Matthew Liu
Lab Report 6
ECE 2031 L09
27 February 2019

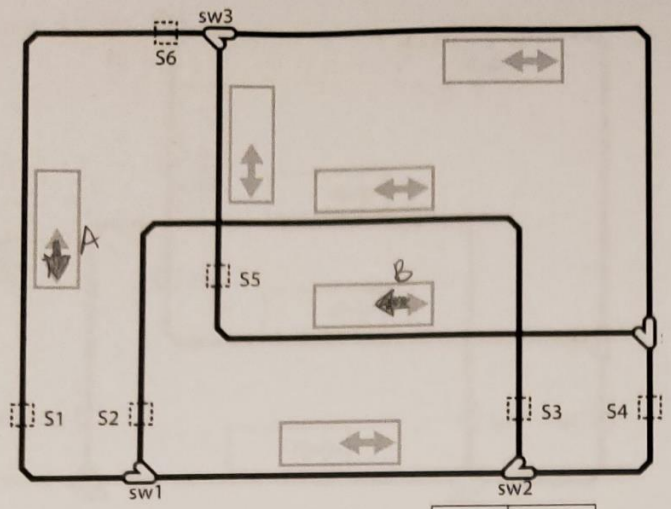
Outputs	States							
	A1B5	A4R6	A5 B1	A6 B4	ASTOP	Astop2	Bstop	Bstop2
Sw1	0	0	0	0	0	0	0	0
Sw2	X	X	X	X	X	X	X	X
Sw3	1	1	1	1	1	1	1	1
Sw4	1	1	1	1	1	1	1	1
DA[1..0]	1	1	1	1	0 0	0 0	0 0	0 1
DB[1..0]	1	1	1	1	0 1	0 1	0 0	0 0

← don't care always 0

Outputs	States							
	BothMove	BothMove1	BothMove2	AstopReset	BstopReset	AstopReset2	BstopReset2	
Sw1	X	0	1	1	0	1	0	
Sw2	X	0	0	0	0	0	0	
Sw3	X	0	1	1	1	1	0	
Sw4	X	0	1	1	1	1	0	
DA[1..0]	01	01	01	0 0	0 1	0 0	0 1	
DB[1..0]	01	01	01	0 1	0 0	0 1	0 0	

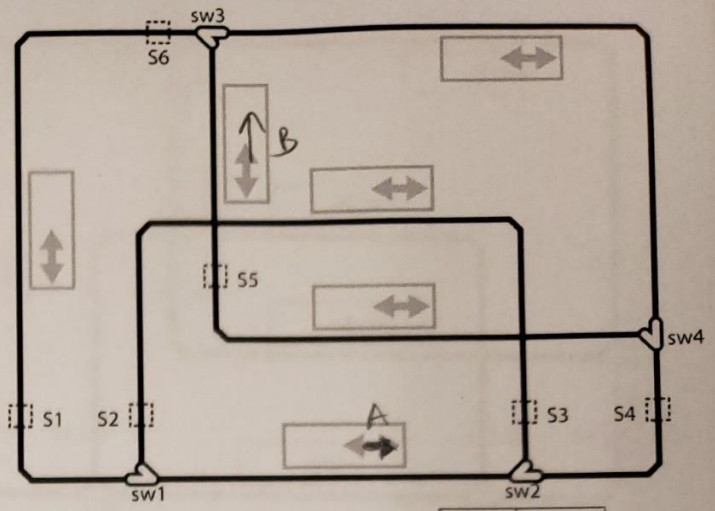
X = don't care

Outputs	States							
Sw1								
Sw2								
Sw3								
Sw4								
DA[1..0]								
DB[1..0]								



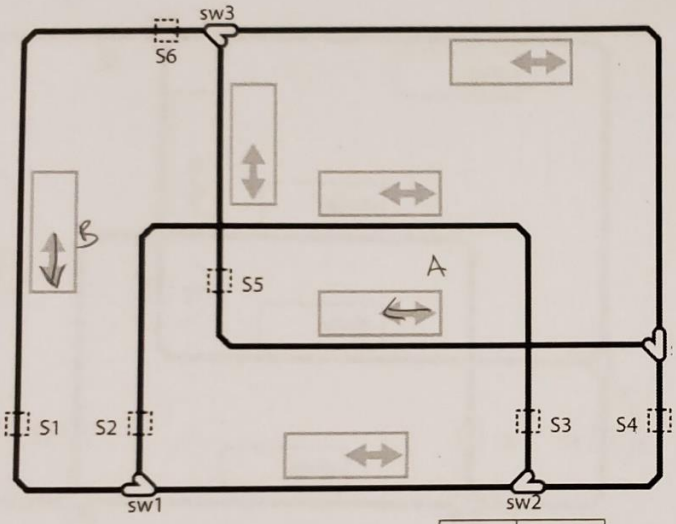
State: A1B5

Input	Next State
S5	Bstop
S1	Astop
S1 & S5	A4B6



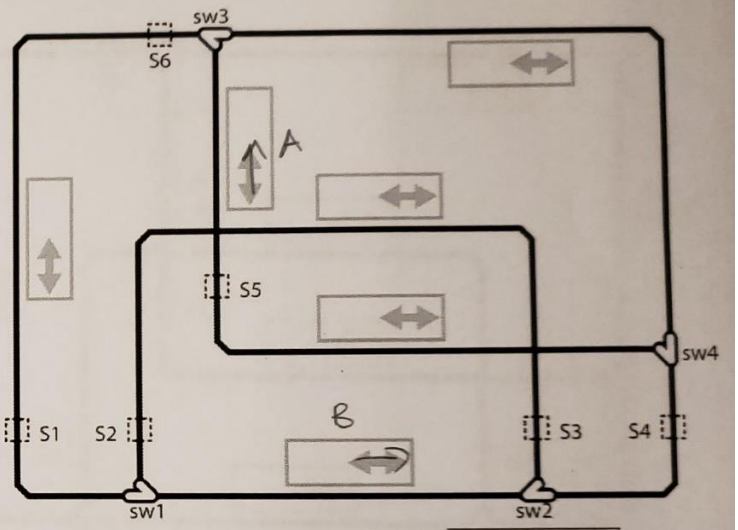
State: A4B6

Input	Next State
S4	Astop
S6	Bstop
B4 & S6	A5B1



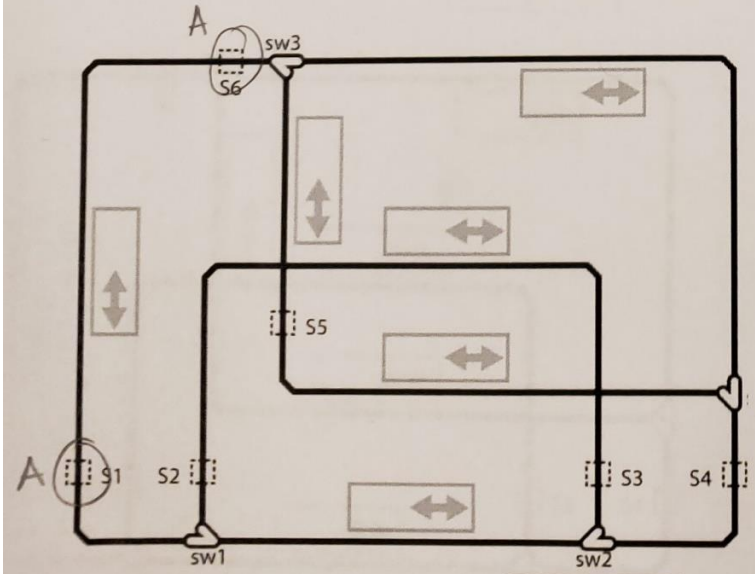
State: A5 B1

Input	Next State
S5	Astop
S1	Bstop
S5 & S1	A6 B4



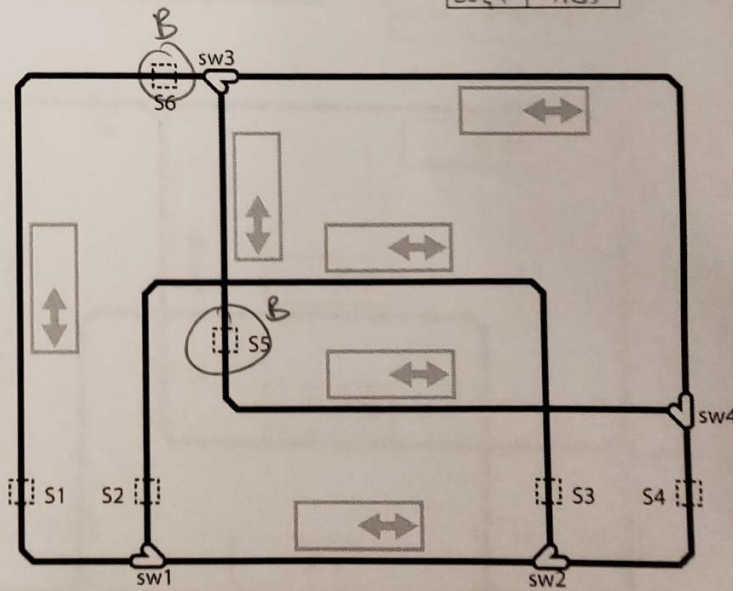
State: A6 B4

Input	Next State
S6	Astop
S4	Bstop
S6 & S4	A1B5



State: AStop

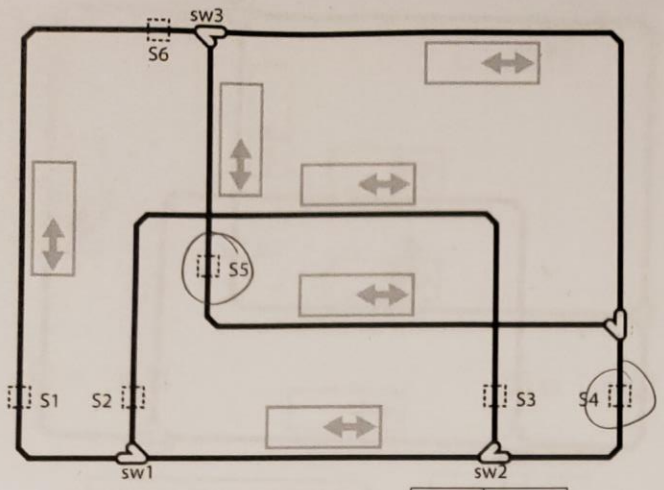
Input	Next State
S5	A4B6
S4	A1B5
4 & 5	Not possible Astop



State: BStop

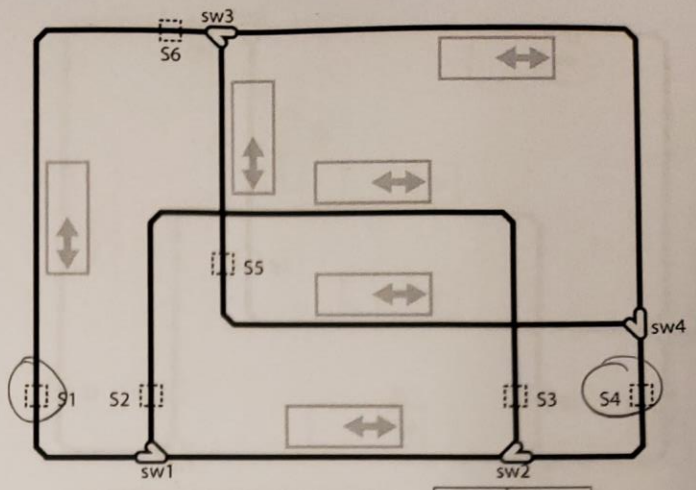
Input	Next State
S1	A4B6
S4	A5B1
1 & 4	Not possible B Stop

○ = stopped loc.



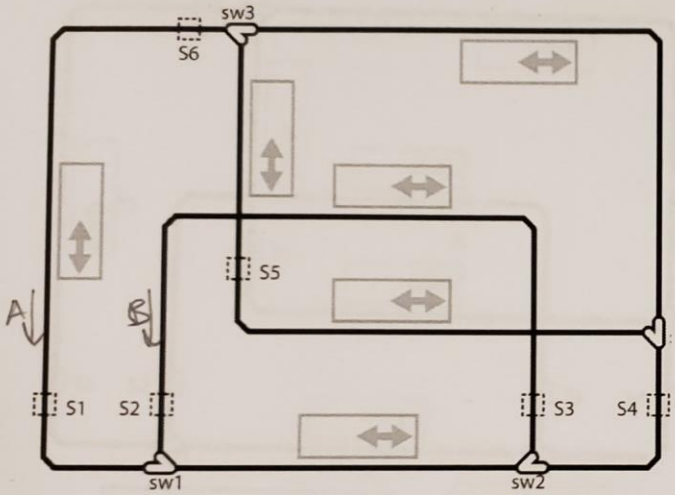
State: AStop2

Input	Next State
S6	AStop1
S1	AGo4
S1 & S6	N/A: AStop2



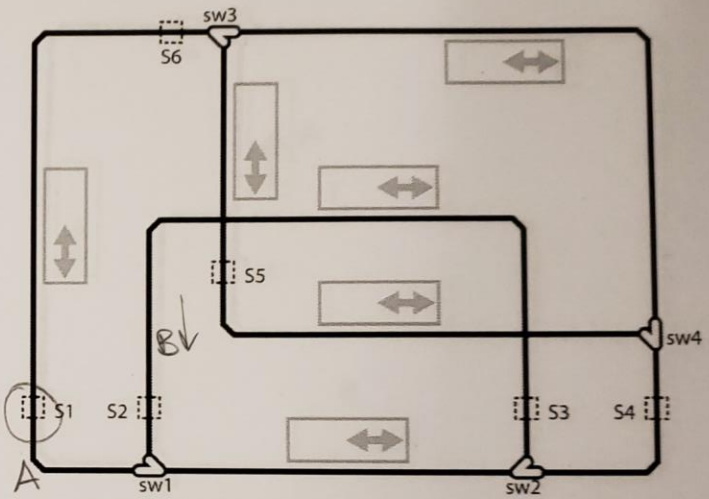
State: BStop2

Input	Next State
S5	AGo4
S6	AStop5
S5 & S6	BStop2 (N/A)



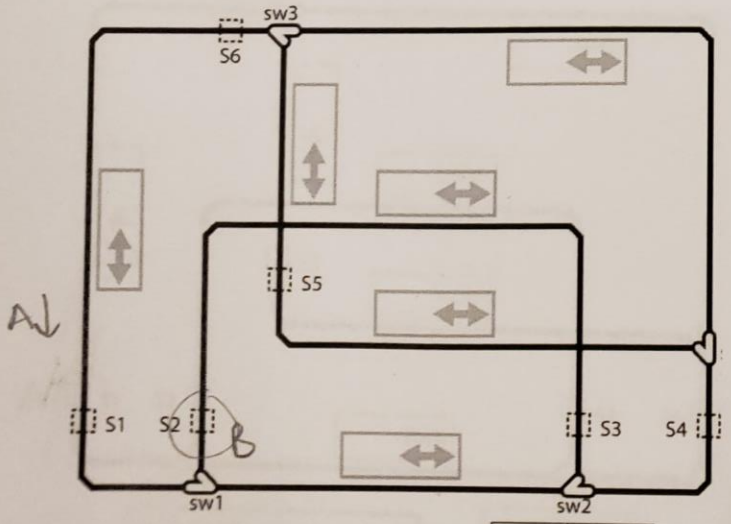
State: Both Move

Input	Next State
S1	BothMove1
S2	BothMove2
S1 & S2	BothMove2



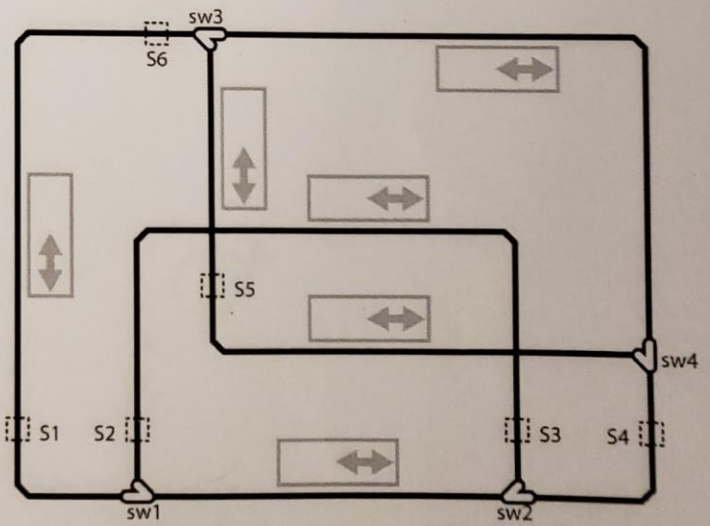
State: Both Move1

Input	Next State
S2	BStopReset2
S6	AStopReset2
S2 & S6	AStopReset2



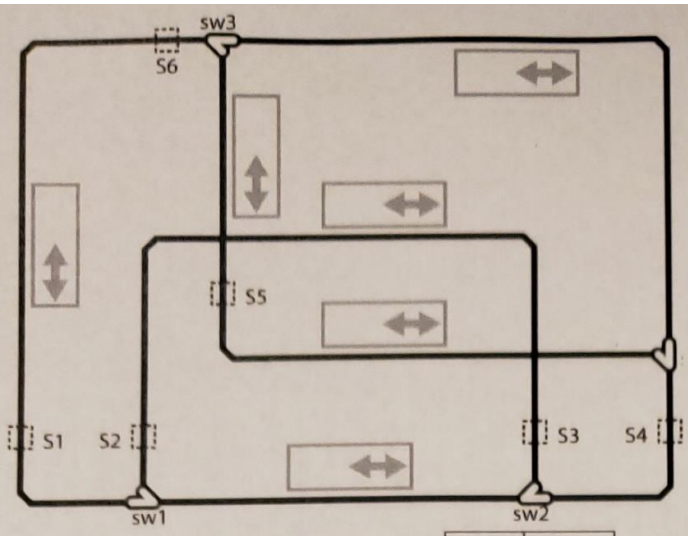
State: Both move 2

Input	Next State
S1	AStopReset
S6	BStopReset
S1 & S6	BStopReset



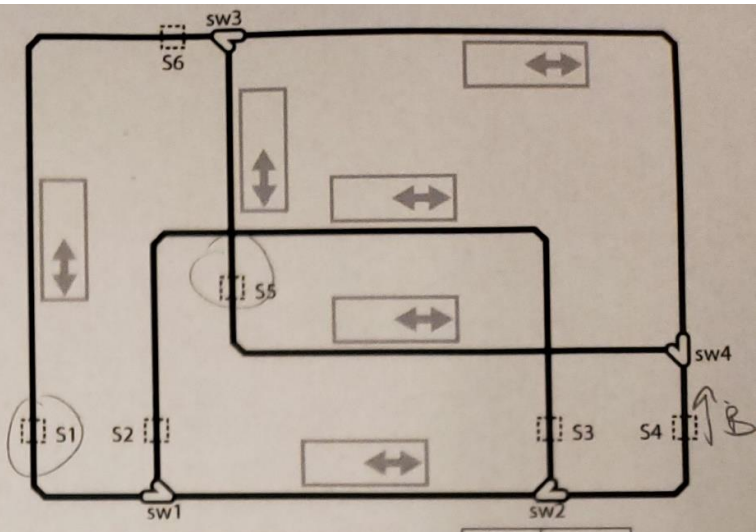
State:

Input	Next State



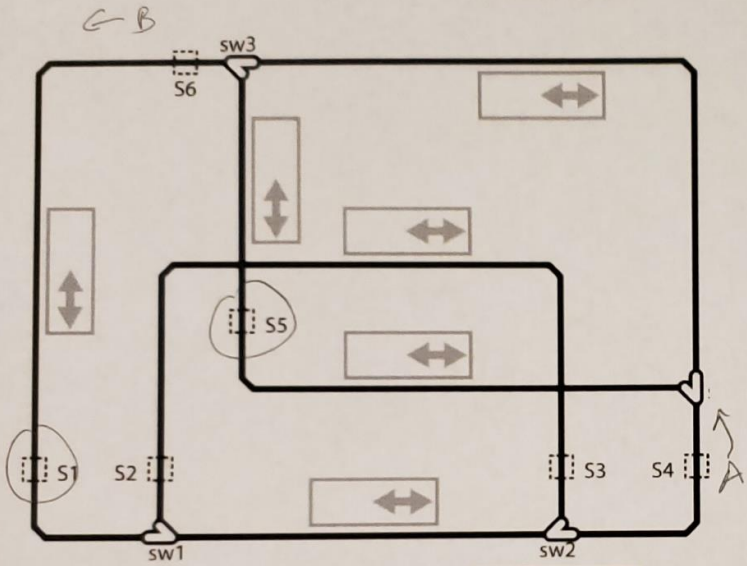
State: Astop Reset

Input	Next State
S6	Bstop Reset
	Astop Reset



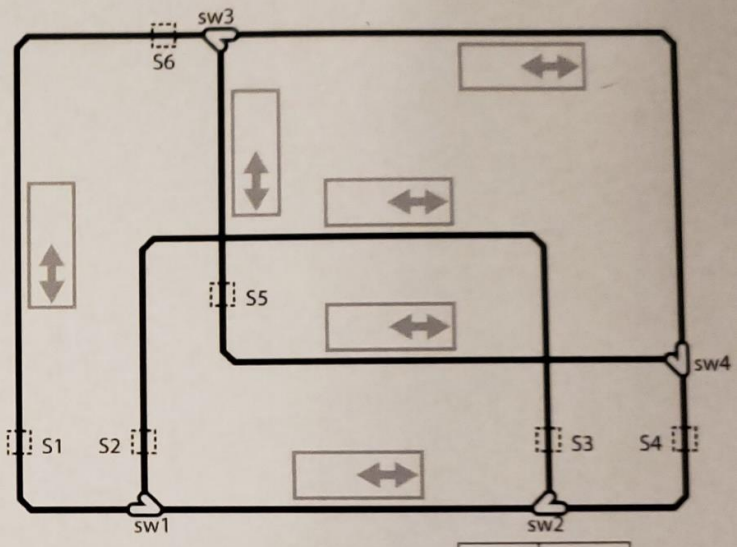
State: Astop Reset2

Input	Next State
S1&S5	A1B5



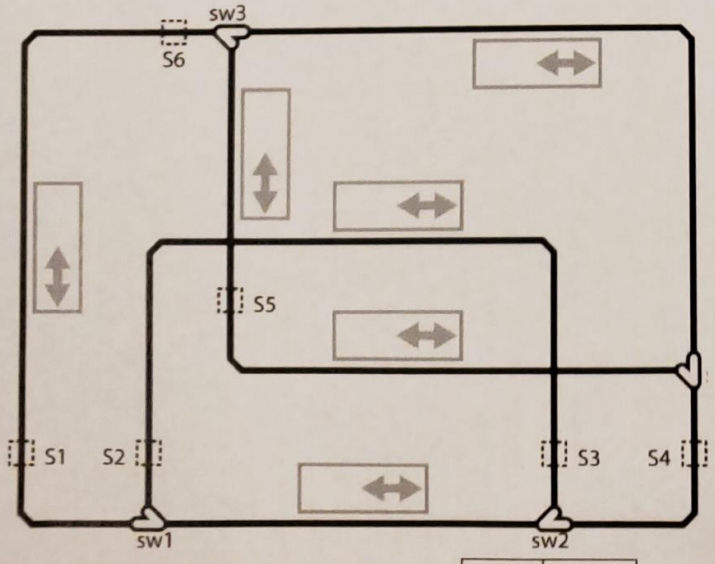
State: Bstop Reset

Input	Next State
S1&S5	A5B1



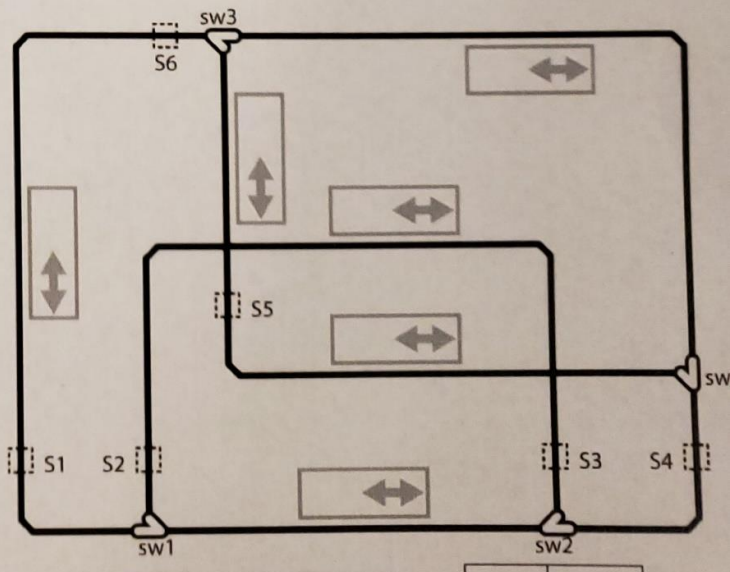
State: Bstop Reset2

Input	Next State
S6	Astop Reset2
	Bstop Reset2



State:

Input	Next State



State:

Input	Next State

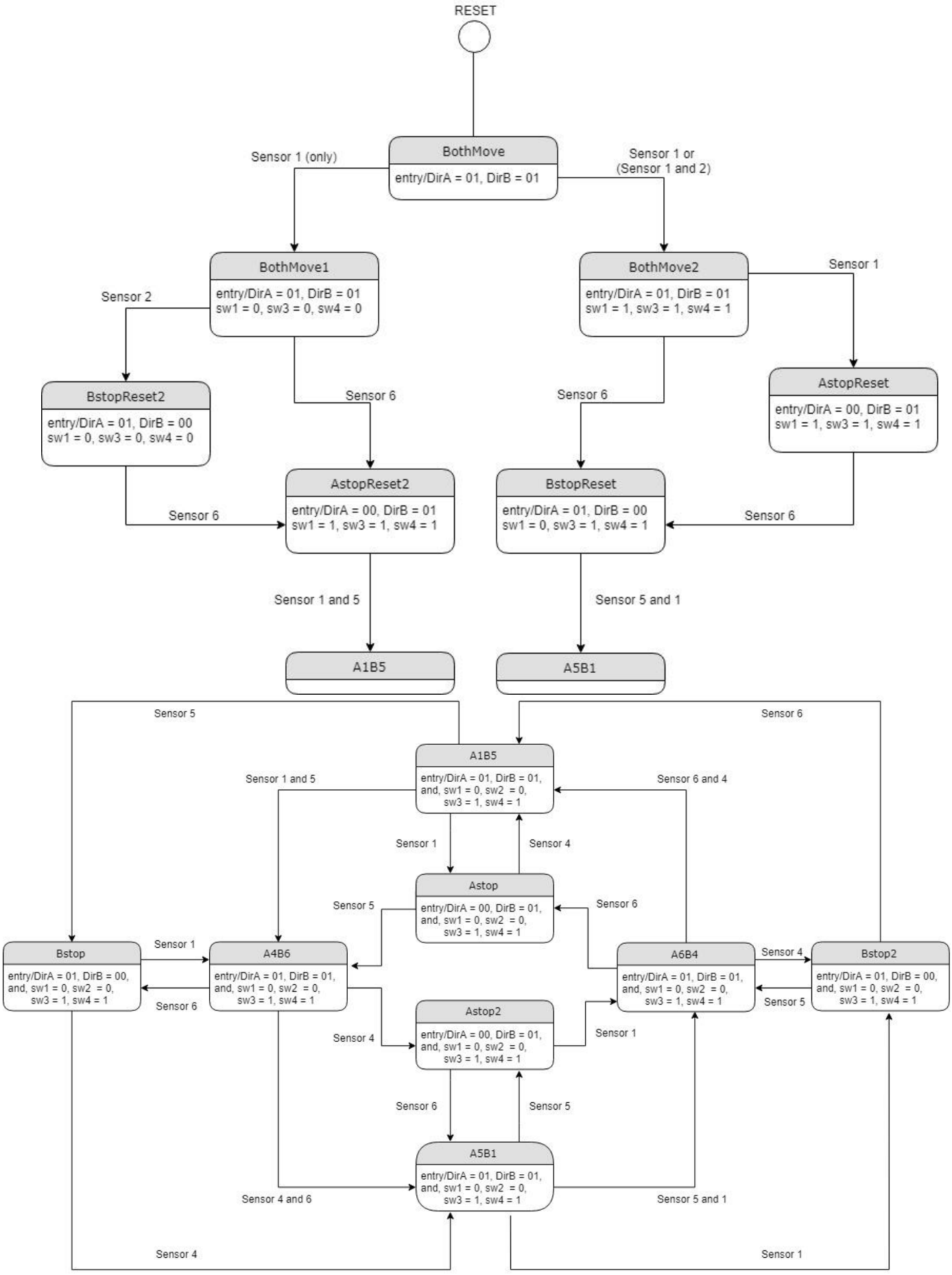


Figure 1. UML state chart for train state machine lab. Conditions to enter steady state are detailed above, and steady state implementation is below (charts separated for clarity). Note that states like AstopReset2 and AstopReset can be simplified to one state; however, both states included in practice for less ambiguity.

APPENDIX A
LAB 6 TRAIN STATE MACHINE IN VHDL

```
-- tcontrol.vhd Source File
-- Lab 6 Train State Machine VHDL Code Source File
-- State machine to control trains
-- Matthew Liu
-- ECE2031 L09
-- 21 February 2019

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


ENTITY Tcontrol 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 Tcontrol;


ARCHITECTURE a OF Tcontrol IS
    -- We create a new TYPE called STATE_TYPE that is only allowed
    -- to have the values specified here. This
    -- 1) allows us to use helpful names for values instead of
    --    arbitrary values
    -- 2) ensures that we can never accidentally set a signal of
    --    this type to an invalid value, and
    -- 3) helps the synthesis software create efficient hardware for our
design.
    TYPE STATE_TYPE IS (
        A1B5,
        A4B6,
        A5B1,
        A6B4,
        Bstop,
        Astop,
        Bstop2,
        Astop2,
        AstopReset,
        BstopReset,
        AstopReset2,
        BstopReset2,
        BothMove,
        BothMove1,
        BothMove2
    );
    -- Now we can create a signal of our new type.  Note that there is
    -- nothing special about the names "state" or "state_type", but it makes
    -- sense to use these names because that is how we are using them.
    SIGNAL state                                : STATE_TYPE;
    -- Here we create some new internal signals which will be concatenations
    -- of some of the sensor signals.  This will make CASE statements easier.
```

```
    SIGNAL sensor12,sensor24, sensor15, sensor46, sensor51,
sensor64,sensor62,sensor61     : std_logic_vector(1 DOWNTO 0);

BEGIN
    -- A process statement is required for clocked logic, such as a state
machine.
    PROCESS (clock, reset)
    BEGIN
        IF reset = '1' THEN
            -- Reset to this state
            state <= BothMove;
        ELSIF clock'EVENT AND clock = '1' THEN
            -- Case statement to determine next state.
            -- Case statements are a nice, clean way to make decisions
            -- based on different values of a signal.
            CASE state IS
            -- A1B5 State represents Train A and B approaching 1 and 5
respectively
            -- if one train reaches sensor before other enter into stop state
while other moves
            -- True for all states maintaining steady state A1B5, A4B6, etc.
                WHEN A1B5 =>
                    CASE sensor15 IS
                        WHEN "00" => state <= A1B5;
                        WHEN "01" => state <= Bstop;
                        WHEN "10" => state <= Astop;
                        WHEN "11" => state <= A4B6;
                        WHEN OTHERS => state <= A1B5;
                    END CASE;
            -- A4B6 State represents Train A and B approaching 4 and 6
respectively
                WHEN A4B6 =>
                        CASE sensor46 IS
                        WHEN "00" => state <= A4B6;
                        WHEN "01" => state <= Bstop;
                        WHEN "10" => state <= Astop2;
                        WHEN "11" => state <= A5B1;
                        WHEN OTHERS => state <= A4B6;
                    END CASE;
            -- A appraoch 4, B appraoch 1
                WHEN A5B1 =>
                    CASE sensor51 IS
                        WHEN "00" => state <= A5B1;
                        WHEN "01" => state <= Bstop2;
                        WHEN "10" => state <= Astop2;
                        WHEN "11" => state <= A6B4;
                        WHEN OTHERS => state <= A5B1;
                    END CASE;
            -- A approach 6, B approach 4
                WHEN A6B4 =>
                    CASE sensor64 IS
                        WHEN "00" => state <= A6B4;
                        WHEN "01" => state <= Bstop2;
                        WHEN "10" => state <= Astop;
                        WHEN "11" => state <= A1B5;
                        WHEN OTHERS => state <= A6B4;
                    END CASE;
```

```
-- Astops while B moves (steady state)
    WHEN Astop =>
        IF sensor4 = '1' THEN
            state <= A1B5;
        ELSIF sensor5 = '1' THEN
            state <= A4B6;
        ELSE
            state <= Astop;
        END IF;
-- Bstops while A moves (steady state)
    WHEN Bstop =>
        IF sensor1 = '1' THEN
            state <= A4B6;
        ELSIF sensor4 = '1' THEN
            state <= A5B1;
        ELSE
            state <= Bstop;
        END IF;
-- A stop while B moves (steady state)
    WHEN Astop2 =>
        IF sensor6 = '1' THEN
            state <= A5B1;
        ELSIF sensor1 = '1' THEN
            state <= A6B4;
        ELSE
            state <= Astop2;
        END IF;
-- B stops while A moves (Steady state)
    WHEN Bstop2 =>
        IF sensor5 = '1' THEN
            state <= A6B4;
        ELSIF sensor6 = '1' THEN
            state <= A1B5;
        ELSE
            state <= Bstop2;
        END IF;
-- A stops while B moves during Beginning/Reset Phase
    WHEN AstopReset =>
        IF sensor61 = "11" THEN
            state <= BstopReset;

        ELSIF sensor46 = "11" THEN
            state <= A1B5; -- Entrance into Steady State
        ELSE
            state <= AstopReset;
        END IF;
-- B stop A moves (reset phase)
    WHEN BstopReset =>
        IF sensor62= "11" THEN
            state <= AstopReset;

        ELSIF sensor46 = "11" THEN
            state <= A5B1; -- Entrance into Steady State

        ELSE
            state <= BstopReset;
        END IF;
```

```
            -- A stop B moves (reset phase) Additional state (2) to provide
clarity in track orientation
                WHEN AstopReset2 =>
                    IF sensor61 = "11" THEN
                        state <= BstopReset2;

                    ELSIF sensor46 = "11" THEN
                        state <= A1B5; -- Entrance into Steady State
                    ELSE
                        state <= AstopReset2;
                    END IF;
            -- A stop B moves (reset phase)
                WHEN BstopReset2 =>
                    IF sensor62= "11" THEN
                        state <= AstopReset2;

                    ELSIF sensor46 = "11" THEN
                        state <= A5B1; -- Entrance into Steady State
                    ELSE
                        state <= BstopReset2;
                    END IF;
            -- Both Trains are moving (reset phase)-  sensor 1 hit first, new
state used for clarity
                WHEN BothMove1 =>
                    IF sensor2 = '1' THEN
                        state <= BstopReset2;
                    ELSIF sensor6 = '1' THEN
                        state <= AstopReset2;
                    ELSE
                        state <= BothMove1;
                    END IF;
            -- Both Trains are moving (reset phase) sensor - sensor 2 hit
first
                WHEN BothMove2 =>
                    IF sensor6 = '1' THEN
                        state <= BstopReset;
                    ELSIF sensor1 = '1' THEN
                        state <= AstopReset;
                    ELSE
                        state <= BothMove2;
                    END IF;
            -- Both Trains are moving at Very Beginning of reset phase
                WHEN BothMove =>
                    IF sensor12 = "11" THEN
                        state <= BothMove2;
                    ELSIF sensor12 = "10" THEN
                        state <= BothMove1;
                    ELSIF sensor12 = "01" THEN
                        state <= BothMove2;
                    ELSIF sensor12 = "00" THEN
                        state <= BothMove;
                    END IF;

            END CASE;
        END IF;
    END PROCESS;
```

```
    -- Notice that all of 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.

    -- Combine bits for the internal signals declared above.
    -- ("&" operator concatenates bits)
    sensor12 <= sensor1 & sensor2;
    sensor24 <= sensor2 & sensor4;
    sensor15 <= sensor1 & sensor5;
    sensor46 <= sensor4 & sensor6;
    sensor51 <= sensor5 & sensor1;
    sensor64 <= sensor6 & sensor4;
    sensor62 <= sensor6 & sensor2;
    sensor61 <= sensor6 & sensor1;


    -- The following outputs depend on the state.
    WITH state SELECT switch1 <=
        '0' WHEN A1B5,
        '0' WHEN A4B6,
        '0' WHEN A5B1,
        '0' WHEN A6B4,
        '0' WHEN Astop,
        '0' WHEN Bstop,
        '1' WHEN AstopReset2,
        '0' WHEN BstopReset2,
        '1' WHEN AstopReset, --only train B is moving
        '0' WHEN BstopReset, --only train A is moving
        '0' WHEN BothMove1, -- Both Move (technically state can be combined
with other move statements)
        '1' WHEN BothMove2,
        '0' WHEN BothMove, --note doesn't matter what state trains have not
hit any switch or sensor
        '0' WHEN OTHERS;
    WITH state SELECT DirA <=
        "01" WHEN A1B5,
        "01" WHEN A4B6,
        "01" WHEN A5B1,
        "01" WHEN A6B4,
        "01" WHEN Bstop,
        "00" WHEN Astop,
        "00" WHEN AstopReset,
        "01" WHEN BstopReset,
        "00" WHEN AstopReset2,
        "01" WHEN BstopReset2,
        "01" WHEN BothMove1,
        "01" WHEN BothMove2,
        "01" WHEN BothMove,
        "00" WHEN Astop2,
        "01" WHEN Bstop2,
        "01" WHEN OTHERS; -- doesn't matter what state

    WITH state SELECT DirB <=
        "01" WHEN A1B5,
        "01" WHEN A4B6,
        "01" WHEN A5B1,
```

```
        "01" WHEN A6B4,
        "01" WHEN Astop,
        "00" WHEN Bstop,
        "00" WHEN BstopReset,
        "01" WHEN AstopReset,
        "00" WHEN BstopReset2,
        "01" WHEN AstopReset2,
        "01" WHEN BothMove1,
        "01" WHEN BothMove2,
        "01" WHEN BothMove,
        "01" WHEN Astop2,
        "00" WHEN Bstop2,
        "01" WHEN OTHERS; -- doesnt matter what state
    WITH state SELECT switch3 <=
        '1' WHEN A1B5,
        '1' WHEN A4B6,
        '1' WHEN A5B1,
        '1' WHEN A6B4,
        '1' WHEN Astop,
        '1' WHEN Bstop,
        '1' WHEN AstopReset,
        '1' WHEN BstopReset,
        '1' WHEN AstopReset2,
        '0' WHEN BstopReset2,
        '0' WHEN BothMove1,
        '1' WHEN BothMove2,
        '0' WHEN BothMove,
        '1' WHEN OTHERS; --doesnt matter what state
    WITH state SELECT switch4 <=
        '1' WHEN A1B5,
        '1' WHEN A4B6,
        '1' WHEN A5B1,
        '1' WHEN A6B4,
        '1' WHEN Astop,
        '1' WHEN Bstop,
        '1' WHEN AstopReset,
        '1' WHEN BstopReset,
        '1' WHEN AstopReset2,
        '0' WHEN BstopReset2,
        '0' WHEN BothMove1,
        '1' WHEN BothMove2,
        '0' WHEN BothMove,
        '1' WHEN OTHERS; -- doesnt matter what state

    -- These outputs happen to be constant values for this solution;
    -- they do not depend on the state.
    Switch2 <= '0';


END a;
```