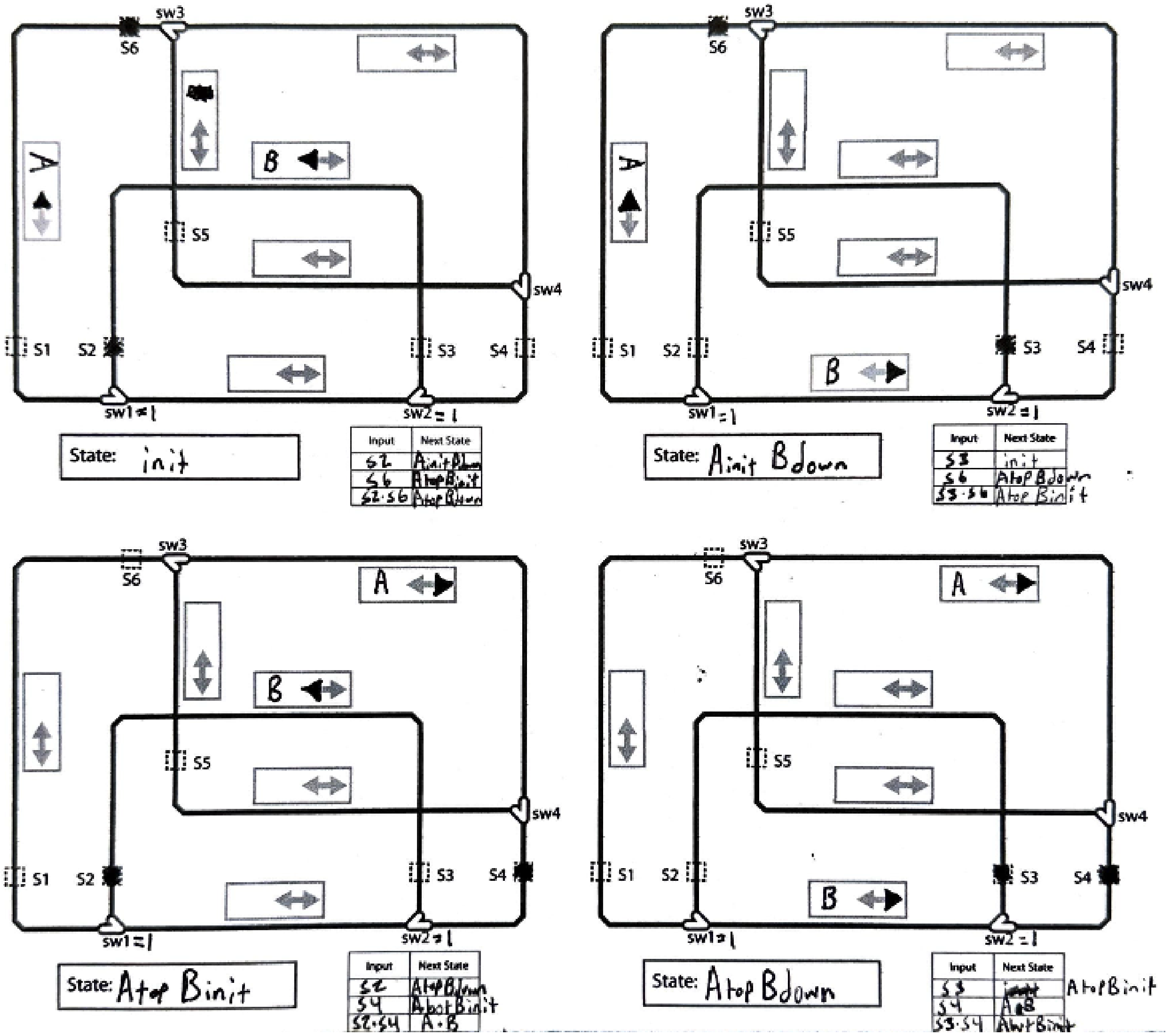
Andrew Friedman
Lab 6 Report
ECE 2031 CS
29 June 2023



State: init

Input	Next State
S2	Ainit Bdown
S6	Atop Binit
S2·S6	Atop Bdown

State: Ainit Bdown

Input	Next State
S3	init
S6	AtoP Bdown
S3·S6	Atop Binit

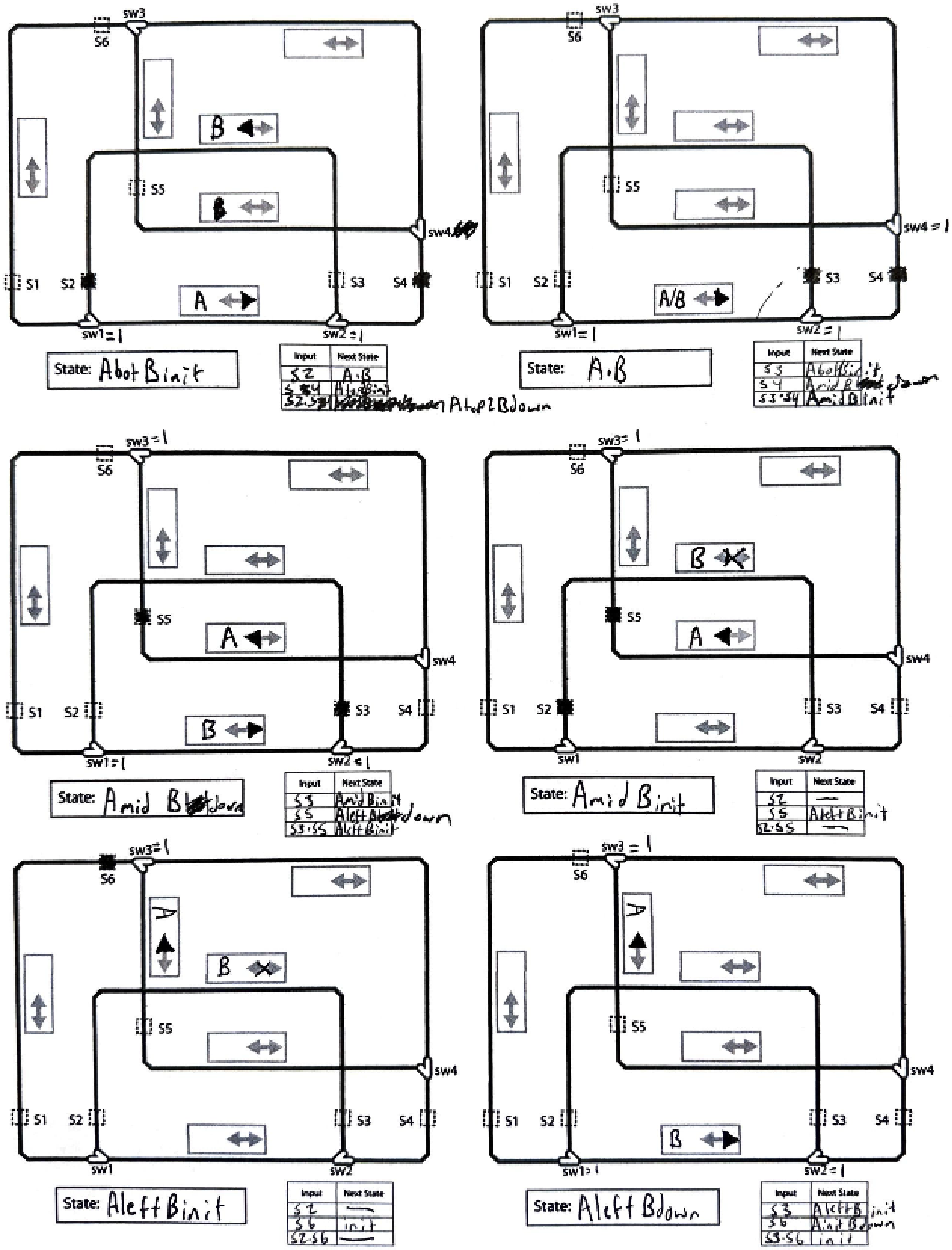
State: Atop Binit

Input	Next State
S2	AtoPBdown
S4	Abot Binit
S2·S4	A·B

State: Atop Bdown

Input	Next State
S3	~~init~~ AtoPBinit
S4	A·B
S3·S4	AtoPBinit

Figure 1. Collection of first four initial state diagrams for the train control system.



State: Abot Binit

Input	Next State
S2	A·B
S4	Abot Binit
S2·S4	Atop2Bdown

State: A·B

Input	Next State
S3	Abot Binit
S4	Amid B ~~init~~ down
S3·S4	Amid Binit

State: Amid Bdown

Input	Next State
S3	Amid Binit
S5	Aleft Bdown
S3·S5	Aleft Binit

State: Amid Binit

Input	Next State
S2	—
S5	Aleft Binit
S2·S5	—

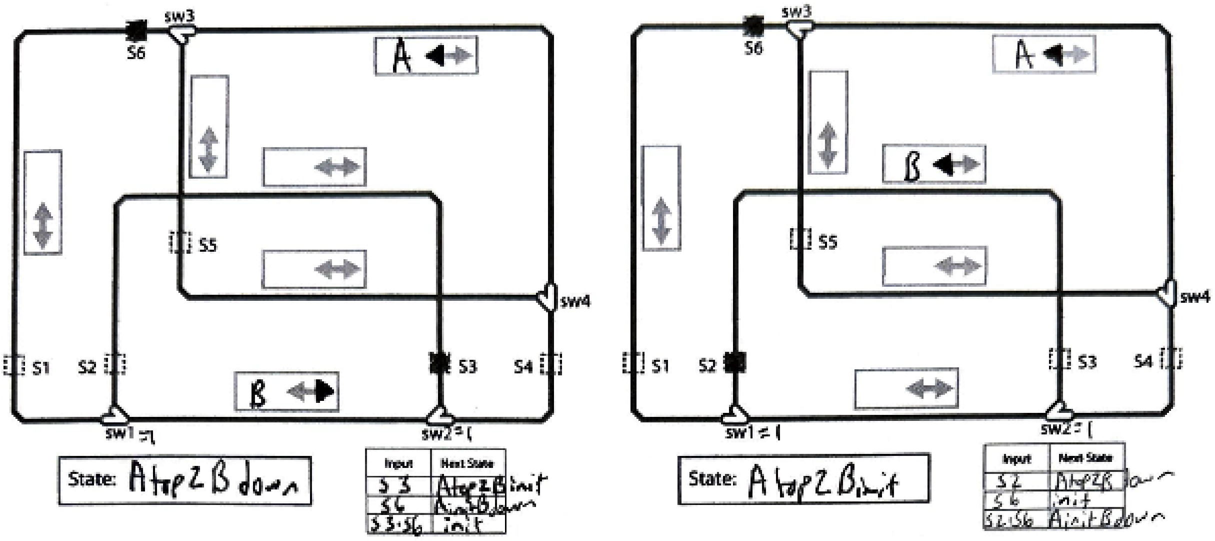
State: Aleft Binit

Input	Next State
S2	—
S6	init
S2·S6	—

State: Aleft Bdown

Input	Next State
S3	Aleft B init
S6	Ainit Bdown
S3·S6	init

Figure 2. Collection of next six initial state diagrams for the train control system.



State: Atop2Bdown

Input	Next State
S3	Atop2Binit
S6	AinitBdown
S3·S6	init

State: Atop2Binit

Input	Next State
S2	Atop2Bdown
S6	init
S2·S6	AinitBdown

Figure 3. Collection of last two initial state diagrams for the train control system.

Outputs	States							
	init	AinitBdown	AtopBdown	AtopBinit	AbotBinit	A·B	AmidBdown	AmidBinit
Sw1	1	1	1	1	1	1	1	0
Sw2	1	1	1	1	1	1	1	0
Sw3	0	0	0	0	0	0	1	1
Sw4	0	0	0	0	0	1	0	0
DA[1..0]	01	01	01	01	10	10	10	10
DB[1..0]	01	01	01	01	01	01	01	00

Outputs	States							
	AleftBinit	AleftBdown	Atop2Bdown	Atop2Binit				
Sw1	0	1	1	1				
Sw2	0	1	1	1				
Sw3	1	1	0	0				
Sw4	0	0	0	0				
DA[1..0]	10	10	10	10				
DB[1..0]	00	01	01	01				

Figure 4. Output table outlining state machine responses, segmented for readability.

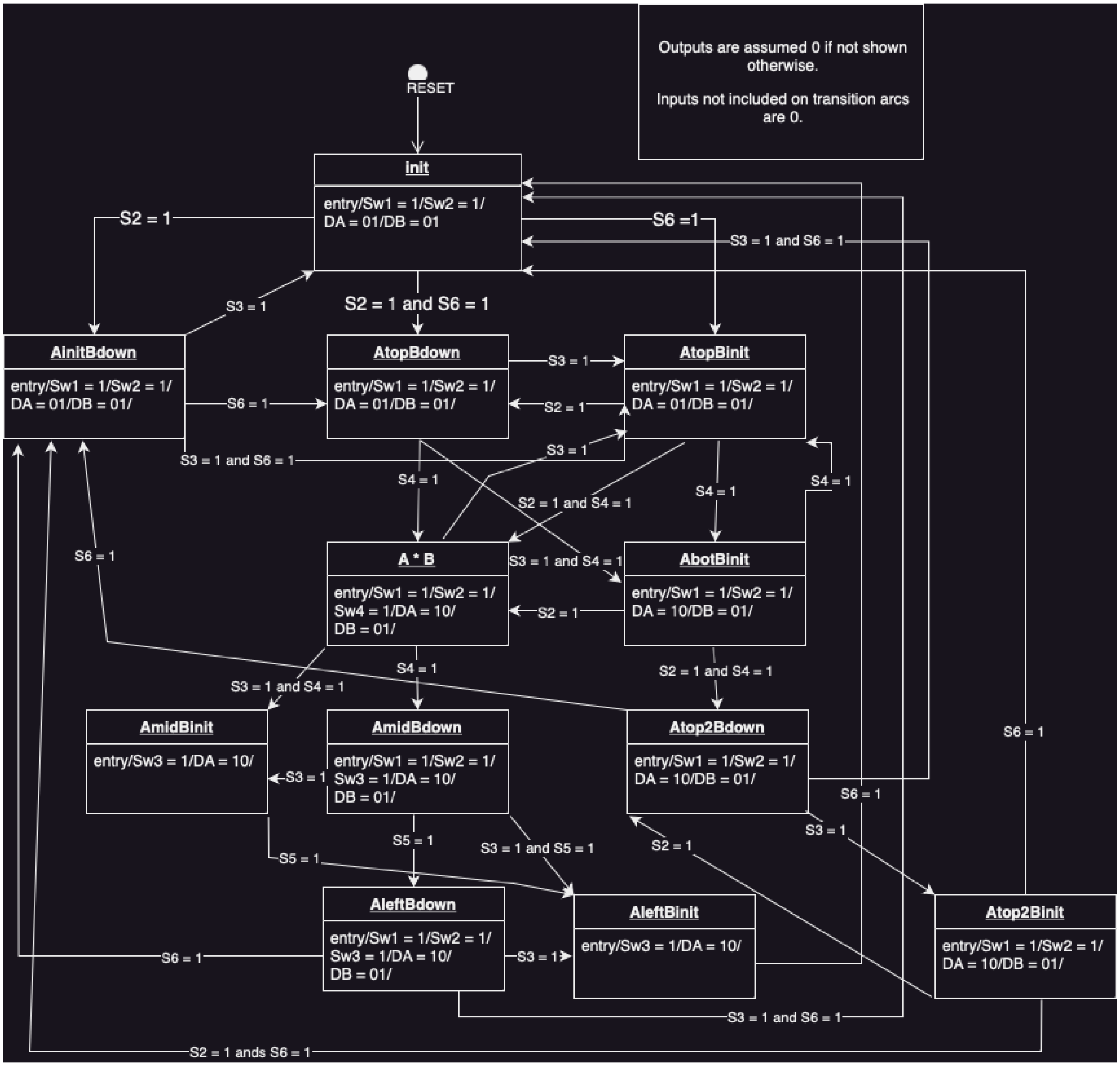

Outputs are assumed 0 if not shown otherwise.
Inputs not included on transition arcs are 0.
RESET
Init
entry/Sw1 = 1/Sw2 = 1/ DA = 01/DB = 01
S2 = 1
S6 =1
S3 = 1 and S6 = 1
S3 = 1
S2 = 1 and S6 = 1
AInitBdown
entry/Sw1 = 1/Sw2 = 1/ DA = 01/DB = 01/
AtopBdown
entry/Sw1 = 1/Sw2 = 1/ DA = 01/DB = 01/
AtopBInit
entry/Sw1 = 1/Sw2 = 1/ DA = 01/DB = 01/
S3 = 1
S6 = 1
S2 = 1
S3 = 1 and S6 = 1
S3 = 1
S4 = 1
S2 = 1 and S4 = 1
S4 = 1
S4 = 1
S6 = 1
A * B
entry/Sw1 = 1/Sw2 = 1/ Sw4 = 1/DA = 10/ DB = 01/
S3 = 1 and S4 = 1
AbotBInit
entry/Sw1 = 1/Sw2 = 1/ DA = 10/DB = 01/
S2 = 1
S4 = 1
S3 = 1 and S4 = 1
S2 = 1 and S4 = 1
S6 = 1
AmidBInit
entry/Sw3 = 1/DA = 10/
AmidBdown
entry/Sw1 = 1/Sw2 = 1/ Sw3 = 1/DA = 10/ DB = 01/
S3 = 1
Atop2Bdown
entry/Sw1 = 1/Sw2 = 1/ DA = 10/DB = 01/
S6 = 1
S5 = 1
S3 = 1 and S5 = 1
S2 = 1
S3 = 1
S5 = 1
AleftBdown
entry/Sw1 = 1/Sw2 = 1/ Sw3 = 1/DA = 10/ DB = 01/
AleftBInit
entry/Sw3 = 1/DA = 10/
Atop2BInit
entry/Sw1 = 1/Sw2 = 1/ DA = 10/DB = 01/
S6 = 1
S3 = 1
S3 = 1 and S6 = 1
S2 = 1 ands S6 = 1

Figure 5. Initial UML state diagram for the train control state machine, subject to changes and updates during testing and debugging in the lab.

APPENDIX A
VHDL STATE MACHINE CODE IMPLEMENTING TRAIN SYSTEM

```
--
--
-- State machine to control trains
-- This template implements the same general path as the example
-- covered in lecture, but does not "release" trains from their
-- stopped state until the other train is completely clear of the
-- relevant sensor.  This is to account for long trains, but that
-- does not affect your assignment this semester.
--

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 (
                init,
                AinitBdown,
                AtopBdown,
                AtopBinit,
                AbotBinit,
                AB,
                AmidBdown,
                AmidBinit,
                AleftBinit,
                AleftBdown,
        AtoptBdown,
        AtoptBinit
    );
    -- Create a signal of the new type.  Note that there is
    -- nothing special about the names "state" or "state_type", but it makes
    -- sense to use these names because those names fit their purpose.
    SIGNAL 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.
    -- Note that the names are *not* what makes them concatenations of the relevant
    -- signals; all of these signals need to be assigned values in the architecture.
    SIGNAL sensor12, sensor13, sensor24         : 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 -- This state machine uses an active-high reset.
            -- Reset to this state
            state <= init;
        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
                WHEN init =>
                    CASE Sensor26 IS
                        WHEN "00" => state <= init;
                        WHEN "01" => state <= AtopBinit;
                        WHEN "10" => state <= AinitBdown;
                        WHEN "11" => state <= AtopBdown;
                        WHEN OTHERS => state <= init;
                    END CASE;

                WHEN AinitBdown =>
                    CASE Sensor36 IS
                        WHEN "00" => state <= AinitBdown;
                        WHEN "01" => state <= AtopBdown;
                        WHEN "10" => state <= init;
                        WHEN "11" => state <= AtopBinit;
                        WHEN OTHERS => state <= AinitBdown;
                    END CASE;

                WHEN AtopBinit =>
    CASE Sensor24 IS
        WHEN "00" => state <= AtopBinit;
        WHEN "01" => state <= AbotBinit;
        WHEN "10" => state <= AtopBdown;
        WHEN "11" => state <= AB;
        WHEN OTHERS => state <= AtopBinit;
    END CASE;

WHEN AtopBdown =>
    CASE Sensor34 IS
        WHEN "00" => state <= AtopBdown;
        WHEN "01" => state <= AB;
        WHEN "10" => state <= AtopBinit;
        WHEN "11" => state <= AbotBinit;
        WHEN OTHERS => state <= AtopBdown;
    END CASE;

-- shmurda 1

WHEN AbotBinit =>
    CASE Sensor24 IS
        WHEN "00" => state <= AbotBinit;
        WHEN "01" => state <= AtopBinit;
        WHEN "10" => state <= AB;
        WHEN "11" => state <= AtoptBdown;
        WHEN OTHERS => state <= AbotBinit;
    END CASE;

WHEN AB =>
    CASE Sensor34 IS
        WHEN "00" => state <= AB;
        WHEN "01" => state <= AmidBdown;
        WHEN "10" => state <= AbotBinit;
        WHEN "11" => state <= AmidBinit;
        WHEN OTHERS => state <= AB;
    END CASE;

WHEN AmidBdown =>
    CASE Sensor35 IS
        WHEN "00" => state <= AmidBdown;
        WHEN "01" => state <= AleftBdown;
        WHEN "10" => state <= AmidBinit;
        WHEN "11" => state <= AleftBinit;
        WHEN OTHERS => state <= AmidBdown;
        END CASE;

WHEN AmidBinit =>
        CASE Sensor25 IS
                WHEN "00" => state <= AmidBinit;
                WHEN "01" => state <= AleftBinit;
                WHEN "10" => state <= AmidBinit;
                WHEN "11" => state <= AmidBinit;
                WHEN OTHERS => state <= AmidBinit;
        END CASE;

WHEN AleftBinit =>
        CASE Sensor6 IS
                WHEN "00" => state <= AleftBinit;
                WHEN "01" => state <= init;
                WHEN "10" => state <= AleftBinit;
                WHEN "11" => state <= AleftBinit;
                WHEN OTHERS => state <= AleftBinit;
        END CASE;

WHEN AleftBdown =>
        CASE Sensor6 IS
                WHEN "00" => state <= AleftBdown;
                WHEN "01" => state <= AinitBdown;
                WHEN "10" => state <= AleftBinit;
                WHEN "11" => state <= init;
                WHEN OTHERS => state <= AleftBdown;
        END CASE;

-- shmurda 2

WHEN AtoptBdown =>
        CASE Sensor36 IS
                WHEN "00" => state <= AtoptBdown;
                WHEN "01" => state <= AinitBdown;
                WHEN "10" => state <= AtoptBdown;
                WHEN "11" => state <= init;
                WHEN OTHERS => state <= AtoptBdown;
        END CASE;

WHEN AtoptBinit =>
        CASE Sensor26 IS
                WHEN "00" => state <= AtoptBinit;
                WHEN "01" => state <= init;
```

```
                                        WHEN "10" => state <= AtoptBdown;
                                        WHEN "11" => state <= AinitBdown;
                                        WHEN OTHERS => state <= AtoptBinit;
                                END CASE;
                        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 is concatenation)
sensor24 <= sensor2 & sensor4;
sensor34 <= sensor3 & sensor4;
sensor25 <= sensor2 & sensor5;
sensor35 <= sensor3 & sensor5;
sensor6 <= sensor6;
sensor26 <= sensor2 & sensor6;
sensor36 <= sensor3 & sensor6;

-- The following outputs depend on the state.  This is a Moore state machine,
-- so they ONLY depend on the state.
WITH state SELECT Switch1 <=
        '1' WHEN init,
        '1' WHEN AinitBdown,
        '1' WHEN AtopBdown,
        '1' WHEN AtopBinit,
        '1' WHEN AbotBinit,
        '1' WHEN AB,
        '1' WHEN AmidBdown,
        '0' WHEN AmidBinit,
        '0' WHEN AleftBinit,
        '1' WHEN AleftBdown,
        '1' WHEN AtoptBdown,
        '1' WHEN AtoptBinit;
WITH state SELECT Switch2 <=
        '1' WHEN init,
        '1' WHEN AinitBdown,
        '1' WHEN AtopBdown,
        '1' WHEN AtopBinit,
        '1' WHEN AbotBinit,
        '1' WHEN AB,
        '1' WHEN AmidBdown,
        '0' WHEN AmidBinit,
        '0' WHEN AleftBinit,
        '1' WHEN AleftBdown,
        '1' WHEN AtoptBdown,
        '1' WHEN AtoptBinit;
WITH state SELECT Switch3 <=
        '0' WHEN init,
        '0' WHEN AinitBdown,
        '0' WHEN AtopBdown,
        '0' WHEN AtopBinit,
        '0' WHEN AbotBinit,
        '0' WHEN AB,
        '1' WHEN AmidBdown,
        '1' WHEN AmidBinit,
        '1' WHEN AleftBinit,
        '1' WHEN AleftBdown,
        '0' WHEN AtoptBdown,
        '0' WHEN AtoptBinit;
WITH state SELECT Switch4 <=
        '0' WHEN init,
        '0' WHEN AinitBdown,
        '0' WHEN AtopBdown,
        '0' WHEN AtopBinit,
        '0' WHEN AbotBinit,
        '1' WHEN AB,
        '0' WHEN AmidBdown,
        '0' WHEN AmidBinit,
        '0' WHEN AleftBinit,
        '0' WHEN AleftBdown,
        '0' WHEN AtoptBdown,
        '0' WHEN AtoptBinit;
WITH state SELECT DirA <=
        '01' WHEN init,
        '01' WHEN AinitBdown,
        '01' WHEN AtopBdown,
        '01' WHEN AtopBinit,
        '10' WHEN AbotBinit,
        '10' WHEN AB,
        '10' WHEN AmidBdown,
        '10' WHEN AmidBinit,
        '10' WHEN AleftBinit,
        '10' WHEN AleftBdown,
        '10' WHEN AtoptBdown,
                    '10' WHEN AtoptBinit;
          WITH state SELECT DirB <=
                    '01' WHEN init,
                    '01' WHEN AinitBdown,
                    '01' WHEN AtopBdown,
                    '01' WHEN AtopBinit,
                    '01' WHEN AbotBinit,
                    '01' WHEN AB,
                    '01' WHEN AmidBdown,
                    '00' WHEN AmidBinit,
                    '00' WHEN AleftBinit,
                    '01' WHEN AleftBdown,
                    '01' WHEN AtoptBdown,
                    '01' WHEN AtoptBinit;
END a;
```