i): According to the question, the **controlled (real world) variables** in this computer system includes: data on active Ringing **Alarm Bell** or not, data on active **Relay Locked** or not and data on active **Light** of Segment 1 – 10 lights or not. For the **Ringing Alarm Bell**, its actual outputs from the computer is Alarm variable set be true, which is a binary data and represent the actual load weight close to limited load weight. Once the actual load exceed load limits, the alarm variables would be set to true and activate built-in alarm sounder in panel. For the **Relay Locked**, its actual outputs from the computer is Relay variables set be true, which is a binary data and control the control function interlock relays. Once the device in a dangerous situation, its value set to ‘0’, and stop operation. As the result, when relay variables is ‘1’, the operation could be operated, and fail stopped. For the **Lights of Segments**, their actual output from the computer is Segment variables set be true, which are binary data and control the lights on the panel which could represent the performance situation. These lights could be make user know the device performance situation obviously.

The **Alarm** Variable is the **critical** **safety** **information** because once this variable become true, the software would interlock control function. Furthermore, this variables would raise the alarm user by unable operate, and inform user the crane was load the materials which close to its limited load weight. This is a main part of safety approach on protection overload operation. Furthermore, the **hazardous failure modes** focus on identify failure modes that would adversely affect overall system reliability. As the result, for the Relay Locked, once the variables are not set to low at the same time because of loss of output, the whole control functions in the crane would not enable at the same time, which would effect on the stable of crane performance. At the same time, for the Alarm Bell, once the variables are not set to true when facing the overload situation because out of tolerance, the bell would not ring and the staff would not realize the overload situation. Moreover, the **Relay Variables** would be transform at the highest **priorities** in the computer system, because of its importance on protection of dangerous operation. The Relay Variables were a serious output, and go to set the relative control function to lock. Once the output is not continuous, or leak on the transform, not all control function would be locked. This kind of situation would effect on the crane effectively because of this asynchronous interlock actions.

ii): According to the question, the **monitored (real world) variables** is in the environment that system observes and respond to. As the result, in this system, the **monitored variables** in computer system inputs include: data of **Jib Length**, **Jib Elevation Angle**, **Hydraulic Ram Pressure**, **Cable Under-Wind or not**, **Cable Wound Out**, **Load Weight**, **Active** **Outrigger 1 – 4 Deployed or not, Fly Jib Rigged, Active** **Button 1 – 4 Input** and **Active** **Override Button Holding**. For **Jib Length**, the jib length would be measured by drum and cable mechanism on side of jib, and converted to digital by input circuitry. For **Jib Elevation Angle**, it would be measured by gear-driven potentiometer from jib pivot, and converted to digital by input circuitry. For **Ram Pressure**, it will measured by pressure transducer in hydraulic system, and converted to digital by input circuitry. For **Cable Under-Wind or not**, it will set the Two Block to 0 to represent cable under-wind is true. For **Cable Wound Out**, the condition sensed by micro switch activated by follower on threaded rod driven from drum. The variable would be set to 0 to represent to cable out is true. For **Load Weight**, it will measured by load cell, and converted to digital by input circuitry. For **Outrigger 1- 4**, they would be sensed by micros witch in outrigger leg mechanism. For **Fly Jib Rigged**, it would sense by micro switch on fly jib mounting plate. For **Button 1-4** and **Override Button**, they would get the input data from panel.

For this system, the **Jib Elevation Angle, Jib Length, Fly Jib Position** **Override Button Holding** and **Load Weight** are the **critical** safety **input information**. For the former three variables, they could be used to calculate to the **effective radius** of crane, which it is an extremely signification parameter to alarm user. Furthermore, the Load Weight is also a critical information which could be compare with the crane limited load weight, and feedback the alarm information to user or not. This input relative to the core safety software function, and should be monitored all the time. In general, it is better the make these **input** could be **respond** by software in real time. However, it is hard to respond to asynchronous event and data all the time. As the result, the **response time parameter** setting determines the maximum reaction time based on the crane work situation. Consider of the computer system performance, several input should be response immediately after getting. At the same time, several characters which would display on the screen would also effect on the work performance of the crane. For the **Load Weight** and **Override Button Holding** (relative to critical safety input), they should be reflect immediate when test the overload, and the reaction time should be greater than 100 revolutions per minute, which could make sure the load weight input would be get immediately. For the **Jib Elevation Angle**, **Jib Length**, **Fly Jib Position**, the change of these parameters are based on the crane operation, and always *slow*. As the result, the reaction time should be greater than 10 revolutions per minute. For the rest input parameters, which not relative to the critical safe, their reaction time should be greater than 5 revolutions per minute.

For the input event, it may possible to have **fault** on **input operation**. The **mistake** on **operation** would active **buttons** on the inappropriate situation. At the same time, the **errors** on **measurement devices** may raise problem on input fault. For the **errors** on **measurement devices**, these errors may be detected when the overflow in range of data. For example, the Jib Elevation Angle become negative. As the result, the measurement devices should **test** by a monitor in its measurement range before performance in real world. For the mistake on operation button, it would be detected by button pressure period time from monitors. For example, the **Hold Button** would active load over limited weight on holding this button by monitor device, and enable critical outputs. As the result, these fault could be detected. When these fault has been detected, the software will not change the input data variables.

In the computer system, the **Load Weight** information input should be transform prioritise when consider of the crane performance situation. The crane’s main task is load materials, however, load over limited weight would cause damage on crane, and effect on the quality of performance. As the result, the load weight information input should be input prioritise to avoid damage.

iii) In this problem, Check-Weight 2000 change its states during performance by user operations. The processes of state situation changing will be represented by state transition diagram. **State transition diagram** is a directed graph representation of system states, transitions between states, and transition rates. These diagrams contain sufficient information for developing the state equations, which are used for probability calculations. The state transition diagram is the backbone of the technique. In the Check-Weight case, it will only be represented with the relationship between state of system and user’s action.

Firstly, **define the system**, which includes examine the system and define the system boundaries, subsystems, and interfaces. In this case, the system **starts with** load weight, **end with** crane stop working. In its performance, it has an **interface** on user to **operation holding button** action to continue or end performance. The next task is **identify the system states**, which aims to establish the goals of the system and determine the system and component states of interest. In this case, it has state includes: **Crane Working (Green Lights Light)** (Guard: load weight, less than warning limited), **Crane Working (Amber Lights Light)** (Guard: load weight, greater than warning limited and less than limited), **Control Locked** (Guard: Amber Lights Light).Finally, construct the state diagram for all of the identified system states. The following graph is the state transition diagram (in this case, assume the limited weight load would cause crane stop working).

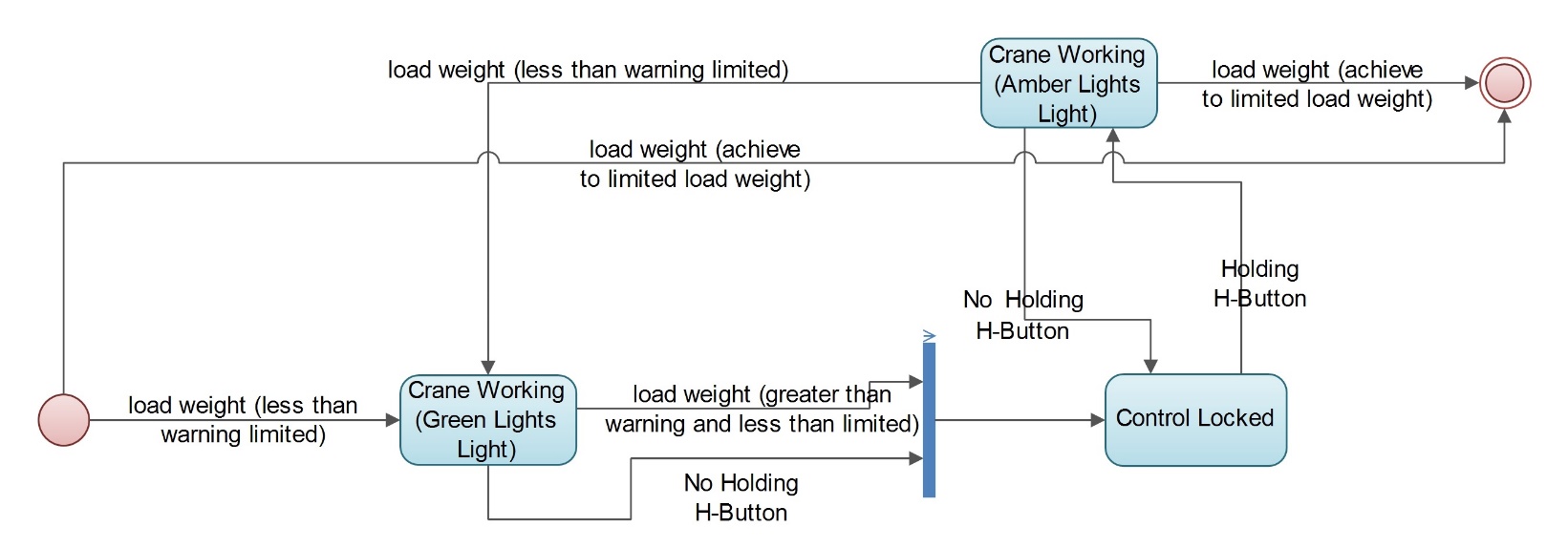


Figure :

The load weight (less than warning limited), load weight (greater than warning and less than limited), Holding holding-button and No Holding holding-button are **normal operations** if the system. The load weight (achieve to limited load weight) represent **failure detection**, and make the system to stop perform.

iv):

According to the question description, the Coyote Technology’s software engineers propose to implement the controller software primarily in MISRA C, with code written in assembly language. At the same time, put all the functions to be called in a single main loop. Moreover, the functional code could be reused by add a hardware abstraction layer.

This design have **positive** characters in the system, on **hardware** and **software cost**. In the design, the user only need to buy hardware devices, which **not** require **high property device** to run operation system, to fulfil the requirement of system. At the same time, the **hardware abstraction layers** plays the role of **interface of program**, which increase the code reused on different hardware without modification. However, this kind of design still have **negative** part. Firstly of all, all functions will run in a **main loop**, which means all functions will run one by one. If one **a function** in the first location **failed**, then **other functions** will **not perform**. As the result, once one method cannot perform, the rest will be effect on. Moreover, this kind of program performance based on the **correction of code** and **correction** of hardware device **installation**. Once the hardware devices have mistaken, the programs will not perform correctly.

In my mind, the design on **implement interface** code for some of the **hardware devices** and **adding abstraction layer** (isolate from details of computing hardware) are the effective and reasonable approach for this system design. However, run all methods in a **main loop** is a bad design approach, because it **increase** the **probability** of functions **fail** in the programs, and make user **hard** to **detect** the **failure** on the devices.

v):

In this system’s design, it separate the code from configuration data, which refers to the arrangement of components to form a system. In the configuration item in computer software, it **aggregates** of software that **satisfies** and **end-use function** and is designated for **separate** configuration management by the **developer** or **acquirer**. The configuration date are selected based on **trade-offs** among software function, size, host or target computers, developer, support concept, plans for reuse, criticality, interface considerations, need to be separately documented and controlled, and other factors. As the result, it should modified based on the requirements. In this case, the *company’s intention is that the core software should never change; customisation for each model of crane to which the system is fitted will be achieved by changing the calibration parameters (mostly look-up tables mapping sensor values to engineering units) in the configuration EPROM*. Therefore, separate the code from configuration could separate the core function and configuration information, and there is no need to modify core software when configure software. For the **developer**, they could focus on the **core software development**. For installation software on **different platform**, the **project engineers** could focus on the **data configuration** based on customer’s requirement.

In this system, it is a **sensible** because it **fulfil** the **requirement** of company intention. According to the development of this company, the managers pay attention to selling the safety management software to different devices. As the result, the software should adopt to different kind of hardware platform. After separate the core software code from data configuration, the core software development and update could be separate from project performance. For example, once the core software should update, the engineer only need to update the core software code, ignore the configuration data. This **reduce** the **probability** of software **failure** in **updating** and **improve** the **stability** of software.