Embedded Systems

PROJECT – SEMESTER 04

Group 3

Minori Perera	UOG0322051
Vidun Nethdina	UOG0221016
Nadev Gunaratne	UOG0221004
Rometh Gunathilake	UOG0221012

Task 1 - Identify the principles of embedded system

1.1 Identify features in embedded system and describe which can apply to your system

Our system can use the following embedded system features as follows:

- Real-time operation
  - The embedded system must operate in real-time to manage the synchronization of conveyor belts and motors.

- Resource Constraints
  - The system should operate within the constraints of limiting processing power and memory.

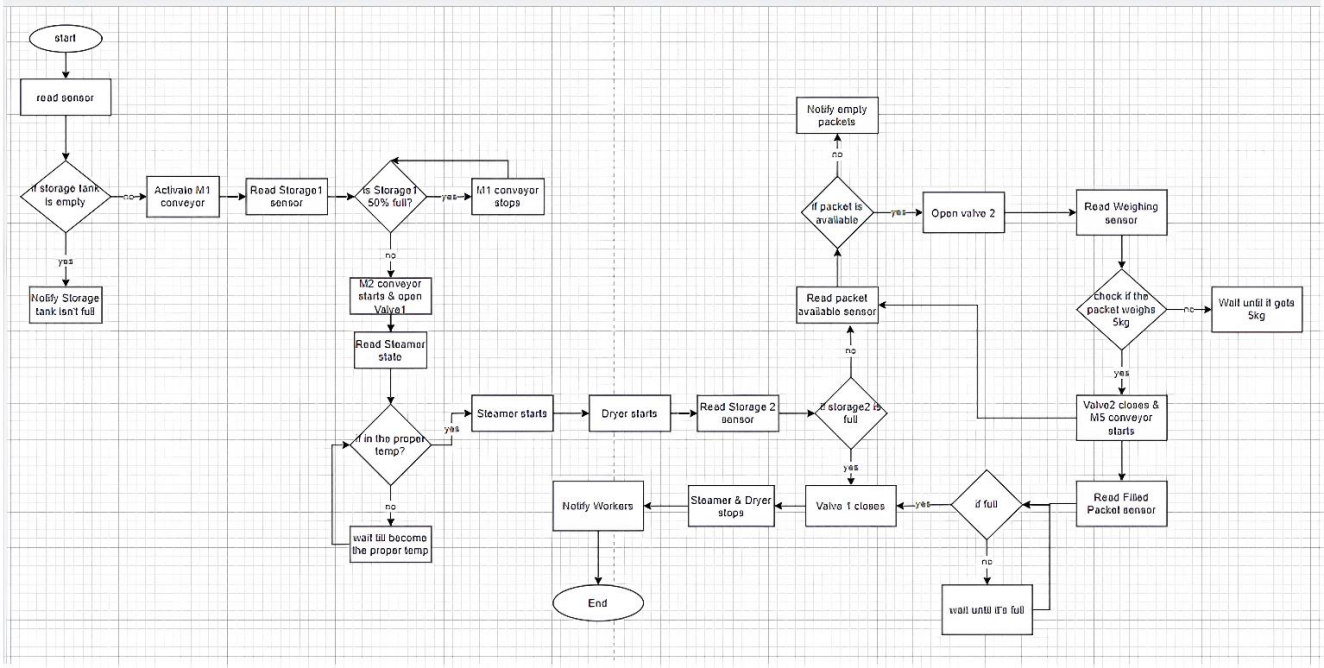
- Integration and Hardware
  - Due to the various stages of the milling process, different types of hardware such as sensors , controllers for the motors, conveyor belts and container need to be interfaced with the software.

- Long Lifecycle
  - As the mill is likely to be in continuous use, the system needs durable components and software that can be maintained for designed for prolonged operational life.

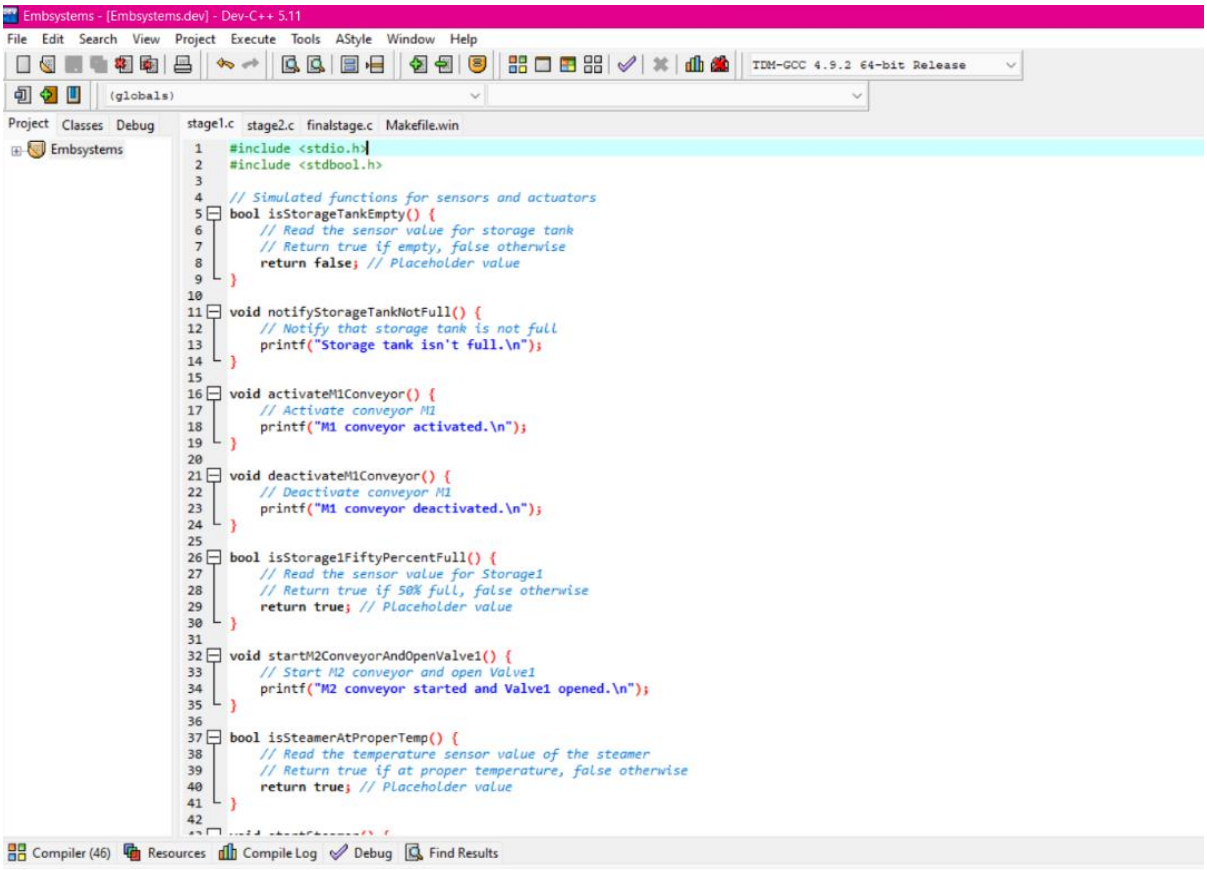
- Security
  - Security measures are crucial to protect the embedded system from unauthorized access or tampering. As such we only allow authorized personnel to adjust and/or control the system.

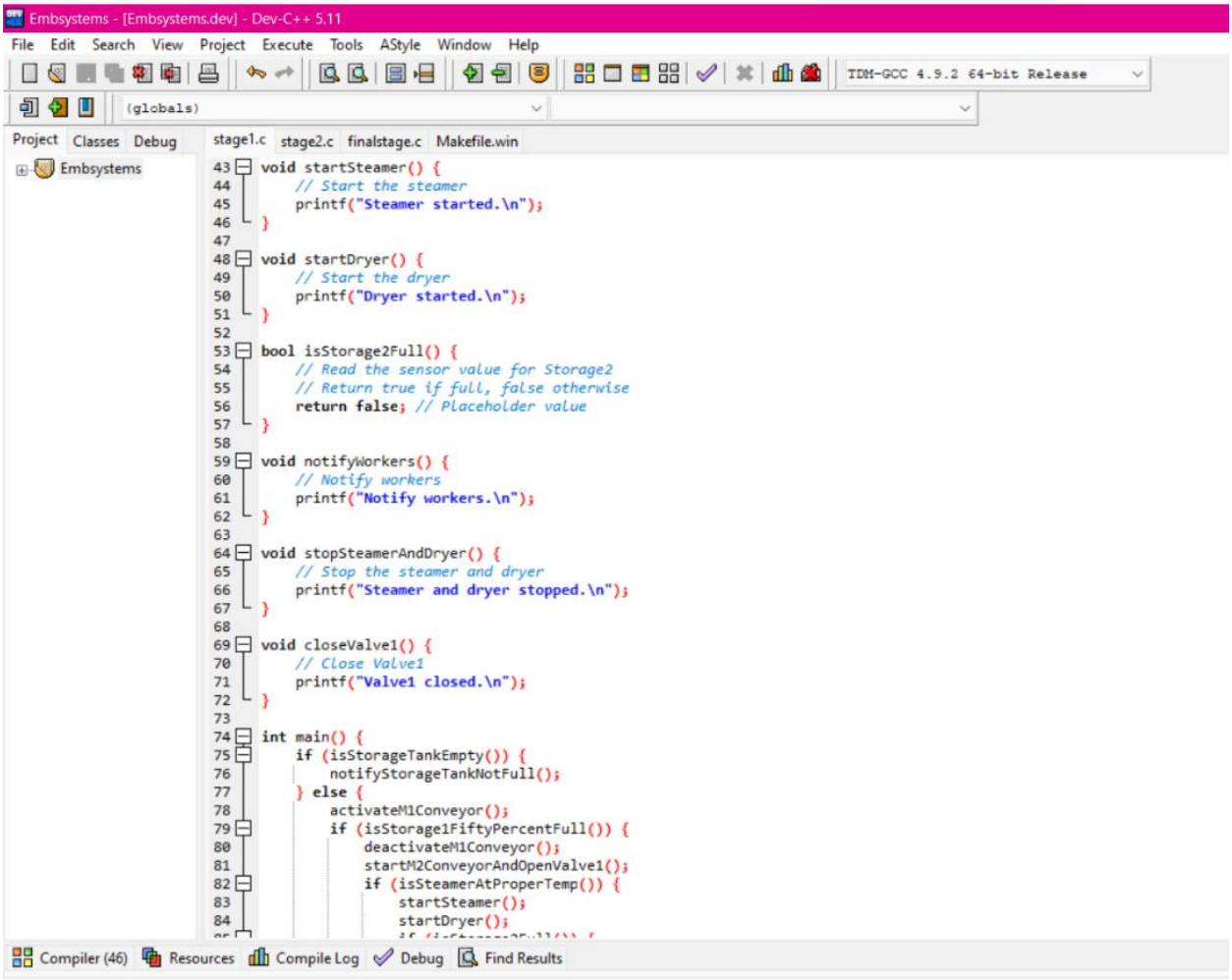
Task 2 – Design embedded system solutions

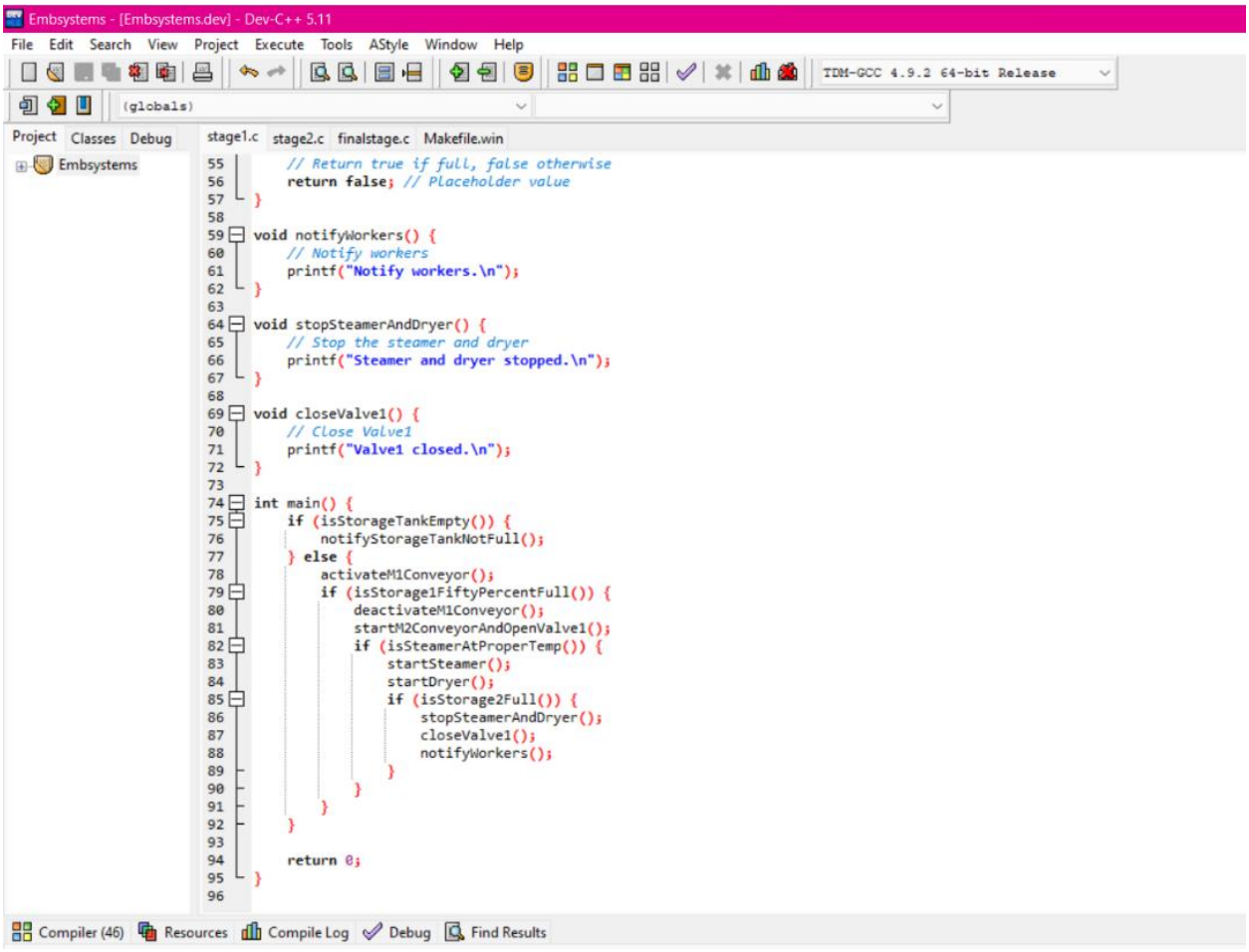
2.1 Design a Flow chart diagram for your system

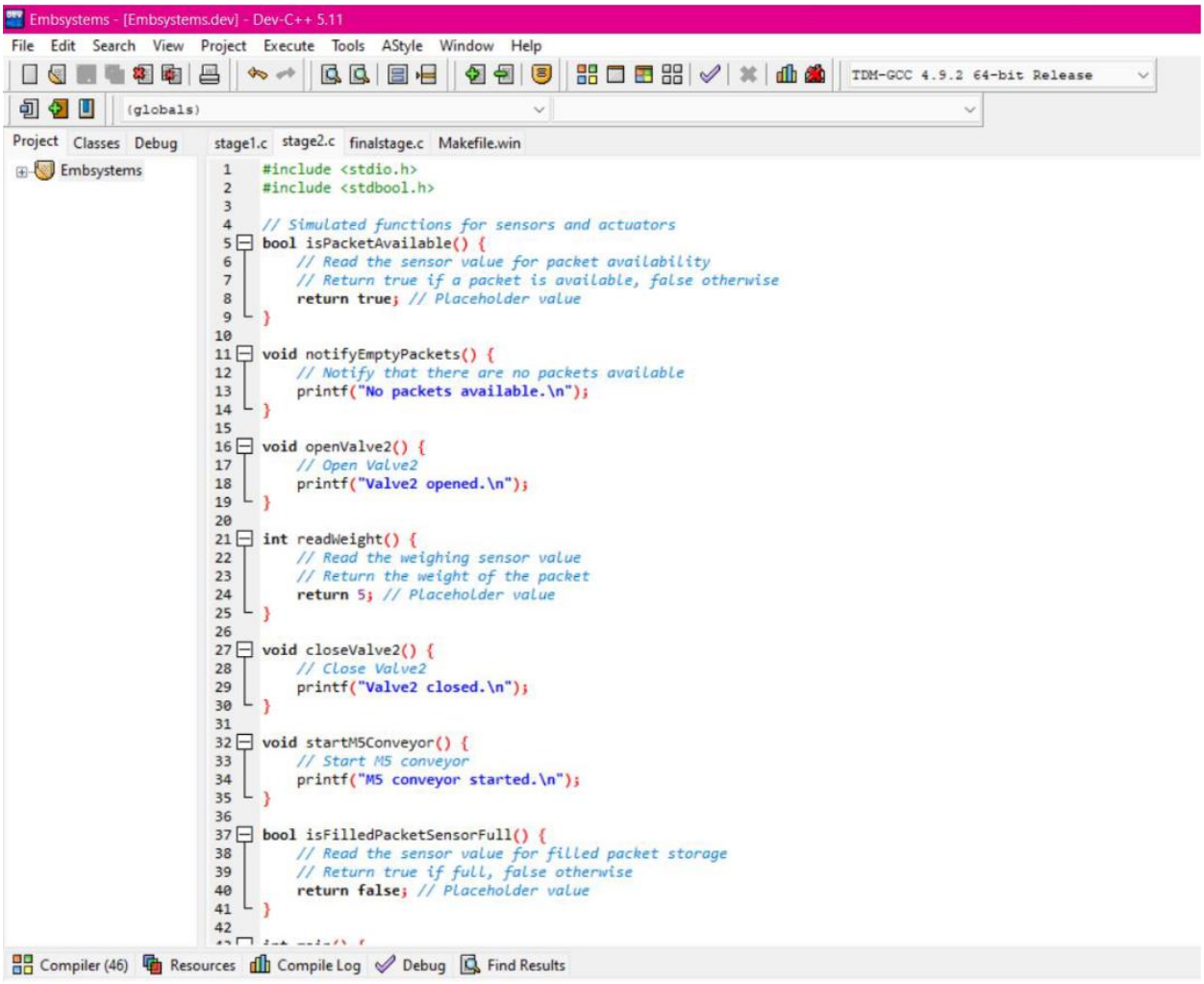


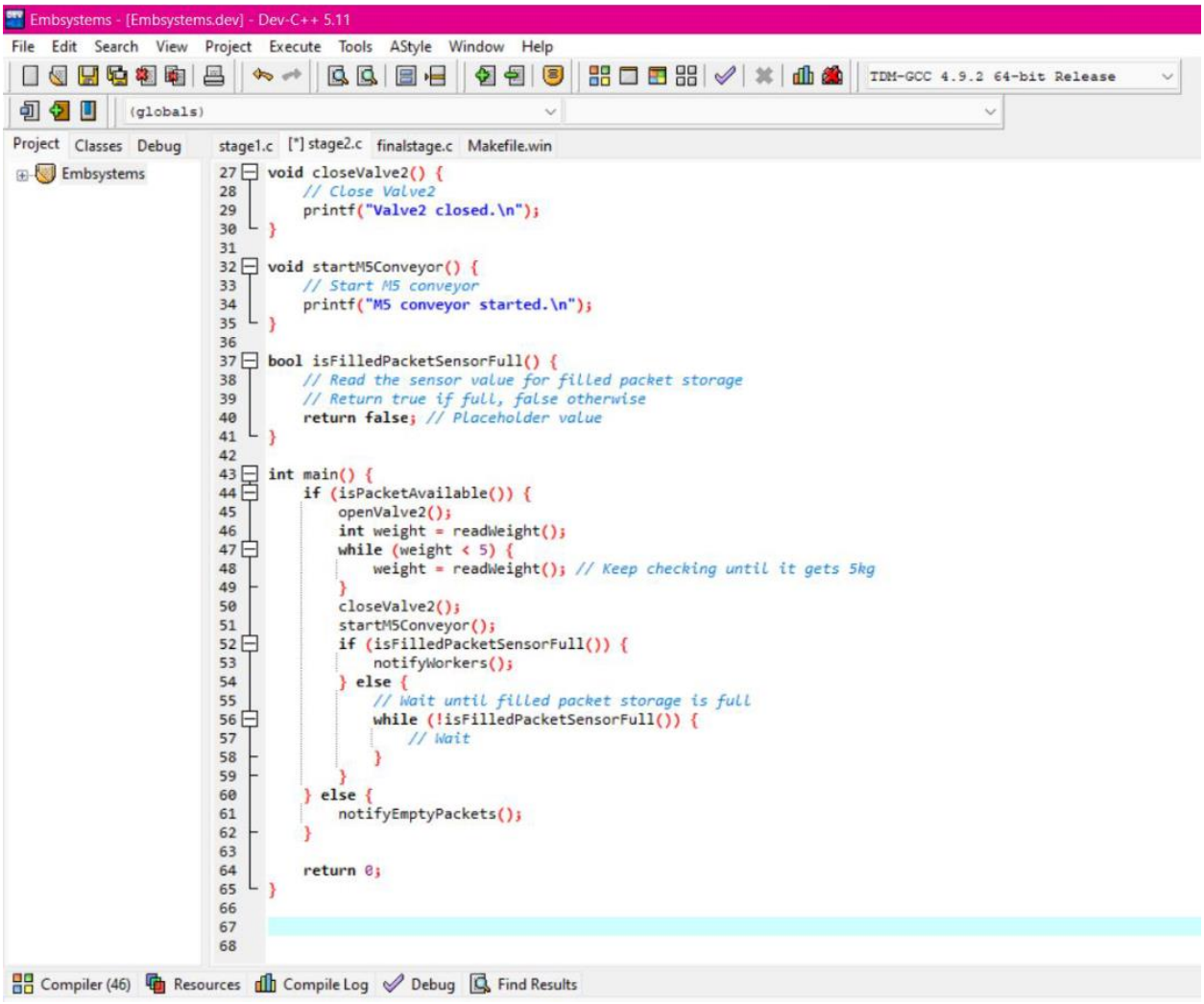
2.2 Implement the code program with selecting suitable microcontroller based on the prepared flow chart diagram

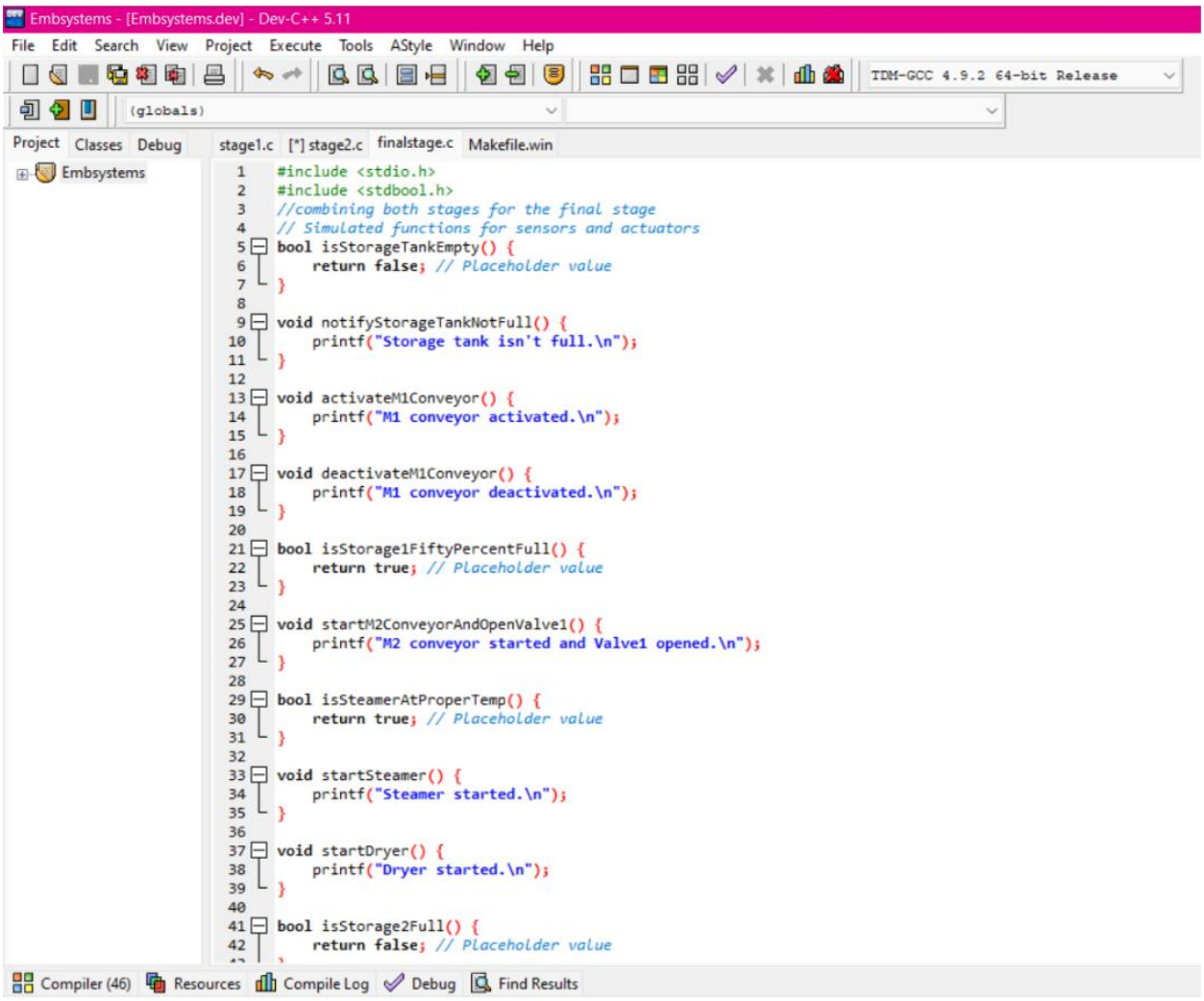


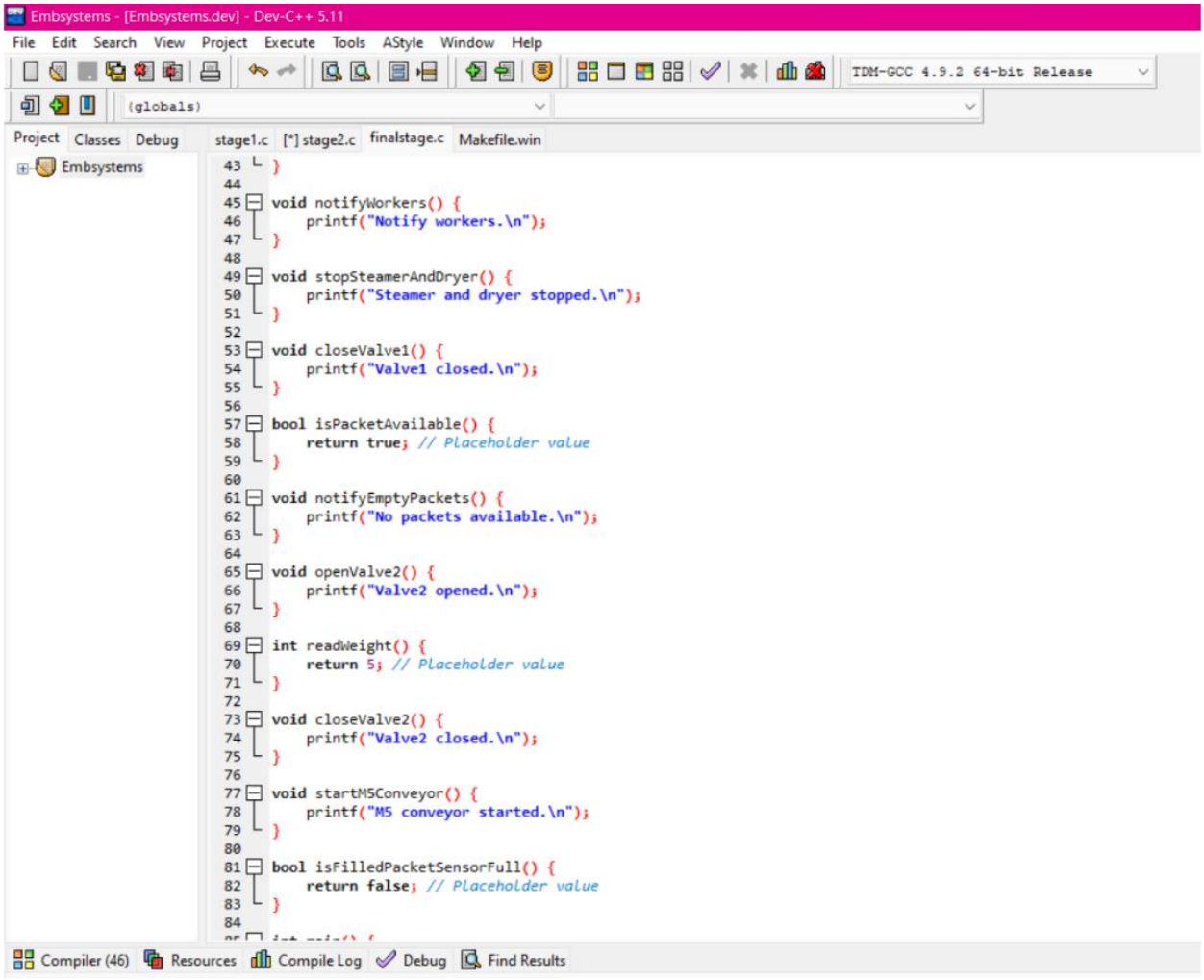

Embsystems - [Embsystems.dev] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
Embsystems
stage1.c stage2.c finalstage.c Makefile.win
43 void startSteamer() {
44 // Start the steamer
45 printf("Steamer started.\n");
46 }
47
48 void startDryer() {
49 // Start the dryer
50 printf("Dryer started.\n");
51 }
52
53 bool isStorage2Full() {
54 // Read the sensor value for Storage2
55 // Return true if full, false otherwise
56 return false; // Placeholder value
57 }
58
59 void notifyWorkers() {
60 // Notify workers
61 printf("Notify workers.\n");
62 }
63
64 void stopSteamerAndDryer() {
65 // Stop the steamer and dryer
66 printf("Steamer and dryer stopped.\n");
67 }
68
69 void closeValve1() {
70 // Close Valve1
71 printf("Valve1 closed.\n");
72 }
73
74 int main() {
75 if (isStorageTankEmpty()) {
76 notifyStorageTankNotFull();
77 } else {
78 activateM1Conveyor();
79 if (isStorage1FiftyPercentFull()) {
80 deactivateM1Conveyor();
81 startM2ConveyorAndOpenValve1();
82 if (isSteamerAtProperTemp()) {
83 startSteamer();
84 startDryer();
Compiler (46) Resources Compile Log Debug Find Results

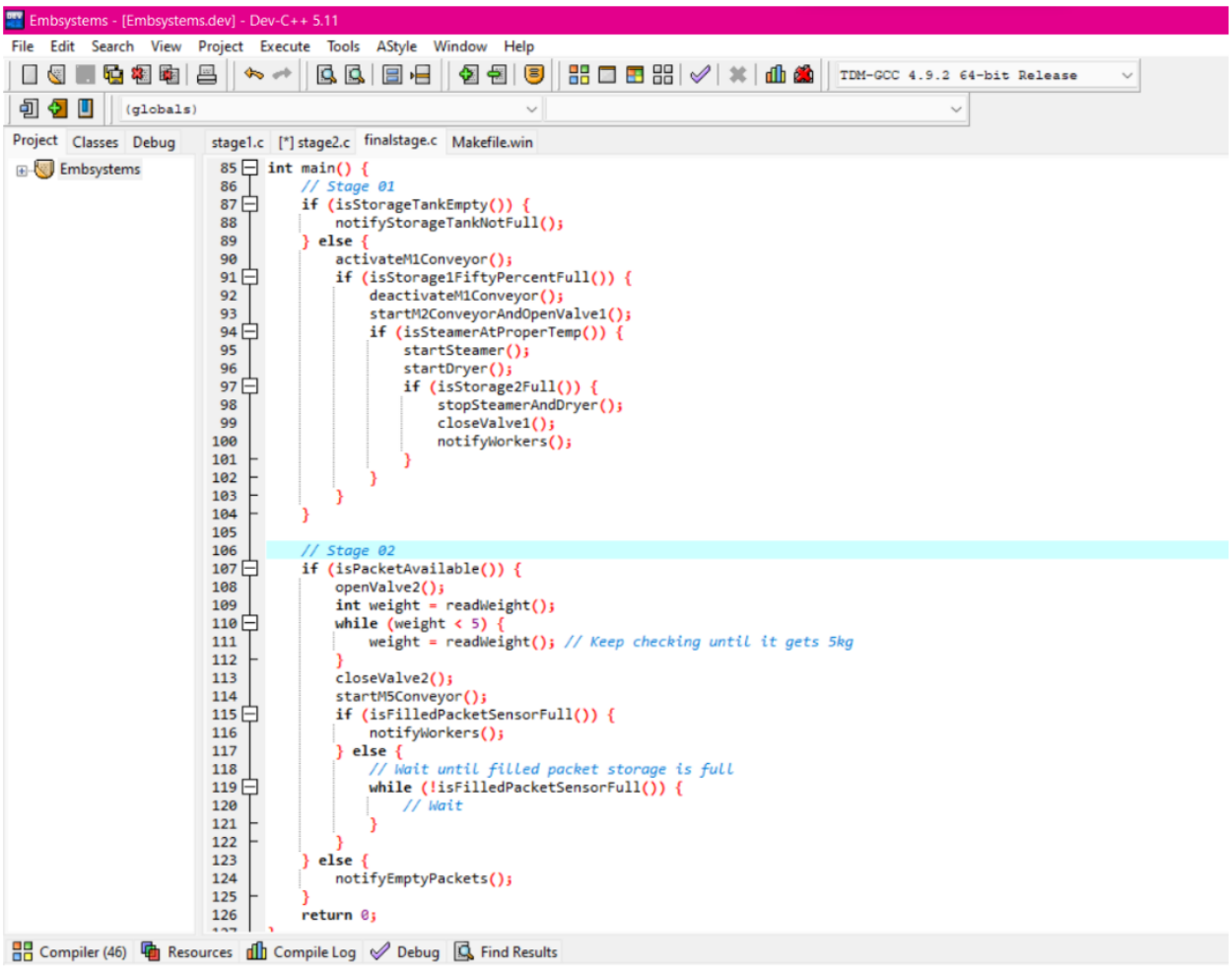

Embsystems - [Embsystems.dev] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
Embsystems
stage1.c stage2.c finalstage.c Makefile.win
// Return true if full, false otherwise
return false; // Placeholder value
}
void notifyWorkers() {
// Notify workers
printf("Notify workers.\n");
}
void stopSteamerAndDryer() {
// Stop the steamer and dryer
printf("Steamer and dryer stopped.\n");
}
void closeValve1() {
// Close Valve1
printf("Valve1 closed.\n");
}
int main() {
if (isStorageTankEmpty()) {
notifyStorageTankNotFull();
} else {
activateM1Conveyor();
if (isStorage1FiftyPercentFull()) {
deactivateM1Conveyor();
startM2ConveyorAndOpenValve1();
if (isSteamerAtProperTemp()) {
startSteamer();
startDryer();
if (isStorage2Full()) {
stopSteamerAndDryer();
closeValve1();
notifyWorkers();
}
}
}
}
return 0;
}
Compiler (46) Resources Compile Log Debug Find Results


Embsystems - [Embsystems.dev] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
Embsystems
stage1.c stage2.c finalstage.c Makefile.win
#include <stdio.h>
#include <stdbool.h>
// Simulated functions for sensors and actuators
bool isPacketAvailable() {
// Read the sensor value for packet availability
// Return true if a packet is available, false otherwise
return true; // Placeholder value
}
void notifyEmptyPackets() {
// Notify that there are no packets available
printf("No packets available.\n");
}
void openValve2() {
// Open Valve2
printf("Valve2 opened.\n");
}
int readWeight() {
// Read the weighing sensor value
// Return the weight of the packet
return 5; // Placeholder value
}
void closeValve2() {
// Close Valve2
printf("Valve2 closed.\n");
}
void startM5Conveyor() {
// Start M5 conveyor
printf("M5 conveyor started.\n");
}
bool isFilledPacketSensorFull() {
// Read the sensor value for filled packet storage
// Return true if full, false otherwise
return false; // Placeholder value
}
Compiler (46) Resources Compile Log Debug Find Results

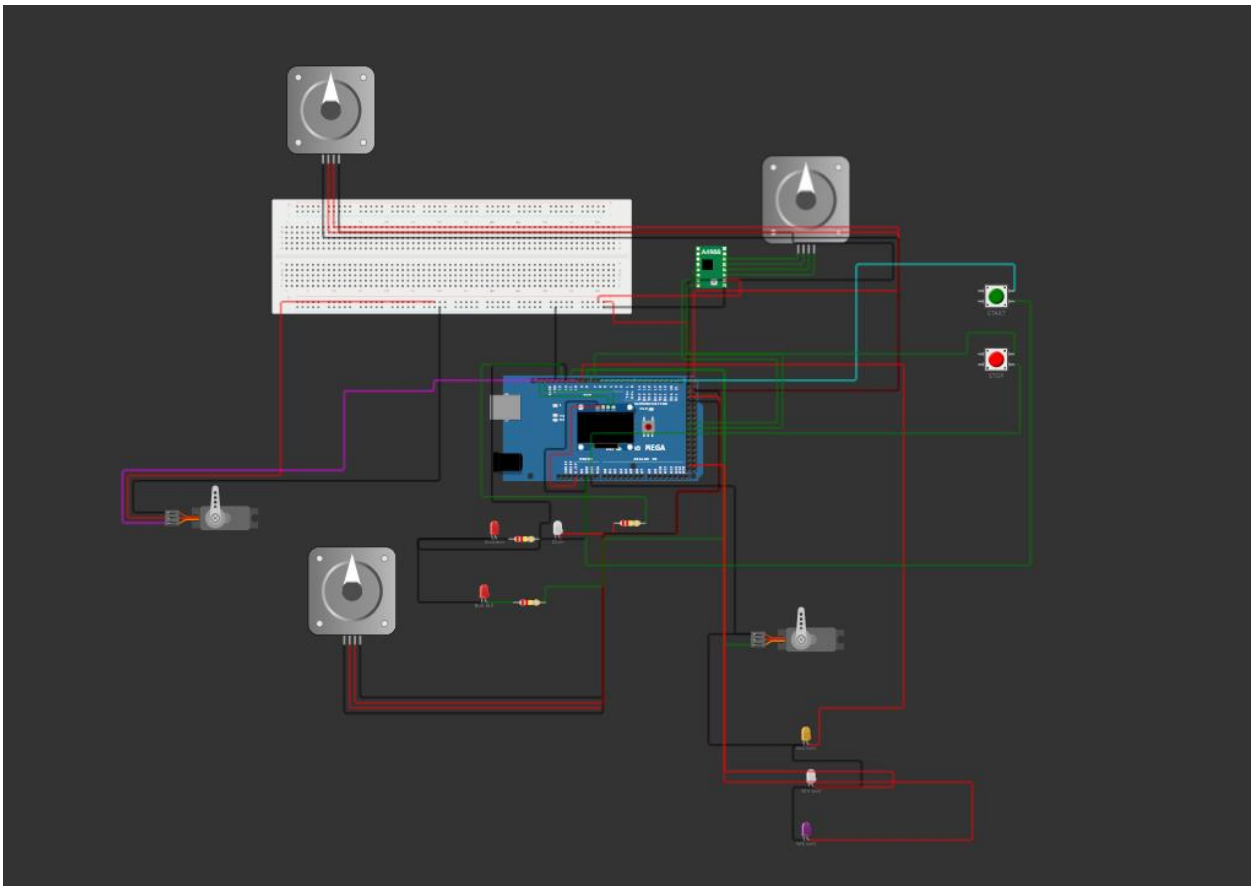

Embsystems - [Embsystems.dev] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
Embsystems
stage1.c [*] stage2.c finalstage.c Makefile.win
27 void closeValve2() {
28 // Close Valve2
29 printf("Valve2 closed.\n");
30 }
31
32 void startM5Conveyor() {
33 // Start M5 conveyor
34 printf("M5 conveyor started.\n");
35 }
36
37 bool isFilledPacketSensorFull() {
38 // Read the sensor value for filled packet storage
39 // Return true if full, false otherwise
40 return false; // Placeholder value
41 }
42
43 int main() {
44 if (isPacketAvailable()) {
45 openValve2();
46 int weight = readWeight();
47 while (weight < 5) {
48 weight = readWeight(); // Keep checking until it gets 5kg
49 }
50 closeValve2();
51 startM5Conveyor();
52 if (isFilledPacketSensorFull()) {
53 notifyWorkers();
54 } else {
55 // Wait until filled packet storage is full
56 while (!isFilledPacketSensorFull()) {
57 // Wait
58 }
59 }
60 } else {
61 notifyEmptyPackets();
62 }
63
64 return 0;
65 }
66
67
68
Compiler (46) Resources Compile Log Debug Find Results


Embsystems - [Embsystems.dev] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
Embsystems
stage1.c [*] stage2.c finalstage.c Makefile.win
#include <stdio.h>
#include <stdbool.h>
//combining both stages for the final stage
// Simulated functions for sensors and actuators
bool isStorageTankEmpty() {
return false; // Placeholder value
}
void notifyStorageTankNotFull() {
printf("Storage tank isn't full.\n");
}
void activateM1Conveyor() {
printf("M1 conveyor activated.\n");
}
void deactivateM1Conveyor() {
printf("M1 conveyor deactivated.\n");
}
bool isStorage1FiftyPercentFull() {
return true; // Placeholder value
}
void startM2ConveyorAndOpenValve1() {
printf("M2 conveyor started and Valve1 opened.\n");
}
bool isSteamerAtProperTemp() {
return true; // Placeholder value
}
void startSteamer() {
printf("Steamer started.\n");
}
void startDryer() {
printf("Dryer started.\n");
}
bool isStorage2Full() {
return false; // Placeholder value
Compiler (46) Resources Compile Log Debug Find Results


Embsystems - [Embsystems.dev] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
Embsystems
stage1.c [*] stage2.c finalstage.c Makefile.win
43 }
44
45 void notifyWorkers() {
46 printf("Notify workers.\n");
47 }
48
49 void stopSteamerAndDryer() {
50 printf("Steamer and dryer stopped.\n");
51 }
52
53 void closeValve1() {
54 printf("Valve1 closed.\n");
55 }
56
57 bool isPacketAvailable() {
58 return true; // Placeholder value
59 }
60
61 void notifyEmptyPackets() {
62 printf("No packets available.\n");
63 }
64
65 void openValve2() {
66 printf("Valve2 opened.\n");
67 }
68
69 int readWeight() {
70 return 5; // Placeholder value
71 }
72
73 void closeValve2() {
74 printf("Valve2 closed.\n");
75 }
76
77 void startM5Conveyor() {
78 printf("M5 conveyor started.\n");
79 }
80
81 bool isFilledPacketSensorFull() {
82 return false; // Placeholder value
83 }
84
Compiler (46) Resources Compile Log Debug Find Results


Embsystems - [Embsystems.dev] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
stage1.c [*] stage2.c finalstage.c Makefile.win
Embsystems
int main() {
// Stage 01
if (isStorageTankEmpty()) {
notifyStorageTankNotFull();
} else {
activateM1Conveyor();
if (isStorage1FiftyPercentFull()) {
deactivateM1Conveyor();
startM2ConveyorAndOpenValve1();
if (isSteamerAtProperTemp()) {
startSteamer();
startDryer();
if (isStorage2Full()) {
stopSteamerAndDryer();
closeValve1();
notifyWorkers();
}
}
}
}
// Stage 02
if (isPacketAvailable()) {
openValve2();
int weight = readWeight();
while (weight < 5) {
weight = readWeight(); // Keep checking until it gets 5kg
}
closeValve2();
startM5Conveyor();
if (isFilledPacketSensorFull()) {
notifyWorkers();
} else {
// Wait until filled packet storage is full
while (!isFilledPacketSensorFull()) {
// Wait
}
}
} else {
notifyEmptyPackets();
}
return 0;
Compiler (46) Resources Compile Log Debug Find Results

2.3 Demonstrate your embedded system selecting suitable hardware and software



Hardware selection

- **Microcontroller Unit (MCU)** – Use a power efficient and good performing microcontroller such as the STM32F4 series which are included in the ARM Cortex-M microcontroller family.
- **Motors** – Use suitable motors (M1 – M6) for managing the delivery to the conveyors (C1 – C5) and to control the movement of the container (C6).
- **Conveyor belts** – Implement resilient and efficient conveyor belts (C1 – C5) which have the proper lengths and weight capacity to transport the grains.
- **Container** – Install a container large enough to accommodate the weight and capacity of a 100 packets of grain.
- **Sensors** – Install different types of sensors which detect weight, temperature and proximity sensors to make sure the system runs accurately.

- **Control Panel** – Design and implement a user friendly and intuitive control panel for operators to monitor and control the system with various buttons, levers and user interface (UI).

Software Components

- **Firmware development** – Using a programming language such as C++ or Python are efficient and well supported for microcontroller programming allowing direct access to hardware resources with minimal overhead.
- **Real-Time Operating System (RTOS)** – An RTOS enable multitasking, allowing the system to handle multiple tasks simultaneously such as monitoring sensors, controlling motors, etc.
- **Motor Control Algorithms** – Develop a code which is needed in order for the motors to run on precise speed and to ensure consistent conveyor belt movement.
- **User Interface (UI)** – A user-friendly interface on LCD or OLED display allows operators to monitor the system's status.
- **Safety Mechanisms** – Software-based safety mechanisms ensure that the system can automatically halt operation in case of an emergency or hardware malfunction.