Technical University of Denmark

31342 Introduction to Programmable Logic Controllers

Exercise 7

Conveyor

Student:

Sefa Kayraklık (s186295)

Instructor:

Søren Hansen

Spring, March 31, 2019

Task 1: The Control System to a Paint Sprayer:

The implementation of the control system can be done by using sequential function chart (SFC) along using structured text in the states. In the system, the detector works as an input, and the sprayer works as an output. Also, an extra input is needed for the transition condition.

There are 2 states in the system, which are responsible for running the system and stopping the system. The extra input is named as *moveRight*. It is high when the system is wanted to move right; otherwise, it is low. Also, there are 2 intermediate variables which keep the information about the 6 hooks that are between the detector and the sprayer: They are 6 bits and named as *in* for the input of the shift register and *out* for the output of the shift register. The figure 1 shows the SFC of the control system.

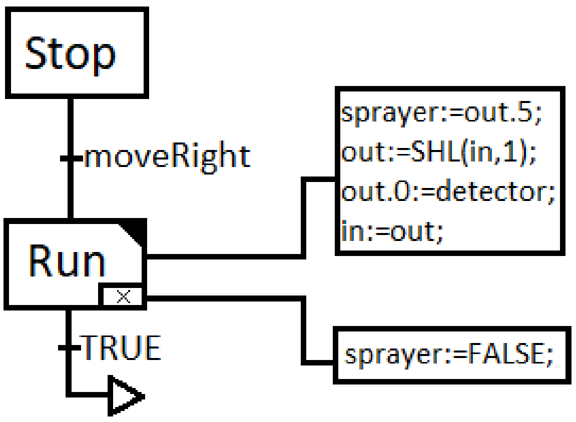


Figure 1: The SFC of the System

The system works as in the following way:

- *out.5* is the hook that is present in front of the sprayer. So, if it is TRUE then sprayer will work; otherwise, it won't.
- Then, *in* is shifted to the left by one.
- *out.0* is the hook that is present in front of the detector. If the detector detects an item, then its value becomes TRUE.
- *in* is equal to the 6 hooks before being moved to the right.
- The exit action of the run state is turning off the sprayer.

The block diagram of the overall system is shown in the Figure 2.

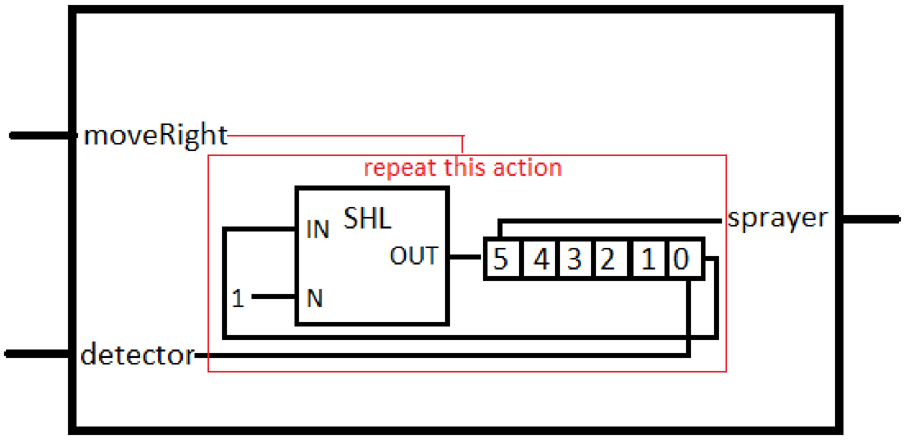


Figure 2: The Block Diagram of the Overall System

Task 2: Conveyer Belt Setup:

The implementation of the convey belt setup can be done by using sequential function chart (SFC) along using structured text in the states. The inputs and the outputs are used as defined in the description of the question. And there is an extra intermediate variable named *continue*, which tells whether the system continues to release a box or not, and the default value of it should be TRUE.

There are 3 states in the system: the initial state, the run state and the stop state. As long as the *continue* variable is TRUE, the system delivers the boxes to the belt.

- In the initial state, the system is in rest.
- In the run state, the system releases one box to the belt, the detector detects the failed items, the gripper removes the failed items. Namely, it is the main state in which all the required tasks are done. The system returns to the initial state if the *do0* becomes to low.
    - Firstly, the input gate is implemented in a way that *do0* is a pulse that longs 1 second. There is a TON timer that is named as *inputgate*.
      The assignment of the *do0* is done by following:

        -
          ```
          inputgate(IN:=TRUE, PT:=t#1s);
          do0:=NOT inputgate.Q;
          ```
    - Secondly, the motor is driven by the distance sensor's output. Since the distance sensor gives out a signal of 7V when no item is present, the motor drivers the conveyer belt in 70cm/s speed; when there is an item in the end of the belt, the motor slows down in order not to throw the item out the output box. Namely, the belt is driven in 70cm/s speed. (The slowing part can be ignored since it doesn't last so long.)
      The assignment of the *ao0* is done by following:

        -
          ```
          ao0:=ai0;
          ```
    - Thirdly, the fail detector detects the failed item and it is written on a 3 bits variable. (There can be only 3 items in between the detector and the gripper.) Also, there is 70 cm spacing between each item since in every 1 second, 1 item is being released. An item will be present in front of the detector after 5/7 second (400-70*5=50; 50/70 sec; there are 7 items before the detector and the 8th item will be present after 5/7 sec); similarly, an item will be present in in front of the gripper after 2/7 second (580-70*8=20; 20/70). There are 2 TON timers that are named as *detector* and *gripper*.
      The assignment of the *do1* is done by following:

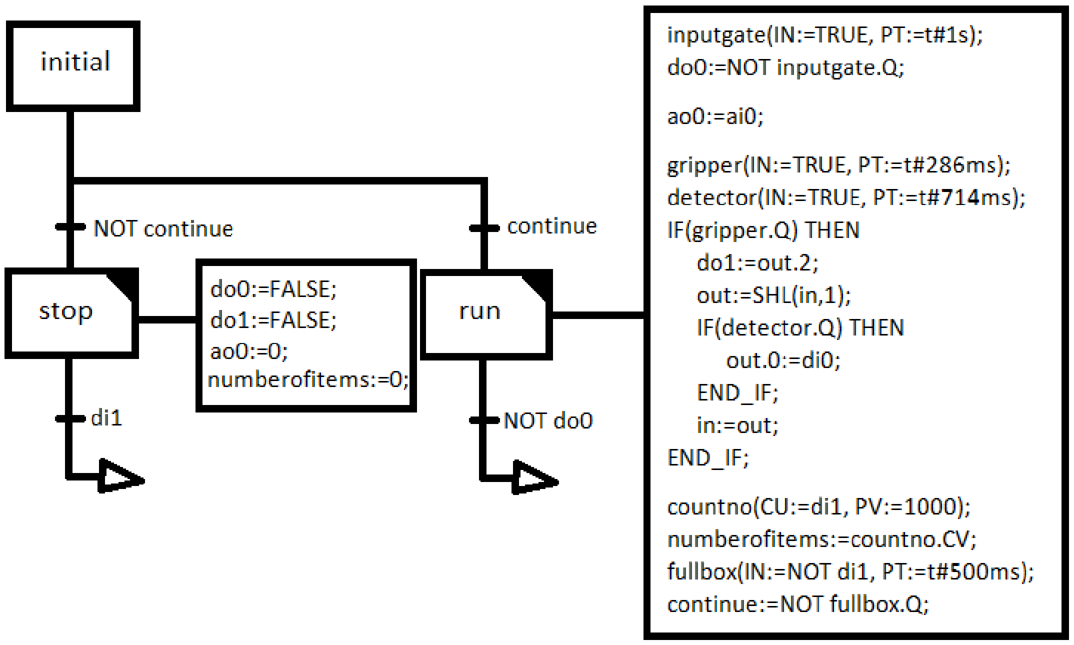
        -
          ```
          gripper(IN:=TRUE, PT:=t#286ms);
          detector(IN:=TRUE, PT:=t#714ms);
          IF(gripper.Q) THEN
              do1:=out.2;
              out:=SHL(in,1);
              IF(detector.Q) THEN
                  out.0:=di0;
              END_IF;
              in:=out;
          END_IF;
          ```
    - Finally, the light receiver determines whether the output box is full or not. As long as the output box is not full, the *continue* variable stays TRUE. The *continue* variable goes low after 0.5 second (an arbitrary choice) not receiving light in the light receiver. Also, the number of items that the output box has is counted via a CTU counter that is named as *countno*. There is a TON timer that is named as *fullbox*.
      The assignment of the *continue* and *numberofitems* is done by following:

        -
          ```
          countno(CU:=di1, PV:=1000);
          numberofitems:=countno.CV;
          ```

```
fullbox(IN:=NOT di1, PT:=t#500ms);
continue:=NOT fullbox.Q;
```

- The system will enter into the stop state when the output box is full. And all the outputs in the system will be assigned FALSE or 0. In order to activate the system again, the input *di1* should be TRUE meaning that the output box is replaced, and it receives the light again.

The Figure 3 shows the SFC of the conveyer belt.



```
do0:=FALSE;
do1:=FALSE;
ao0:=0;
numberofitems:=0;
```

```
inputgate(IN:=TRUE, PT:=t#1s);
do0:=NOT inputgate.Q;

ao0:=ai0;

gripper(IN:=TRUE, PT:=t#286ms);
detector(IN:=TRUE, PT:=t#714ms);
IF(gripper.Q) THEN
    do1:=out.2;
    out:=SHL(in,1);
    IF(detector.Q) THEN
        out.0:=di0;
    END_IF;
    in:=out;
END_IF;

countno(CU:=di1, PV:=1000);
numberofitems:=countno.CV;
fullbox(IN:=NOT di1, PT:=t#500ms);
continue:=NOT fullbox.Q;
```

Figure 3: The SFC of the conveyer belt

The block diagram of the overall system is shown in the Figure 4.

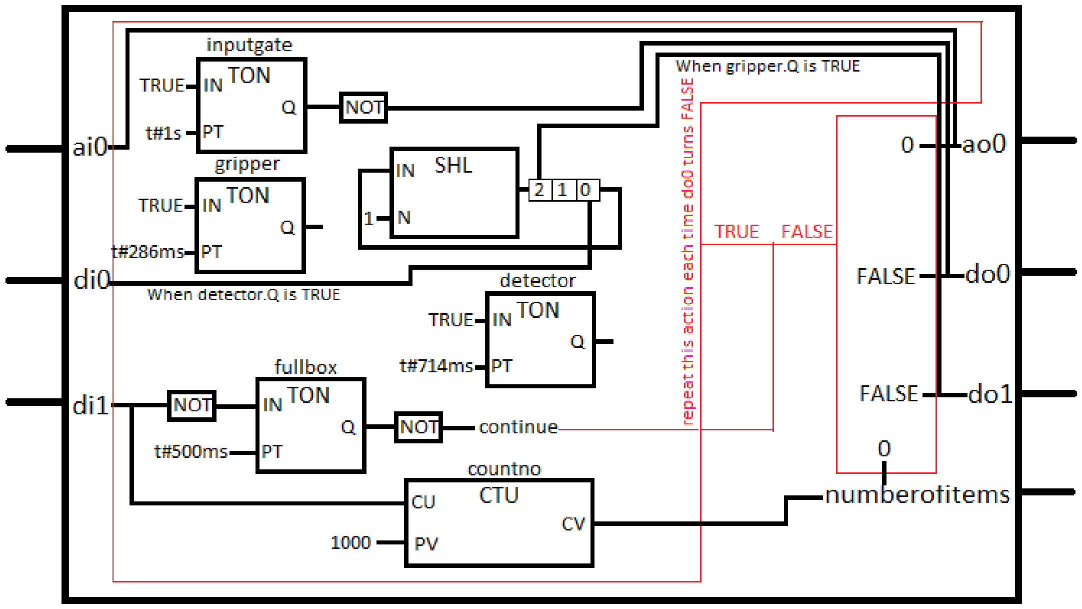


Figure 4: The block diagram of the overall system

If the variable *continue* is FALSE, then the system reaches the stop state: (Figure 5)

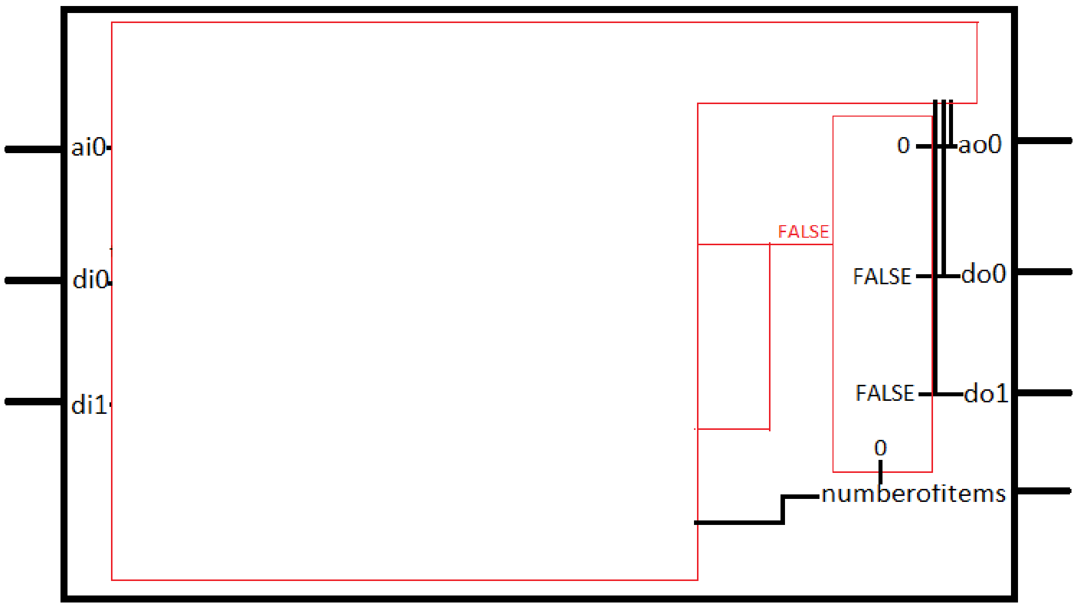


Figure 6: The block diagram of the stop state

If the variable *continue* is TRUE, then the system reaches the run state: (Figure 6)

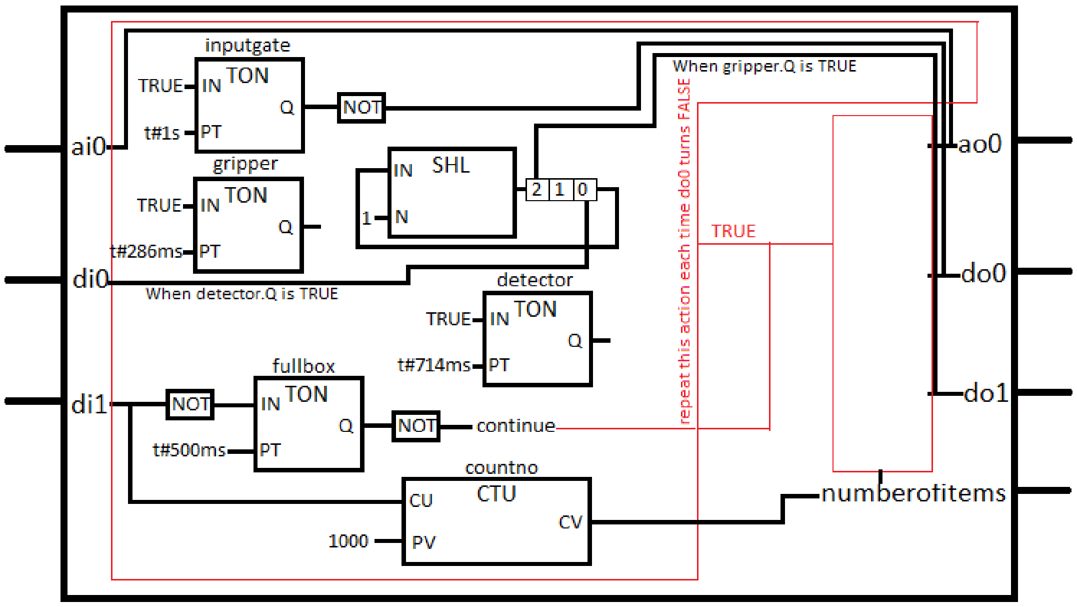


Figure 5: The block diagram of the run state