Rudra Goel
Lab 06 Report
ECE 2031 L10
06 October 2024

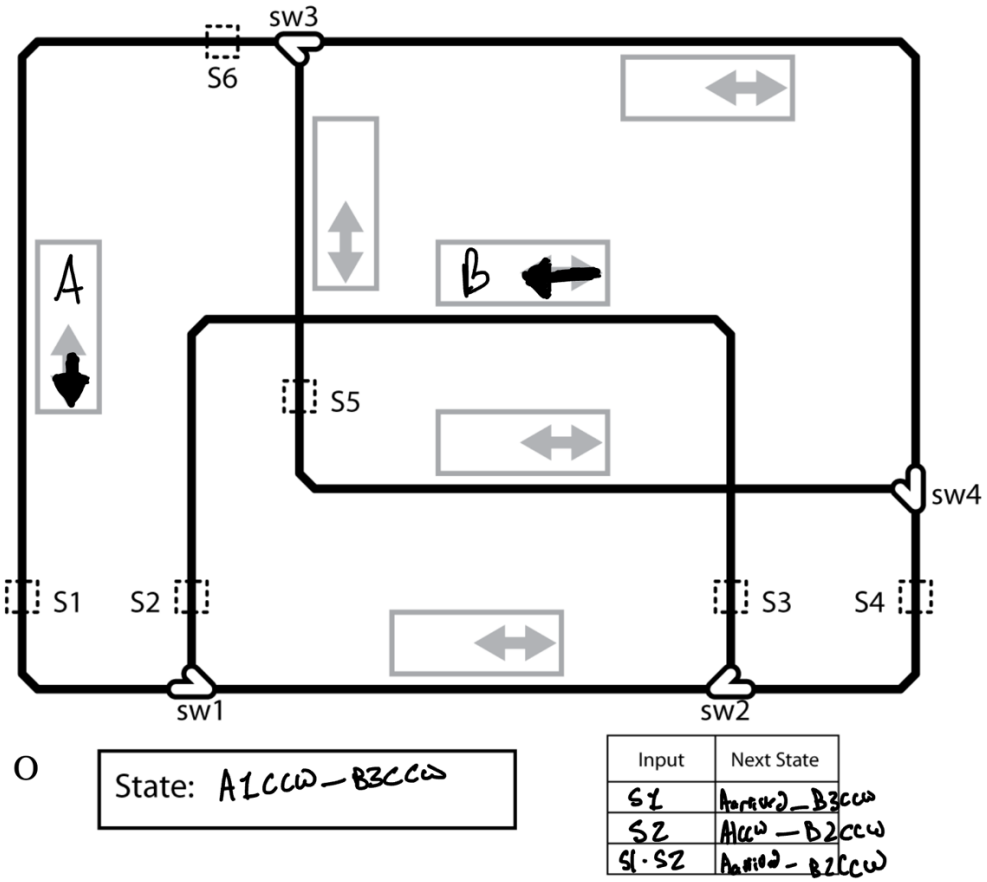

Figure 1. Worksheet diagram of state A1CCW_B3CCW where Train A is on track 1 going counterclockwise, and Train B is on track 3 going counterclockwise. At this state, the switches do not need to be asserted to any logic level. When sensor 1 (S1) is asserted, state transitions to Aarrived_B3CCW; when sensor 2 (S2) is asserted, state moves to A1CCW_B2CCW; when both sensors are asserted ($S1 \cdot S2$), state transitions to Aarrived_B2CCW.

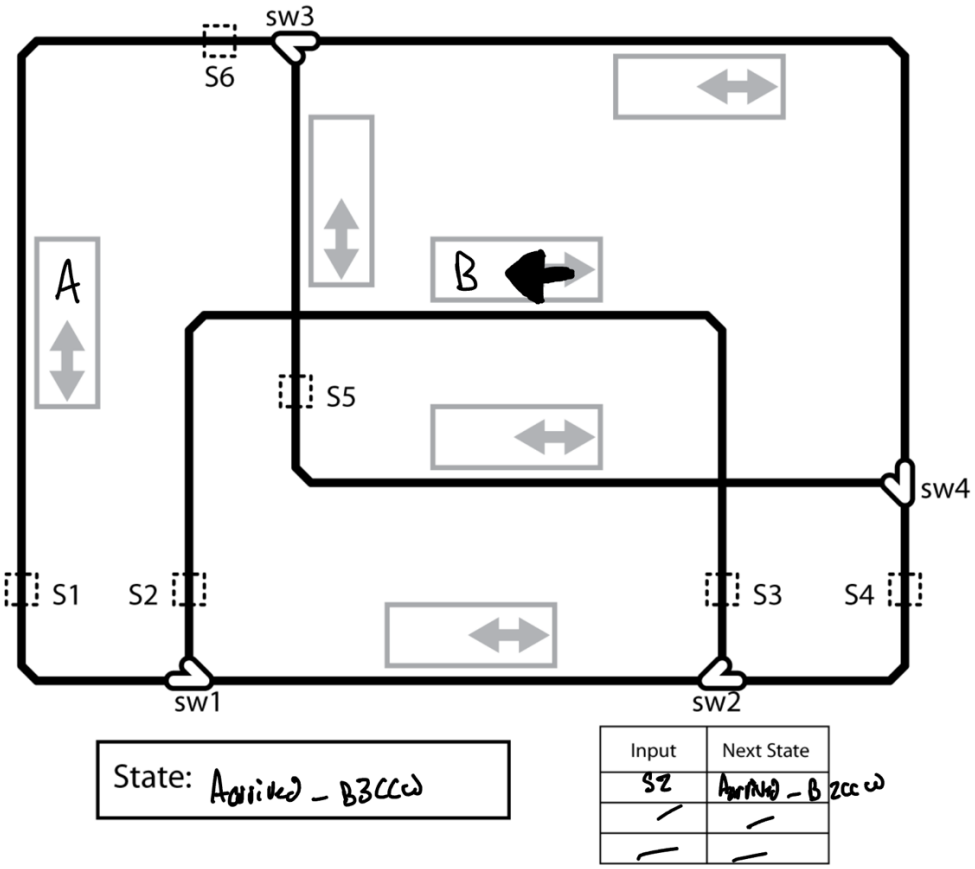

Figure 2. Worksheet diagram of state Aarrived_B3CCW where Train A is stopped at sensor 1, and Train B is on track 3 going counterclockwise. At this state, the switches do not need to be asserted to any logic level. When sensor 2 (S2) is asserted, state moves to Aarrived_B2CCW.

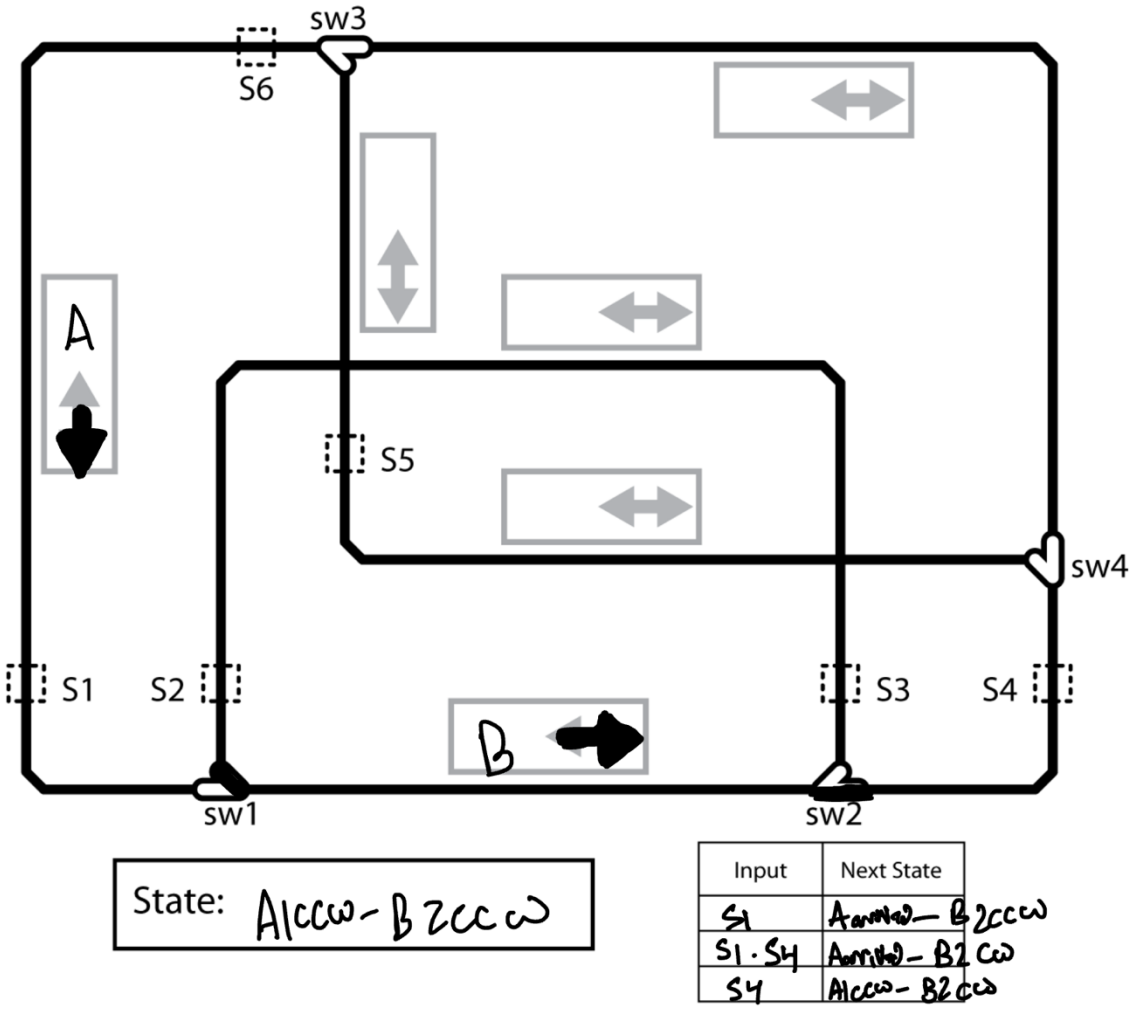

Figure 3. Worksheet diagram of state A1CCW_B2CCW where Train A is on track 1 going counterclockwise, and Train B is on track 2 going counterclockwise. Switch 1 is asserted logic high to connect track 3 to track 2. Switch two is asserted logic low to keep track 2 connected with itself. When sensor 1 (S1) is asserted, state transitions to Aarrived_B2CCW; when sensor 4 (S4) is asserted, state moves to A1CCW_B2CW; when both sensors are asserted ($S1 \cdot S4$), state transitions to Aarrived_B2CW.

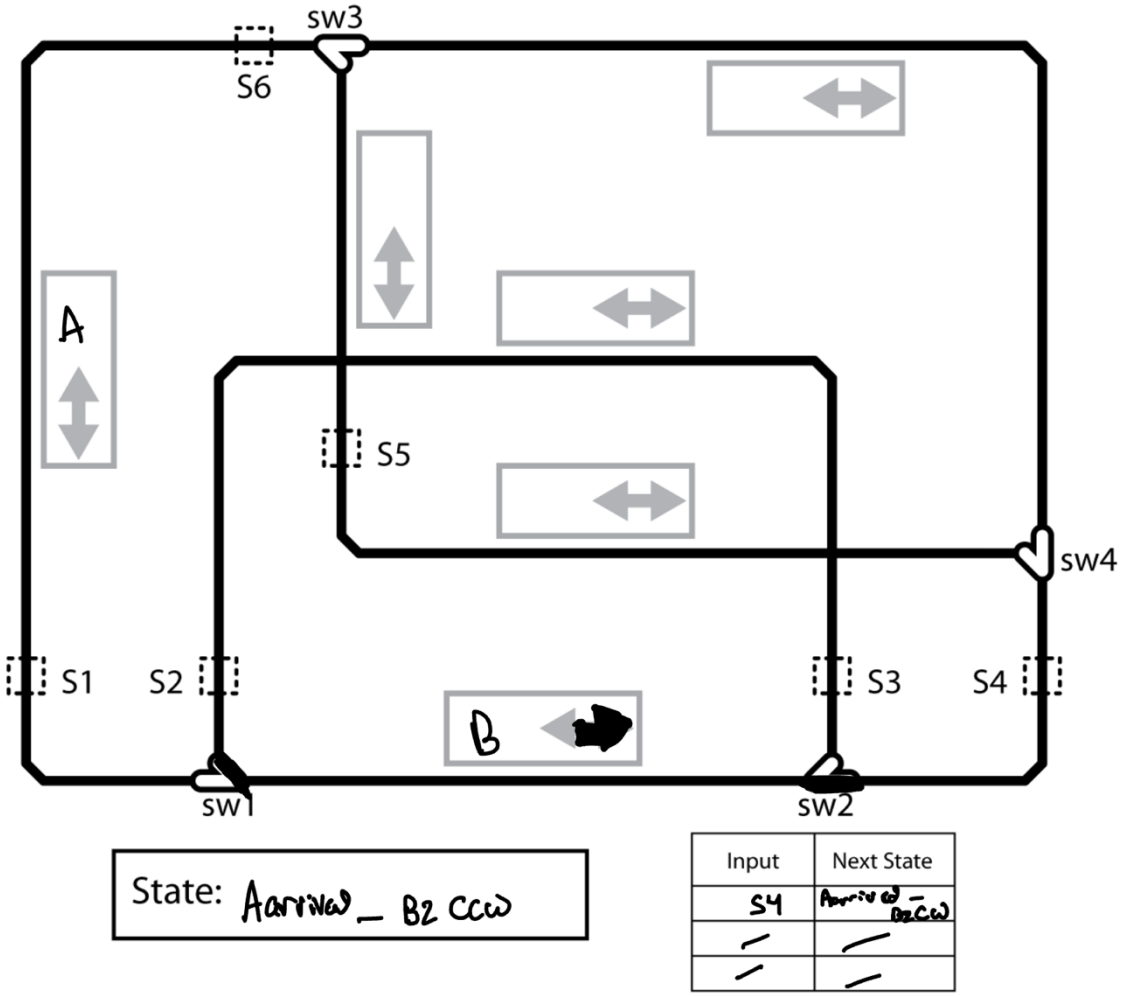

Figure 4. Worksheet diagram of state Aarrived_B2CCW where Train A is stationary at sensor 1, and Train B is on track 2 going counterclockwise. Switch 1 is asserted logic high to connect track 3 to track 2. Switch 2 is asserted logic low to keep track 2 connected. When sensor 4 (S4) is asserted, state moves to A1CCW_B2CW.

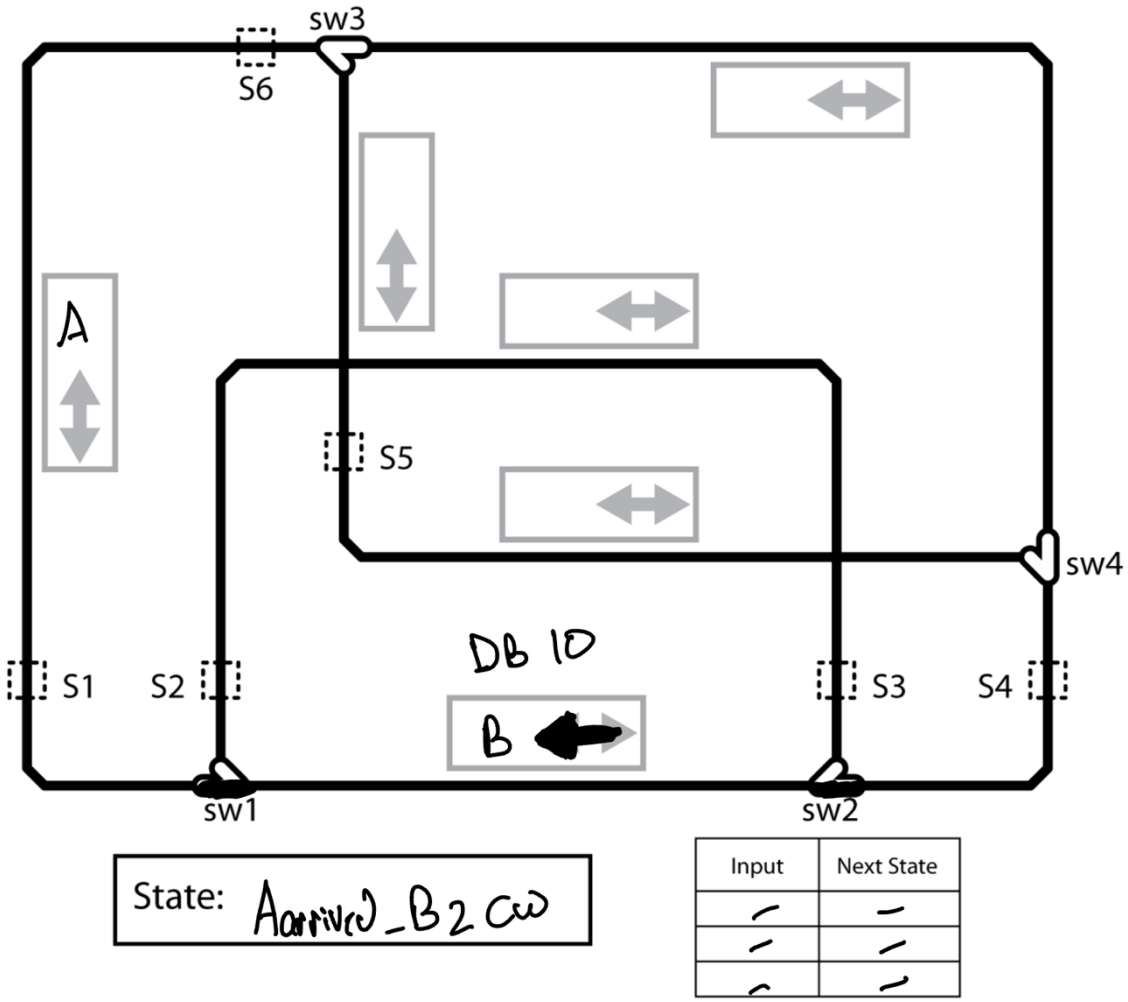

Figure 5. Worksheet diagram of state Aarrived_B2CW where Train A is stationary at sensor 1, and Train B is on track 2 going counterclockwise. Switch 1 is asserted logic low to connect track 1 to track 2. Switch 2 is asserted logic low to keep track 2 connected with itself. There is no next state as this state is certain to crash the trains at sensor 1.

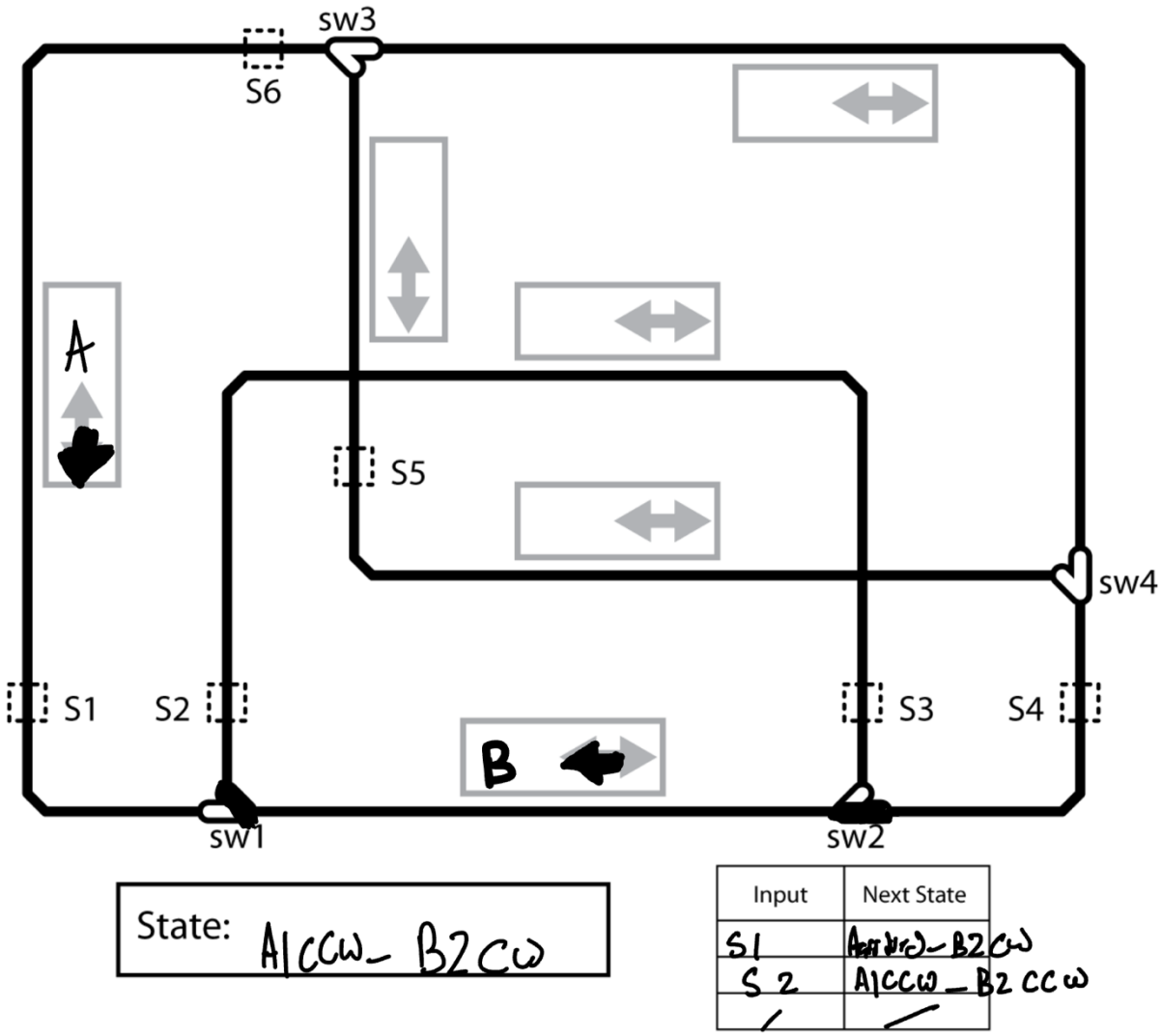

Figure 6. Worksheet diagram of state A1CCW_B2CW where Train A is on track 1 going counterclockwise, and Train B is on track 2 going clockwise. At this state, switch 1 is asserted to logic high to connect track 3 to track 2. Switch 2 is asserted logic low to keep track 2 connected with itself. When sensor 1 (S1) is asserted, state transitions to Aarrived_B2CW; when sensor 2 (S4) is asserted, state moves to A1CCW_B2CCW.

Table 1
Table Of Outputs For Train Controller State Machine

Outputs	States							
	A1CCW – B3CCW	Aarrived – B3CCW	A1CCW – B2CCW	Aarrived – B2CCW	Aarrived – B2CW	A1CCW – B2CW	Not used	Not used
Sw1	X	X	1	1	0	1	X	X
Sw2	X	X	0	0	0	0	X	X
Sw3	X	X	X	X	X	X	X	X
Sw4	X	X	X	X	X	X	X	X
DA[1..0]	01	00	01	00	00	01	XX	XX
DB[1..0]	01	01	01	01	10	10	XX	XX

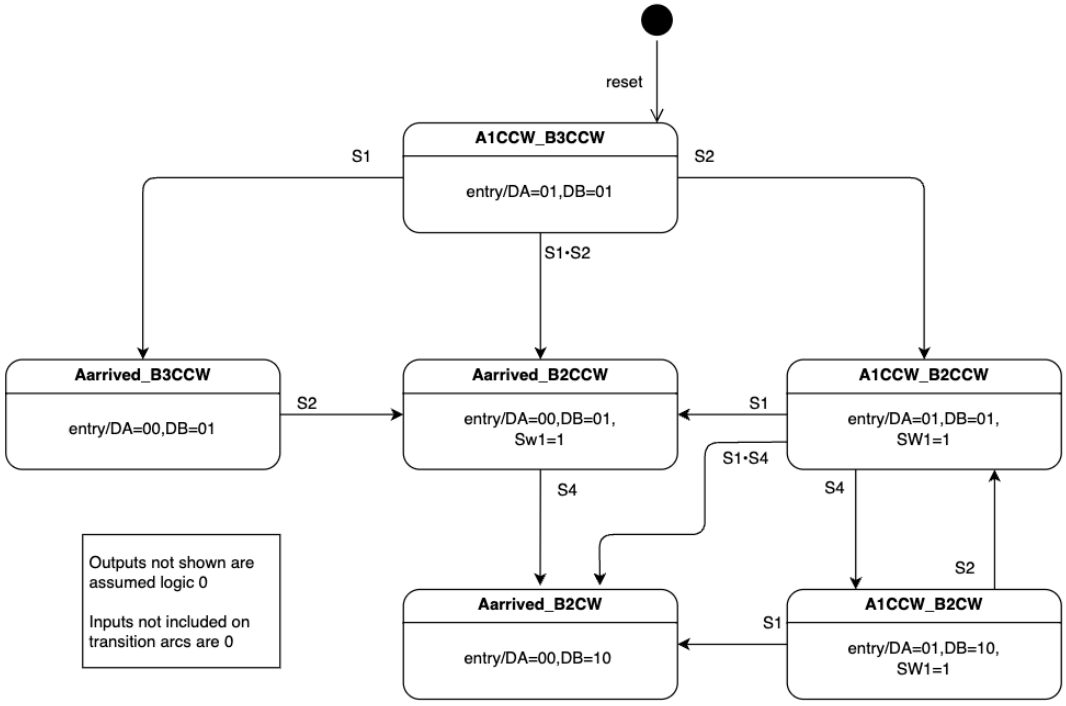

Figure 7. UML state diagram for Train Controller state machine meant to crash both trains at sensor 1. Initial state marked by black dot is A1CCW_B3CCW. Crash state is Aarrived_B2CW. Train B bounces back-and-forth on track 2 until Train A arrives at sensor 1 where it then travels clockwise on track 2 onto track 1 to crash at sensor 1.

Appendix A
VHDL Code Implementing Train Controller Finite State Machine

```
-- State machine to control trains, adapted from class template
-- Author: Rudra Goel
-- Date 10/3/2024

LIBRARY IEEE;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_ARITH.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;


ENTITY TrainController IS
     PORT(
          reset, clock, sensor1, sensor2      : IN std_logic;
          sensor3, sensor4, sensor5, sensor6  : IN std_logic;
          switch1, switch2, switch3, switch4  : OUT std_logic;
          dirA, dirB                          : OUT
std_logic_vector(1 DOWNTO 0)
     );
END TrainController;


ARCHITECTURE a OF TrainController IS
     -- Create a new TYPE called STATE_TYPE that is only allowed
     -- to have the values specified here. This
     -- 1) enables using helpful names for values instead of
     --    arbitrary values
     -- 2) ensures that signals of this type can
     -- only have valid values, and
     -- 3) helps the synthesis software create
     -- efficient hardware for the design.
     TYPE STATE_TYPE IS (
          A1CCW_B3CCW,
          A1CCW_B2CCW,
          A1CCW_B2CW,
          Aarrived_B3CCW,
          Aarrived_B2CCW,
          Aarrived_B2CW
     );
     -- Create a signal of the new type.
     SIGNAL current_state     :    STATE_TYPE;

     -- This creates some new internal signals which will be
     -- concatenations of some of the sensor signals.
     -- This will make CASE statements easier.

     SIGNAL sensor12, sensor14 : std_logic_vector(1 DOWNTO 0);
```

```
BEGIN
     -- This is the sequential logic portion that is clocked
     PROCESS (clock, reset)
     BEGIN
     IF reset = '1' THEN
          current_state <= A1CCW_B3CCW;
     ELSIF clock'EVENT and clock='1' THEN
          CASE current_state IS

          WHEN A1CCW_B3CCW =>
               CASE sensor12 IS

               WHEN "11" => current_state <= Aarrived_B2CCW;
               WHEN "10" => current_state <= Aarrived_B3CCW;
               WHEN "01" => current_state <= A1CCW_B2CCW;
               WHEN others => current_state <= A1CCW_B3CCW;

               END CASE;

          WHEN Aarrived_B3CCW =>
               CASE sensor12 IS

               WHEN "11" => current_state <= Aarrived_B2CCW;
               WHEN others => current_state <= Aarrived_B3CCW;

               END CASE;
          WHEN A1CCW_B2CCW =>
               CASE sensor14 IS

               WHEN "11" => current_state <= Aarrived_B2CW;
               WHEN "10" => current_state <= Aarrived_B2CCW;
               WHEN "01" => current_state <= A1CCW_B2CW;
               WHEN others => current_state <= A1CCW_B2CCW;

               END CASE;
          WHEN Aarrived_B2CCW =>
               CASE sensor4 is

               WHEN '1' => current_state <= Aarrived_B2CW;
               WHEN others => current_state <= Aarrived_B2CCW;

               END CASE;
          WHEN A1CCW_B2CW =>
               CASE sensor12 is

               WHEN "10" => current_state <= Aarrived_B2CW;
```

```
                WHEN "01" => current_state <= A1CCW_B2CCW;
                WHEN others => current_state <= A1CCW_B2CW;

                END CASE;
           WHEN Aarrived_B2CW =>

                current_state <= Aarrived_B2CW;

           END CASE;

     END IF;
END PROCESS;

     -- The following logic is NOT in a process block,
     -- and thus does not depend on any clock.
     -- Everything here is pure combinational
     -- logic, and exists in parallel with everything else.
     -- These outputs are constant values for this solution;

     Switch3 <= '0';
     Switch4 <= '0';

     -- Combine bits for the internal signals declared above.
     -- ("&" operator is concatenation)
     sensor12 <= sensor1 & sensor2;
     sensor14 <= sensor1 & sensor4;



     --since this is moore FSM,
     --the outputs only depend on the current state

     WITH current_state SELECT switch1 <=
          '1' WHEN A1CCW_B2CCW,
          '1' WHEN Aarrived_B2CCW,
          '0' WHEN Aarrived_B2CW,
          '1' WHEN A1CCW_B2CW,
          '0' WHEN others;
     WITH current_state SELECT switch2 <=
          '0' WHEN A1CCW_B2CCW,
          '0' WHEN Aarrived_B2CCW,
          '0' WHEN Aarrived_B2CW,
          '0' WHEN A1CCW_B2CW,
          '0' WHEN others;
     WITH current_state SELECT dirA <=
          "01" WHEN A1CCW_B3CCW,
          "00" WHEN Aarrived_B3CCW,
```

```
            "01" WHEN A1CCW_B2CCW,
            "00" WHEN Aarrived_B2CCW,
            "00" WHEN Aarrived_B2CW,
            "01" WHEN A1CCW_B2CW;
        WITH current_state SELECT dirB <=
            "01" WHEN A1CCW_B3CCW,
            "01" WHEN Aarrived_B3CCW,
            "01" WHEN A1CCW_B2CCW,
            "01" WHEN Aarrived_B2CCW,
            "10" WHEN Aarrived_B2CW,
            "10" WHEN A1CCW_B2CW;
END a;
```