the production process would not operate in an ideal state.

To handle this, we introduce a Start_State_Delay_3 timer here to wait for 5 seconds (this duration can be adjusted at the beginning of the program by modifying the State_Delay_3_PT variable), giving time for the signal indicating that a new box has been successfully placed. If Boxes_rdy becomes true during this period, it means the box has been placed, and the process continues with starting the conveyor and related steps.

However, if the timeout completes and the signal still hasn't arrived, it suggests a malfunction in the placing robot arm. The system then transitions into the suspending state and records it as Suspend_Case = 4.

```
        ELSE
            Start_State_Delay_3 := TRUE;
            State_Delay_3(IN :=Start_State_Delay_3, PT := State_Delay_3_PT);
            State_Delay_DN_3 := State_Delay_3.Q;
                IF State_Delay_DN_3 = TRUE THEN
                    IF Boxes_rdy = TRUE THEN
                        Run_Conveyor            := TRUE;
                        Proximity_Sensor        := FALSE;
                        Level_Sensor            := FALSE;
                        Count_IN                := Count_IN + 1;
                        Boxes_rdy               := FALSE;
                        Start_Conveyor_Sensor   := FALSE;
                        State_Delay_DN_2        := FALSE;
                        State_Delay_3(IN :=Start_State_Delay_3, PT := State_Delay_3_PT);
                    ELSE
                        CurrentState := 10;
                        Suspend_Case := 4;  //put on robot arm error
                    END_IF
                END_IF
        END_IF
    END_IF
```

Figure 22. CodeSys Extract - Execute State: Confirm Box Removed from EOC

4.4.5 EOC Box Pickup Logic

Lines 204–221 contain the code for picking up the filled box. When a box reaches the end of the conveyor, it triggers the End Conveyor Sensor (rising edge), indicating that the box needs to be transferred to the next cell, and enter the 'Fill Box Pickup' state. If the robotic arm of the next cell is ready, we receive the signal Robot_Arm_Post_Ready from next cell. Then we set Robot_Arm_Post to TRUE to notify the next cell that it can start picking up the box. Once the box has been successfully transferred, a Robot_Arm_Post_Done signal will be sent by next cell. Then, we increase the Count_Out value by 1 which keeps track of the number of boxes dispatched and exit the 'Fill Box Pickup' state. If the robotic arm is not ready or the pickup is not complete, the system enters a suspend state and records the corresponding Suspend_Case.

Lines 224–226 handle restarting the conveyor after the pickup process has been completed. If the box is picked up successfully, the box departure will create a falling edge of the End Conveyor Sensor, and that’s what line 224 indicates. Then the conveyor will run again when the pick-up is done.

The special thing is, we use the variable End_Conveyor_Sensor_Old to store the status of End_Conveyor_Sensor from the previous scan cycle. The pickup state is only triggered when the previous sensor value was FALSE and the current value is TRUE — this captures the rising edge of the sensor. This approach ensures that Count_Out only increase once per actual box, preventing it from being repeatedly incremented due to the program scanning the sensor continuously while it remains high.

```
202         // Box pickup logic
203             //Pick up when end converyor rising edge
204             IF End_Conveyor_Sensor = TRUE AND End_Conveyor_Sensor_Old = FALSE THEN //only record rising edge
205                 Filled_Box_Pickup := TRUE;
206                 Run_Conveyor := FALSE;
207                 IF Robot_Arm_Post_Ready = TRUE THEN
208                     Robot_Arm_Post := TRUE;
209                 ELSE
210                 CurrentState := 10; //suspending state
211                 Suspend_Case := 1; // suspend case 1: robot arm not ready to pick up
212                 END_IF
213                 IF Robot_Arm_Post_Done = TRUE THEN
214                     Count_Out := Count_Out +1;
215                     Filled_Box_Pickup := FALSE;
216                     Robot_Arm_Post := FALSE;
217                 ELSE
218                     CurrentState := 10; //suspending state
219                     Suspend_Case := 2; // suspend case 2: pick up not done so don't start the conveyor
220                 END_IF
221             END_IF
222
223             //Run the conveyor again when end conveyor falling edge
224             IF End_Conveyor_Sensor = FALSE AND End_Conveyor_Sensor_Old = TRUE AND NOT Fill_Box AND Robot_Arm_Post_Done THEN //only record rising edge
225                 Run_Conveyor := TRUE;
226             END_IF
```

Figure 23. CodeSys Extract - Execute State: Confirm Box Removed from EOC

4.4.5 EOC Box Pickup Logic

If we are not in continuously producing mode then the Exectue state needs to check how many boxes have been produced. Once the required number of boxes has been produced. The system then needs to transition to the Completing State. This is achieved by the snippet of code outlined in Figure 24.

```
        // Transition to COMPLETING
        IF (NOT Continuous_production) AND Count_Out = Cmd_Num_Boxes THEN // If production is complete (except continuous case)
            CurrentState := 15; // Transition to COMPLETING
        END_IF
    END_IF
```

Figure 24. CodeSys Extract - Execute State: Transition to Completing State

4.5 Stopping State

For various reasons the user may press the stop button, represented by the Cmd_Stop variable, to halt all current operations and force the system into the stopping state. This is presented in Figure 25. As the Stopping state can transition from any state except the Aborting states, as illustrated in Figure 2, the ELSIF statement in Line 17 has been written at the start of the program to take precedence over all other state transitions. However, the Cmd_Abort takes precedence over the Cmd_Stop.

```
12  //Manual buttons taking precedence over all/majority of states
13
14      IF Cmd_Abort OR (NOT DC_Power_OK) THEN
15          Power_supply_fault := NOT DC_Power_OK;
16          CurrentState := 4; // Aborting State
17      ELSIF Cmd_Stop THEN
18          CurrentState := 3; //stopping state
19      END_IF
```

Figure 25. CodeSys Extract – Stop and Abort transitions

Once this occurs the segment of code from Figure ***26*** will be activated. The code will activate the stopping state lamp while also stopping the conveyor, closing the hopper solenoid and preventing a box from being placed onto and taken off the conveyor belt. Once these actions are complete, the State_Delay timer will be activated and after five seconds the system will move into the stopped state. The Stopped state functionality is outlined in section 4.6 of this report.

```
// --- STOPPING -------------------------------------------------------
IF CurrentState = STOPPING THEN
    // --- FOR VISUALISATION ----
    UNHOLDING_Lamp      := FALSE;
    HELD_Lamp           := FALSE;
    HOLDING_Lamp        := FALSE;
    EXECUTE_Lamp        := FALSE;
    STARTING_Lamp       := FALSE;
    IDLE_Lamp           := FALSE;
    COMPLETING_Lamp     := FALSE;
    COMPLETE_Lamp       := FALSE;
    RESETTING_Lamp      := FALSE;
    UNSUSPENDING_Lamp   := FALSE;
    SUSPENDED_Lamp      := FALSE;
    SUSPENDING_Lamp     := FALSE;
    STOPPED_Lamp        := FALSE;
    STOPPING_Lamp       := TRUE;
    //--------------------------------

    Run_Conveyor    := FALSE;
    Hopper_Solenoid := FALSE;
    Robot_Arm_Pre   := FALSE;
    Robot_Arm_Post  := FALSE;

    //Timer for lamp transitions
    Start_State_Delay := TRUE;
    State_Delay(IN :=Start_State_Delay, PT := State_Delay_PT);
    State_Delay_DN := State_Delay.Q;
    IF State_Delay_DN = TRUE THEN
        CurrentState := STOPPED;
        Start_State_Delay := FALSE;
        State_Delay(IN :=Start_State_Delay, PT := State_Delay_PT);
    END_IF
END_IF
//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Figure 26. CodeSys Extract - Stopping State: State Initialisation & Forcing Variables

4.6 Stopped State

The Stopped state is a transitionary state between the Stopping and Resetting State. Figure 27 portrays the ST code for the Stopped state. The code will activate the states corresponding Stopped_Lamp for the user and wait until the Cmd_Rest button is pressed. The system will stay in the stopped stay until the Reset button is pressed by the user, once pressed. The system will instantaneously move into the resetting state.

```
// --- STOPPED -----------------------------------------------------
IF CurrentState = STOPPED THEN
    // --- FOR VISUALISATION ----
    CLEARING_Lamp := FALSE;
    STOPPING_Lamp   := FALSE;
    STOPPED_Lamp    := TRUE;
    //--------------------------

    IF Cmd_Reset = TRUE THEN
        CurrentState := RESETTING;
    END_IF
END_IF
//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Figure 27. CodeSys Extract - Stopped State: State Initialisation & Forcing Variables

4.7 Resetting State

The Resetting state can be entered via the Complete state or Stopped state hence both these visualisation lamps are turned off and the Resetting state lamp is turned on. During this state many internal variables are reset including; Hopper_Eroor, Fill_Box, Stop_Box_at_Hopper, Stop_Filling_Box, Filled_Box_Pickup, Count_IN, Count_Out, Suspend_Case. Continuous_production. Boxes_rdy, Start_Conv_Delay_Timer. Start_Hopper_Solenoid_Timer. TimerDone_Solenoid_Timer. Start_Completing_Timer, Start_State_Delay_2, and Start_State_Delay_3. All timers are also reset in this state. The following lines of code start the state transition delay timer. Upon completion of this timer the CurrentState is set to Idle and this delay is Reset.

```
// --- RESETTING ---------------------------------------------------
 IF CurrentState = RESETTING THEN
    // --- FOR VISUALISATION ----
    COMPLETE_Lamp := FALSE;
    STOPPED_Lamp    := FALSE;
    RESETTING_Lamp  := TRUE;
    //--------------------------

    //Resetting internal variables
    Hopper_Error            := FALSE;
    Fill_Box                := FALSE;
    Stop_Box_at_Hopper      := FALSE;
    Stop_Filling_Box        := FALSE;
    Filled_Box_Pickup       := FALSE;
    Count_IN                := 0;
    Count_Out               := 0;
    Suspend_Case            := 0;
    Continuous_production   := FALSE;
    Boxes_rdy               := FALSE;
    Start_Conv_Delay_Timer := FALSE;
    Start_Hopper_Solenoid_Timer := FALSE;
    TimerDone_Solenoid_Timer := FALSE;
    Start_Completing_Timer  := FALSE;
    Start_State_Delay_2 := FALSE;
    Start_State_Delay_3 := FALSE;

    Initalise_Stop_Filling_Timer(IN := Start_Hopper_Solenoid_Timer, PT := Solenoid_Timer_PT);
    Conv_Delay_Timer(IN :=Start_Conv_Delay_Timer, PT :=Conv_Delay_Timer_PT);
    Completing_Timer(IN := Start_Completing_Timer, PT := Completing_Timer_PT);
    State_Delay_2(IN :=Start_State_Delay_2, PT := State_Delay_2_PT);
    State_Delay_3(IN :=Start_State_Delay_3, PT := State_Delay_3_PT);


    //Timer for state transition
    Start_State_Delay := TRUE;
    State_Delay(IN :=Start_State_Delay, PT := State_Delay_PT);
    State_Delay_DN := State_Delay.Q;
    IF State_Delay_DN = TRUE THEN
        CurrentState := IDLE;
        Start_State_Delay := FALSE;
        State_Delay(IN :=Start_State_Delay, PT := State_Delay_PT);
    END_IF
 END_IF
```

Figure 28. CodeSys Extract - Resetting State: State Initialisation & Forcing Variables/Timers

4.8 Suspending State

When the system enters the suspending state, the indicator lamp is turned on and execute lamp is turned off. The conveyor and hopper solenoid are turned off, and the robot arms are deactivated to prevent any unsafe situation. After the state transition delay timer finishes counting, the system enters the suspended state.

```
// --- SUSPENDING ----------------------------------------------------
    IF CurrentState = SUSPENDING THEN
        // --- FOR VISUALISATION ----
        EXECUTE_Lamp := FALSE;
        SUSPENDING_Lamp := TRUE;
        // --------------------------

        Run_Conveyor := FALSE;
        Hopper_Solenoid := FALSE;
        Robot_Arm_Pre := FALSE;
        Robot_Arm_Post := FALSE; //stop all output

        //Timer for state transition
        Start_State_Delay := TRUE;
        State_Delay(IN :=Start_State_Delay, PT := State_Delay_PT);
        State_Delay_DN := State_Delay.Q;
        IF State_Delay_DN = TRUE THEN
            CurrentState := SUSPENDED; // move to suspended state
            Start_State_Delay := FALSE;
            State_Delay(IN :=Start_State_Delay, PT := State_Delay_PT);
        END_IF
    END_IF
```

Figure 29. CodeSys Extract - Suspending State: State Initialisation & Timer

4.9 Suspended State

Entering the suspended state, the suspended lamp is on and suspending lamp is off. The system start monitoring the conditions for exiting it. The suspended state can be ended manually when receiving an external cmd_unsuspending command. In the case of Suspend_Case = 1 or 2, if the robotic arm in the next unit sends a signal indicating it is ready to continue, the system can automatically transition to the unsuspending state. For suspend cases 3 and 4, which represent hopper error and the place box robot arm error respectively, manual inspection is required to identify the cause of the fault. Once the issue is resolved, the operator must manually press the unsuspend button and wait for a state transition delay to return the system to the Execute state.

```
    // --- SUSPENDED ----------------------------------------------------
    IF CurrentState = SUSPENDED THEN
        // --- FOR VISUALISATION ----
        SUSPENDING_Lamp := FALSE;
        SUSPENDED_Lamp := TRUE;
        //--------------------------

        //Timer for lamp transitions
        Start_State_Delay := TRUE;
        State_Delay(IN :=Start_State_Delay, PT := State_Delay_PT);
        State_Delay_DN := State_Delay.Q;
        IF State_Delay_DN = TRUE THEN
            IF Cmd_Unsuspend THEN
                CurrentState := UNSUSPENDING;
                Start_State_Delay := FALSE;

                State_Delay(IN :=Start_State_Delay, PT := State_Delay_PT);
            ELSIF Suspend_Case = 1 AND Robot_Arm_Post_Ready = TRUE THEN
                CurrentState := UNSUSPENDING;
                Start_State_Delay := FALSE;
                State_Delay(IN :=Start_State_Delay, PT := State_Delay_PT);
            ELSIF Suspend_Case = 2 AND Robot_Arm_Post_DONE = TRUE THEN
                CurrentState := UNSUSPENDING;
                Start_State_Delay := FALSE;
                State_Delay(IN :=Start_State_Delay, PT := State_Delay_PT);
            END_IF
        END_IF
    END_IF
```

Figure 30. CodeSys Extract - Suspended State: State Initialisation & Timer

4.10 Unsuspending State

After transferring to the unsuspending state, the corresponding lamps change. For different suspend cases, we reset the corresponding variables (such as the robot arm state, hopper filling timer, etc.) to ensure that when the system returns to the Execute state, it can operate normally without looping back into the suspending state. We also reset the suspend_case variable to 0 to allow it to indicate a new suspend case the next time.

```
// --- UNSUSPENDING -----------------------------------------------------
IF CurrentState = UNSUSPENDING THEN
    // --- FOR VISUALISATION ----
    SUSPENDED_Lamp := FALSE;
    UNSUSPENDING_Lamp := TRUE;
    //--------------------------

    IF Suspend_Case = 1 THEN
        Robot_Arm_Post_Ready := TRUE;
    ELSIF Suspend_Case = 2 THEN
        Robot_Arm_Post_Done := TRUE;
    ELSIF Suspend_Case = 3 THEN
        Hopper_Error := FALSE;
        Start_Hopper_Solenoid_Timer := FALSE;
        TimerDone_Solenoid_Timer := FALSE; //filling over time
        Initalise_Stop_Filling_Timer(IN := Start_Hopper_Solenoid_Timer, PT := Solenoid_Timer_PT); // included to reset the accumulated value
    END_IF

    Suspend_Case := 0;
```

Figure 31. CodeSys Extract - Unsuspending State: State Initialisation & Checks

Before moving back to execute state, we also need to reset all the timers in case unexpected time out. And then after the state transition delay, the system will go back to execute state and continue previous works.

```
    //Resetting all timers
    Start_State_Delay_2 := FALSE;
    Start_State_Delay_3 := FALSE;
    State_Delay_2(IN :=Start_State_Delay_2, PT := State_Delay_PT);
    State_Delay_3(IN :=Start_State_Delay_3, PT := State_Delay_3_PT);
    State_Delay(IN :=Start_State_Delay, PT := State_Delay_PT);
    State_Delay_DN_2 := State_Delay_2.Q;
    State_Delay_DN_3 := State_Delay_3.Q;

    //Timer for lamp transitions
    Start_State_Delay := TRUE;
    State_Delay(IN :=Start_State_Delay, PT := State_Delay_PT);
    State_Delay_DN := State_Delay.Q;

    IF State_Delay_DN = TRUE THEN
        CurrentState := EXECUTE;
        Start_State_Delay := FALSE;
        State_Delay(IN :=Start_State_Delay, PT := State_Delay_PT);
    END_IF
END_IF
```

Figure 32. CodeSys Extract - Unsuspending State: Timer Resets and State Transition

4.11 Holding State

When ‘CurrentState’ of the program transitions to the Holding State, all outputs of the conveyor are de-activated (ie. the conveyor, hopper solenoid and robot arms) by setting them to ‘FALSE’. The program then transitions to the Held State after a delay of 5-seconds, which is implemented with the State Delay TON function explained in Section 4.1.

According to the PackML State diagram, the Holding state can only transition from the Exeute State. Therefore, the Execute Visualisation Lamp has been set to ‘FALSE’ to turn it off, while setting the Holding Visualisation Lamp to ‘TRUE’ to turn it on.

```
// --- HOLDING -----------------------------------------------------
IF CurrentState = HOLDING THEN
    //// --- FOR VISUALISATION ----
    EXECUTE_Lamp := FALSE;
    HOLDING_Lamp := TRUE;
    //---------------------------

    Run_Conveyor := FALSE;
    Hopper_Solenoid := FALSE;
    Robot_Arm_Pre := FALSE;
    Robot_Arm_Post := FALSE;

    //Timer for state transition
    Start_State_Delay := TRUE;
    State_Delay(IN :=Start_State_Delay, PT := State_Delay_PT);
    State_Delay_DN := State_Delay.Q;
    IF State_Delay_DN = TRUE THEN
        CurrentState := HELD; // move to HELD state
        Start_State_Delay := FALSE;
        State_Delay(IN :=Start_State_Delay, PT := State_Delay_PT);
    END_IF
END_IF
//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Figure 33. CodeSys Extract - Holding State: State Initialisation & Timer

4.12 Held State

The Held State is a waiting state that responds to an ‘Unhold’ command to prompt the transition to the Unholding State. Hence, no outputs are changed during this state, as shown in Figure 34. The ‘Cmd_Unhold’ variable is used to implement an Unhold Command from the user. The transition to the Unholding State is triggered when ‘Cmd_Unhold’ is pressed, which is indicated by a ‘TRUE’ value. An IF loop statement is used to implement this condition in order to set the ‘CurrentState’ to ‘Unholding’ when the button is pressed.

```
//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

// --- HELD -----------------------------------------------------
IF CurrentState = HELD THEN
    // --- FOR VISUALISATION ----
    HOLDING_Lamp := FALSE;
    HELD_Lamp := TRUE;
    //---------------------------

    //Resetting conveyor delay or jam timer
    Start_Conv_Delay_Timer := FALSE;
    Conv_Delay_Timer(IN :=Start_Conv_Delay_Timer, PT :=Conv_Delay_Timer_PT);
    TimerDone_Conv_Delay_Timer := FALSE;

    IF Cmd_Unhold = TRUE THEN
        CurrentState := UNHOLDING;
    END_IF
END_IF
//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Figure 34. CodeSys Extract - Held State: State Initialisation & Timer

As the Holding State is triggered by the 'Conv_Delay_Timer' being complete, Lines 460-463 are used to reset the state of the timer. Otherwise, the 'CurrentState' would continue to be set to the Holding State due to the IF Loop in Lines 94-95 in Figure 15 being executed. Lines 456-457 are used to turn off the Holding and Held Visualisation Lamps off and on, by assigning 'FALSE' and 'TRUE' to the lamp variables, respectively.

4.13 Unholding State

In the Unholding State, the conveyor is turned on to resume the movement of the boxes on the conveyor. This is implemented by setting 'Run_Conveyor' to 'TRUE', according to Figure 35. The program then transitions to the Execute State to resume its normal operation with all input sensors and outputs enabled. Lines 481-487 implements the state transition timer for visualisation purposes, as explained in Section 4.1.

```
471     // --- UNHOLDING -----------------------------------------------------
472     IF CurrentState = UNHOLDING THEN
473         // --- FOR VISUALISATION ----
474         HELD_Lamp := FALSE;
475         UNHOLDING_Lamp := TRUE;
476         //--------------------------
477
478         Run_Conveyor := TRUE;
479
480         //Timer for state transition
481         Start_State_Delay := TRUE;
482         State_Delay(IN :=Start_State_Delay, PT := State_Delay_PT);
483         State_Delay_DN := State_Delay.Q;
484         IF State_Delay_DN = TRUE THEN
485             CurrentState := EXECUTE; // move to HELD state
486             Start_State_Delay := FALSE;
487             State_Delay(IN :=Start_State_Delay, PT := State_Delay_PT);
488         END_IF
489     END_IF
490     //xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Figure 35. CodeSys Extract - Unholding State: State Initialisation & Timer

4.14 Completing State

The Completing state occurs when the conveyor system has produced the desired number of boxes and exits the Execute state. Therefore, the visualisation lamp for Execute is turned off by assigning it to 'FALSE' in Line 570 in Figure 36 and the Completing lamp is assigned 'TRUE' to indicate that the program is in the completing state. A 5-second delay is programmed before transitioning to the Complete state to allow the program to remain momentarily idle. This delay is implemented with the 'Completing_Timer' TON function, which is enabled by the 'Start_Completing_Timer' variable being set to 'TRUE'. Line 579 uses an IF loop so that the program transitions to the Complete state if the timer is done.

```
567     // --- COMPLETING -----------------------------------------------------
568     IF CurrentState = COMPLETING THEN
569         // --- FOR VISUALISATION ----
570         EXECUTE_Lamp := FALSE;
571         COMPLETING_Lamp := TRUE;
572         //--------------------------
573
574         //5 second timer before moving to Complete
575         Start_Completing_Timer := TRUE;
576         Completing_Timer(IN := Start_Completing_Timer, PT := Completing_Timer_PT);
577         TimerDone_Completing_Timer := Completing_Timer.Q;
578         IF TimerDone_Completing_Timer THEN
579             CurrentState := 16; // move to Complete
580         END_IF
581     END_IF
582     //xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Figure 36. CodeSys Extract - Completing State: State Initialisation & Timer

4.15 Complete State

In the Complete State, the Completing visualisation lamp is set to ‘FALSE’ and the Complete lamp is set to ‘TRUE’ to display the state transition on the visualisation. The Complete state is a waiting state and only responds to a reset command, which is indicated by the ‘Cmd_Reset’ variable. The IF Loop in Line 591 in Figure 37 is used to transition to the Reset state if this variable is set to ‘TRUE’.

```
584     // -- COMPLETE --------------------------------------------------------
585     IF CurrentState = COMPLETE THEN
586         // --- FOR VISUALISATION ----
587         COMPLETING_Lamp := FALSE;
588         COMPLETE_Lamp := TRUE;
589         //-------------------------
590
591         IF Cmd_Reset THEN
592             CurrentState := 6; //move to Resetting
593         END_IF
594     END_IF
595     //xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Figure 37. CodeSys Extract - Complete State: State Initialisation & Timer

4.16 Aborting State

The Aborting State can occur by pressing a manual Abort button, or when a DC Power Supply fault occurs. The code implements this in Figure 25, which shows that the transition to the Aborting State in Line 16 is triggered if Cmd_Abort becomes TRUE or if DC_Power_OK turns FALSE. An additional variable, Power_supply_fault, has been linked to DC_Power_OK to allow the program to differentiate if the Aborting State is triggered by a command or a power supply condition.

When the program is in the Aborting State, all other visualisation lamp variables are set to FALSE as the program can be aborted from any state. This is shown in Figure 38. DC_Power_OK is set back to TRUE to internally acknowledge and reset the variable and avoid the IF Loop in Line 14 from triggering again. All outputs of the conveyor system are then de-activated by their variables being set to FALSE, as shown in Lines 516 to 519. When complete, the State Delay, explained in section 4.1, is implemented when transitioning to the Aborted State.

```
521         //Timer for state transition
522         Start_State_Delay := TRUE;
523         State_Delay(IN :=Start_State_Delay, PT := State_Delay_PT);
524         State_Delay_DN := State_Delay.Q;
525         IF State_Delay_DN = TRUE THEN
526             CurrentState := ABORTED; // move to aborted state
527             Start_State_Delay := FALSE;
528             State_Delay(IN :=Start_State_Delay, PT := State_Delay_PT);
529         END_IF
530     END_IF
531     //xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Figure 38. CodeSys Extract - Aborting State: State Lamp Controls and Variable Reset

4.17 Aborted State

In the Aborted State, the Aborting visualisation lamp is turned off and the Aborted lamp is turned on by setting the corresponding variables to FALSE and TRUE, respectively, to represent the state transition. This is shown in Figure 39. The Aborted state is a waiting state that only responds to a Clear command. The IF Loop in Lines 540-542 uses the Clear command (ie. Cmd_clear) to enable the transition to the Clearing state.

```
493     // --- ABORTING ------------------------------------------------------
494     IF CurrentState = ABORTING THEN
495         // --- FOR VISUALISATION ----
496         IDLE_Lamp            := FALSE;
497         STARTING_Lamp        := FALSE;
498         IDLE_Lamp            := FALSE;
499         EXECUTE_Lamp         := FALSE;
500         STOPPING_Lamp        := FALSE;
501         RESETTING_Lamp       := FALSE;
502         HOLDING_Lamp         := FALSE;
503         HELD_Lamp            := FALSE;
504         UNHOLDING_Lamp       := FALSE;
505         SUSPENDING_Lamp      := FALSE;
506         SUSPENDED_Lamp       := FALSE;
507         UNSUSPENDING_Lamp    := FALSE;
508         CLEARING_Lamp        := FALSE;
509         STOPPED_Lamp         := FALSE;
510         COMPLETING_Lamp      := FALSE;
511         COMPLETE_Lamp        := FALSE;
512         ABORTING_Lamp        := TRUE;
513         //-----------------------------------
514
515         DC_Power_OK := TRUE; // internally acknowledges (and resets state) of fault. Mitigates against looping back to Aborting
516         Run_Conveyor := FALSE;
517         Hopper_Solenoid := FALSE;
518         Robot_Arm_Pre := FALSE;
519         Robot_Arm_Post := FALSE;
520
521         //Timer for state transition
522         Start_State_Delay := TRUE;
523         State_Delay(IN :=Start_State_Delay, PT := State_Delay_PT);
524         State_Delay_DN := State_Delay.Q;
525         IF State_Delay_DN = TRUE THEN
526             CurrentState := ABORTED; // move to aborted state
527             Start_State_Delay := FALSE;
528             State_Delay(IN :=Start_State_Delay, PT := State_Delay_PT);
529         END_IF
530     END_IF
531     //xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Figure 39. CodeSys Extract - Aborted State: State Initialisation & Timer

4.18 Clearing State

The Clearing state involves turning the Aborted visualisation lamp off and the Clearing lamp on, which is implemented in Lines 549-550 in Figure 40. This state involves clearing the power supply fault, hence Power_supply_fault being set to FALSE in Line 553. The program then transitions to the Stopped state after the State Delay timer, explained in section 4.1, is complete.

```
546     // --- CLEARING ------------------------------------------------------
547     IF CurrentState = CLEARING THEN
548         // --- FOR VISUALISATION ----
549         ABORTED_Lamp := FALSE;
550         CLEARING_Lamp := TRUE;
551         //---------------------------
552
553         Power_supply_fault := FALSE;
554
555         //Timer for state transition
556         Start_State_Delay := TRUE;
557         State_Delay(IN :=Start_State_Delay, PT := State_Delay_PT);
558         State_Delay_DN := State_Delay.Q;
559         IF State_Delay_DN = TRUE THEN
560             CurrentState := STOPPED; // move to Stopped state
561             Start_State_Delay := FALSE;
562             State_Delay(IN :=Start_State_Delay, PT := State_Delay_PT);
563         END_IF
564     END_IF
565     //xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Figure 40. CodeSys Extract - Clearing State: State Initialisation & Timer

5. Ability for User Modifications

It’s critical to ensure that the code can be modified to meet specific user requirements. This is important as the size of production lines may vary between factories. This means that the time it takes for a box to fill or for an empty box to be placed on the conveyor would differ. Therefore, to overcome these challenges. The timers for all aspect of the code can be easily modified within the initial lines of CodeSys. This is seen from Figure 41.

```
//--- For User Modification of Timer Presets: ---
    Conv_Delay_Timer_PT := T#15S;
    Solenoid_Timer_PT := T#20S;
    Completing_Timer_PT := T#5S;
    State_Delay_PT := T#5S;
    State_Delay_2_PT := T#2S;
    State_Delay_3_PT := T#5S;

//---------------------------------
```

Figure 41. CodeSys Extract, Timers: Ability for Modifications

Additionally, since a manufacturing line can vary from the developed four box model. The length of the conveyor can be modified to meet the user’s current factory configuration. However, the system does require enough space for at least three boxes to fit on the conveyor with space between them. This is required as one box is simultaneously placed on and taken off the conveyor while another box is being filled. This functionality was achieved by only notifying the next cell that a box is ready to be picked up, once the end conveyor sensor is triggered. Therefore, the system can theoretically be of any length.

The other major aspect with allows for the user to meet specific quotas is the ability to choose the production type. The operator can decide on if they want to operate the cell as a continuous production. Or if they only want to produce a specific number of boxes. This functionality can easily be achieved by selecting a value from the drop-down box in the top left corner of the visualisation panel, this can be seen from Figure 42 below.

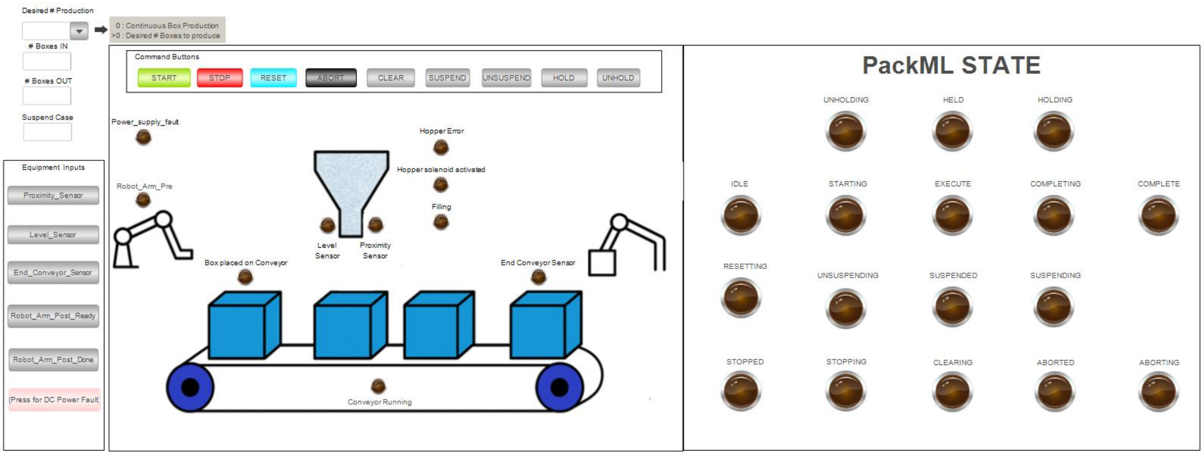

Figure 42. Visualisation Model: Production Type

6. Potential Design Improvements

While the code appears to operate as intended, it can still be improved to potentially improve computational time and efficiency. The ST is currently modelled using an extensive number of conditional IF statements. This could be improved by using a more segmented approach when developing the code. By having smaller sections of code which can call upon each other, the PLC code can potentially run quicker. Additionally, this methodology can help reduce the likelihood that the system unexpectedly moves into another state during an edge case scenario which hasn't been considered.

Another aspect of the design which hasn't been considered is the effect that free floating Styrofoam may have on the system. It is assumed that the boxes will fill perfectly with no loss of product. However, if some styrofoam does get onto the conveyor belt then it may produce false readings to any one of the proximity sensors. Resulting in the system to get stuck at a specific stage and not returning a fault. These such scenarios have been considered out of scope. However, they would impact the operation of the system within a manufacturing line and should be considered in future work.

An alternative approach could be using another format entirely. By utilising either FBD or ladder logic the model would become much easier for technicians and frontline engineers to understand and learn within a short period of time. This would be critical if errors arise within the field and require troubleshooting. FBD or ladder logic will allow people to understand and modify the foundational model as required throughout the lifespan of the asset.

5. References

[1] B. Nener, *Introduction to PackML*, UWA ELEC5506, Semester 1, 2025, University of Western Australia, Western Australia.

[2] D. Meyer, *Designing your first PackML implementation for machine control: Three key design decisions to get you started*, WP.MTN.02, Yaskawa America, Inc., 2012.

[3] 'Machine and Unit States: an implementation Example of ISA-88'. OMAC, NC, Aug. 01, 2008. doi: 978-1-934394-81-6.

6. Appendix

6.1Appendix A: Internal Variables

Table 4. Internal & State Variables

Name	Description	
CurrentState	At every state transition checked and incremented accordingly	
IDLE	0: When system in Idle state	False: Otherwise
STARTING	1: When system in Starting state	False: Otherwise
EXECUTE	2: When system in Execute state	False: Otherwise
STOPPING	3: When system in Stopping state	False: Otherwise
ABORTING	4: When system in Aborting state	False: Otherwise
ABORTED	5: When system in Aborted state	False: Otherwise
RESETTING	6: When system in Resetting state	False: Otherwise
HOLDING	7: When system in Holding state	False: Otherwise
HELD	8: When system in Held state	False: Otherwise
UNHOLDING	9: When system in Unholding state	False: Otherwise
SUSPENDING	10: When system in Suspending state	False: Otherwise
SUSPENDED	11: When system in Suspended state	False: Otherwise
UNSUSPENDING	12: When system in Unsuspending state	False: Otherwise
CLEARING	13: When system in Clearing state	False: Otherwise
STOPPED	14: When system in Stopped state	False: Otherwise
COMPLETING	15: When system in Completing state	False: Otherwise
COMPLETE	16: When system in Complete	False: Otherwise
Hopper_Error	True: Hopper filling over-time	False: Otherwise
Stop_Box_at_Hopper	True: If Run_Conveyor AND Proximity_Sensor	False: Otherwise
Conveyor_Run_Post_Fill		
Stop_Filling_Box	True: If Proximity_Sensor AND NOT Run_Conveyor AND NOT Emergency_Stop AND Level_Sensor;	False: Otherwise
Count_IN	Number of boxes filled	
Count_Out	Number of filled boxes sent to the next work-cell	
Suspend_Case	Indicates the error type causing the program suspension	
Continuous_production	True: If system in Stopping, Stopped or Resetting	False: Otherwise
Boxes_rdy	True: When State_Delay timer complete	False: Otherwise
Power_supply_fauly	True: DC Power supply fault	False: Otherwise
End_Conveyor_Sensor_Old	True: If End_Conveyor_Sensor	False: otherwise

Table 5. Visualisation Bits

Name	Description	
IDLE_Lamp	True: When system in Idle state	False: Otherwise
STARTING_Lamp	True: When system in Starting state	False: Otherwise
EXECUTE_Lamp	True: When system in Execute state	False: Otherwise
STOPPING_Lamp	True: When system in Stopping state	False: Otherwise
ABORTING_Lamp	True: When system in Aborting state	False: Otherwise
ABORTED_Lamp	True: When system in Aborted state	False: Otherwise
RESETTING_Lamp	True: When system in Resetting state	False: Otherwise
HOLDING_Lamp	True: When system in Holding state	False: Otherwise
HELD_Lamp	True: When system in Held state	False: Otherwise
UNHOLDING_Lamp	True: When system in Unholding state	False: Otherwise
SUSPENDING_Lamp	True: When system in Suspending state	False: Otherwise
SUSPENDED_Lamp	True: When system in Suspended state	False: Otherwise
UNSUSPENDING_Lamp	True: When system in Unsuspending state	False: Otherwise
CLEARING_Lamp	True: When system in Clearing state	False: Otherwise
STOPPED_Lamp	True: When system in Stopped state	False: Otherwise
COMPLETING_Lamp	True: When system in Completing state	False: Otherwise
COMPLETE_Lamp	True: When system in Complete state	False: Otherwise