

Evaluation of Transit Signal Priority Strategies

Washington Street – Bowdoin Street – Harvard Street Intersection

Duy Nguyen

CIVE 7382 – Advanced Signal Control

Instructor: Professor Peter Furth

December 19, 2025

Contents

1	Introduction	4
2	Base Signal Timing Plan	4
2.1	Phasing and Ring-Barrier Structure	5
2.2	Minimum and Maximum Green Times	5
2.3	Detection and Call Logic	6
2.4	Pedestrian Timing and Leading Pedestrian Intervals	6
2.5	Controller Implementation in VISSIM	6
3	Alternative 1: Transit Signal Priority via Green Extension	7
3.1	Description of TSP Strategy	7
3.2	Detector Placement and Logic	7
3.3	Dummy Signal Indicators for TSP Activation	8
3.4	Implementation in VISSIM	9
4	Alternative 2: Green Extension with Phase Rotation	9
4.1	Description of Phase Rotation Strategy	9
4.2	Conditions for Phase Rotation	9
4.3	Dummy Signal Indicators for Phase Rotation	10
4.4	Implementation in VISSIM	10
5	Traffic Demand and Bus Arrival Modeling	10
5.1	Vehicular Traffic Demand	10
5.2	Peak-Hour Demand	11
5.3	Non-Peak Demand	11
5.4	Bus Arrival Modeling	11
6	Simulation Results	12
6.1	Non-Peak Results	12
6.2	Peak Results	12
7	Discussion	13
7.1	Effectiveness of Green Extension	13
7.2	Effectiveness of GE+Phase Rotation	13
7.3	Peak vs. Non-Peak Performance	13
7.4	Impacts on General Traffic	14

8 Conclusion	14
A Video for GE and GE + Phase Rotation link	15
B VAP Signal Controller Code	15
B.1 Base Signal Timing Plan	15
B.2 Alternative 1: Green Extension	22
B.3 Alternative 2: Green Extension with Phase Rotation	32

1 Introduction

This report evaluates transit signal priority (TSP) strategies at the Washington Street – Bowdoin Street – Harvard Street intersection using microsimulation in PTV VISSIM. Two TSP alternatives are implemented and compared against a base signal timing plan provided by the City.

The objectives of this study are to:

- Describe how each TSP alternative is programmed,
- Verify correct TSP activation using dummy signal groups,
- Evaluate bus and general vehicle performance relative to the base plan.

2 Base Signal Timing Plan

The base signal timing plan for the Washington Street–Bowdoin Street–Harvard Street intersection was developed to replicate the existing field timing plan provided by the City of Boston. The intersection operates under an actuated, eight-phase NEMA dual-ring controller with pedestrian timing and leading pedestrian intervals (LPI) on selected approaches.

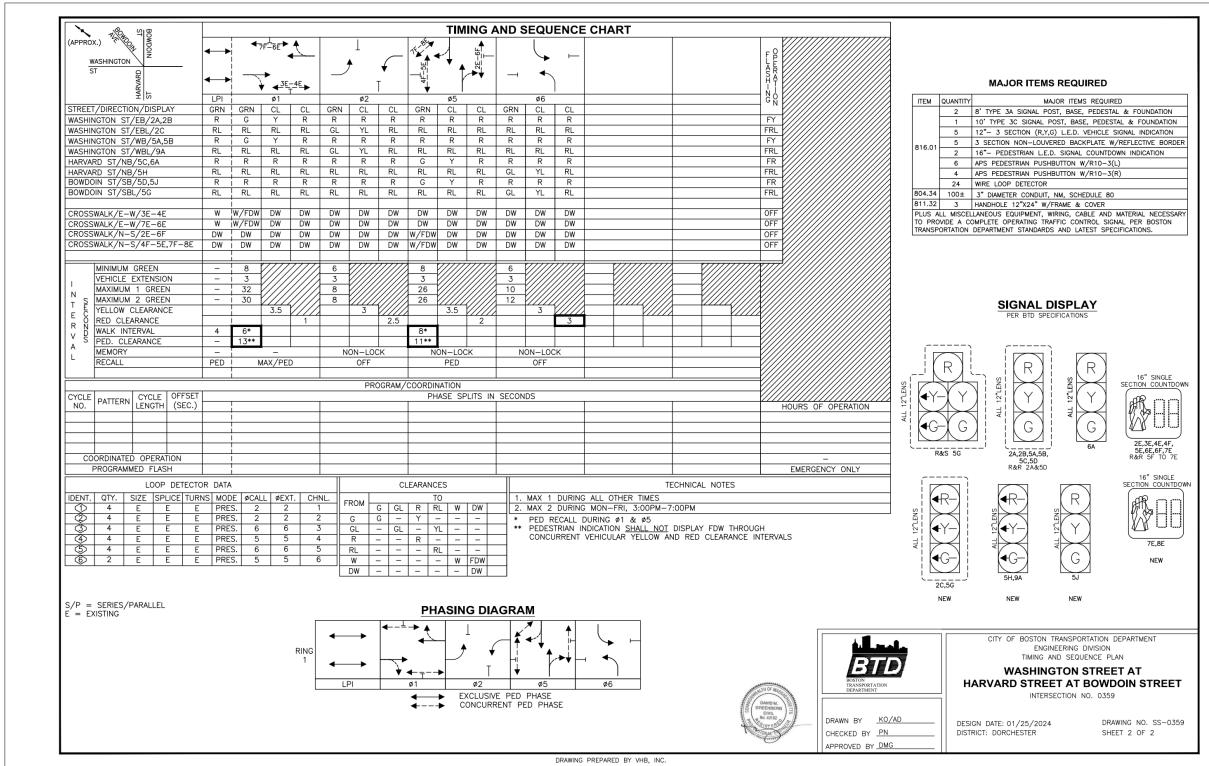


Figure 1: City of Boston Timing and Sequence Chart for the Washington Street–Bowdoin Street–Harvard Street intersection

Figure 1 presents the official timing and sequence chart used as the reference for this study.

2.1 Phasing and Ring-Barrier Structure

The controller follows a standard dual-ring configuration with Phases 1–4 operating in Ring 1 and Phases 5–8 operating in Ring 2. Barrier separation occurs between Phases 2 and 6, and again between Phases 4 and 8. Within each ring, phases are served sequentially unless skipped due to the absence of demand.

Phases 4 and 8 represent the major street through movements and are programmed with recall to ensure service even under low-demand conditions. All other phases operate under vehicle actuation and may be skipped if no call is present.

2.2 Minimum and Maximum Green Times

Minimum and maximum green times were implemented directly from the city timing plan. Table 1 summarizes the maximum green settings used in the VISSIM model.

Table 1: Base Plan Maximum Green Times

Phase	Maximum Green (s)
1	12
2	26
3	8
4	32
5	10
6	26
7	8
8	30

Minimum green times were defined within the VISSIM signal group settings to match the values shown in the timing plan. Vehicle gap-out was enabled for all phases except Phases 4 and 8, which were configured with effectively disabled gap-out to reflect their operation as major through movements.

2.3 Detection and Call Logic

Vehicle detection was implemented using presence detectors placed on each approach. Phases 4 and 8 were programmed with locking recall, ensuring that once a call is registered, the phase will be served. All other phases operate under non-locking actuation.

A detector call delay of two seconds was applied to reduce false or short-duration calls. This delay was implemented in the VAP logic using an occupancy-based threshold.

2.4 Pedestrian Timing and Leading Pedestrian Intervals

Pedestrian phases were implemented consistent with the timing plan. Leading pedestrian intervals were provided for the appropriate crosswalks to improve pedestrian safety by allowing pedestrians to establish presence in the crosswalk before the concurrent vehicular movement receives a green indication.

Pedestrian recall was enabled for selected phases (4 and 8) as specified in the timing plan, while pedestrian service for other movements operated on demand.

2.5 Controller Implementation in VISSIM

The base plan was implemented using a VISSIM Vehicle Actuated Programming (VAP) controller operating at a frequency of 10 Hz. Phase transitions were governed by conflict

status checks, minimum green enforcement, gap-out conditions, and maximum green constraints. Intergreen times were explicitly modeled to ensure realistic yellow and red clearance intervals.

No transit signal priority strategies were applied in the base plan. This configuration serves as the benchmark against which all TSP alternatives are evaluated in subsequent sections.

3 Alternative 1: Transit Signal Priority via Green Extension

3.1 Description of TSP Strategy

The first transit signal priority (TSP) alternative applies conditional green extension to the northbound and southbound through movements (Phases 8 and 4). When a bus is detected approaching the intersection during the final portion of a green phase, the controller allows the green to extend beyond its nominal maximum. The maximum extension is capped to limit impacts on conflicting movements.

Specifically, if a bus is detected within the final 10 seconds before the scheduled maximum green, the controller permits an extension of up to 10 additional seconds. If the bus clears the intersection before the extension limit is reached, the phase terminates immediately.

3.2 Detector Placement and Logic

Two detectors are used for each prioritized approach: a check-in detector and a check-out detector. The check-in detectors are placed approximately 450 feet upstream of the stop line. Given a bus free-flow speed of approximately 50 km/h (about 45 ft/s), this placement allows the bus to be detected roughly 10 seconds before arrival at the stop line, aligning with the extension eligibility window.

The check-out detectors are located immediately downstream of the stop line. Once a bus is detected at the check-out location, the green extension is terminated to prevent unnecessary additional green time.



Figure 2: Detector placement for transit signal priority. Upstream check-in detectors are placed approximately 450 ft upstream of the stop line to detect approaching buses, while downstream check-out detectors are located immediately after the stop line to terminate green extension once the bus clears the intersection.

3.3 Dummy Signal Indicators for TSP Activation

Dummy signal groups are used to visually verify TSP operation during simulation. One dummy signal indicates detection at the upstream check-in detector. A second dummy signal indicates when a bus is eligible for green extension, and a third dummy signal indicates when

green extension is actively being applied. A final dummy signal indicates detection at the downstream check-out detector.

These dummy signals are visible in both the simulation and the recorded video clips and are used to confirm correct implementation of the TSP logic.

3.4 Implementation in VISSIM

The green extension logic is implemented using the VISSIM VAP programming interface. Bus presence is tracked using a logical flag that is set when a bus is detected between the check-in and check-out detectors. Extension eligibility is evaluated based on the remaining green time relative to the programmed maximum green.

When extension conditions are satisfied, the controller increments the green time in small time steps until either the bus clears the intersection or the maximum allowable extension is reached. All other phase sequencing, recall, and conflict logic remain identical to the base plan to ensure a controlled comparison.

4 Alternative 2: Green Extension with Phase Rotation

4.1 Description of Phase Rotation Strategy

The second transit signal priority (TSP) alternative builds upon the green extension strategy described in Alternative 1 by adding conditional phase rotation at the barrier. When a bus is approaching the intersection during a cycle in which a leading left turn would normally be served, the controller temporarily converts the sequence to a lagging left turn.

Specifically, if a bus is active on the northbound or southbound approach, the through movement (Phases 8 or 4) is served first, followed by the corresponding left-turn movement (Phases 7 or 3). This allows the bus to receive priority without truncating the left-turn green.

4.2 Conditions for Phase Rotation

Phase rotation is evaluated only at the release of the main barrier between Rings 1 and 2. Rotation is applied when both of the following conditions are satisfied:

- A bus is active on the through movement (Phase 4 or Phase 8), defined as being detected between the upstream check-in detector and the downstream check-out detector.
- There is an active call for the corresponding left-turn phase (Phase 3 or Phase 7).

If these conditions are met, the controller serves the through phase first and ends the barrier on the left-turn phase. If the conditions are not met, the controller follows the normal leading-left sequence used in the base plan.

4.3 Dummy Signal Indicators for Phase Rotation

Dummy signal groups are used to verify correct phase rotation behavior. In addition to the dummy signals used for green extension, an additional dummy indicator is activated when phase rotation is selected at the barrier. This allows visual confirmation that the left-turn sequence has been shifted from leading to lagging in response to an approaching bus.

The dummy signals are visible in both the VISSIM simulation and the recorded video clips used for evaluation.

4.4 Implementation in VISSIM

The phase rotation logic is implemented in VISSIM using the VAP programming interface. Bus presence is latched using the same check-in and check-out detectors as Alternative 1, and no changes are made to detector placement or base signal timing parameters.

At each barrier release, rotation flags are set based on bus activity and left-turn demand. These flags modify the order of phase activation within each ring and determine which phases terminate the barrier. After the barrier is completed, the rotation flags are reset to ensure that subsequent cycles operate normally unless another bus request is detected.

5 Traffic Demand and Bus Arrival Modeling

5.1 Vehicular Traffic Demand

Turning-movement counts were collected over a 20-minute observation period and scaled to equivalent hourly volumes. Table 2 summarizes the observed 20-minute counts along with field conditions. Peak-hour demand volumes were obtained by scaling these counts to one hour. A non-peak demand scenario was created by applying a reduction factor of 0.6 to each peak-hour movement and rounding to the nearest vehicle, consistent with class guidance.

No spillback was observed for the turn lanes during data collection, and no conflicts between vehicular movements and pedestrians were present at the intersection.

Table 2: Turning Movement Counts and Field Conditions

Approach	Movement Volume (20 minutes)											
	NB			SB			EB			WB		
	L	T	R	L	T	R	L	T	R	L	T	R
Volume	19	71	52	27	91	10	9	97	11	40	61	11

Bus Headway: 8 minutes

Spillback: No for turn lanes

Pedestrian Conflicts: None observed

5.2 Peak-Hour Demand

The 20-minute turning-movement counts were scaled to represent peak-hour traffic demand. The resulting peak-hour volumes by approach and movement are shown in Table 3. These volumes were used as the base demand scenario for evaluating all signal control alternatives.

Table 3: Peak-Hour Vehicle Demand (1 Hour)

Approach	NB			SB			EB			WB		
	L	T	R	L	T	R	L	T	R	L	T	R
Peak Volume	57	213	156	81	273	30	27	291	33	120	183	33

5.3 Non-Peak Demand

To represent non-peak operating conditions, each peak-hour turning movement was multiplied by a factor of 0.6 and rounded to the nearest vehicle. The resulting non-peak hourly demand volumes are summarized in Table 4. This demand scenario was used to evaluate the robustness of Transit Signal Priority (TSP) strategies under lower traffic volumes.

Table 4: Non-Peak Vehicle Demand (1 Hour)

Approach	NB			SB			EB			WB		
	L	T	R	L	T	R	L	T	R	L	T	R
Non-Peak Volume	34	128	94	49	164	18	16	175	20	72	110	20

5.4 Bus Arrival Modeling

Bus arrivals on Washington Street were modeled stochastically to reflect realistic MBTA operations. Field observations indicate an average bus headway of approximately 8 minutes

at the study intersection. To avoid unrealistically regular bus arrivals, arrival times were generated externally using Python and then imported into VISSIM.

Bus headways were sampled from a normal distribution with a mean of 480 seconds (8 minutes) and a standard deviation of 120 seconds (2 minutes). Separate arrival time sequences were generated for northbound and southbound buses. Arrival times (in seconds from simulation start) are shown below:

- **Northbound bus arrivals:** 300, 678.5, 1057.7, 1472.3, 2021.6, 2399.2, 2933.7, 3491.4, 3893.5
- **Southbound bus arrivals:** 300, 885.0, 1268.8, 1763.7, 2227.5, 2676.2, 3234.4, 3811.7

This approach preserves the observed average bus frequency while introducing natural variability in arrival times. As a result, Transit Signal Priority strategies were evaluated under realistic and unsynchronized bus arrivals, ensuring that observed benefits and impacts are not artifacts of deterministic timing.

6 Simulation Results

6.1 Non-Peak Results

Table 5: Non-Peak Average Delay Results

Signal Control Strategy	Avg Delay (All Veh) (s/veh)	Avg Delay (Bus) (s/bus)
Base Plan	18.51	24.73
Green Extension (GE)	24.02	16.85
Green Extension + Phase Rotation (GE+PR)	16.47	8.09

6.2 Peak Results

Table 6: Peak Average Delay Results

Signal Control Strategy	Avg Delay (All Veh) (s/veh)	Avg Delay (Bus) (s/bus)
Base Plan	26.80	27.11
Green Extension (GE)	26.59	28.02
Green Extension + Phase Rotation (GE+PR)	23.48	11.13

7 Discussion

7.1 Effectiveness of Green Extension

The green extension strategy reduced bus delay under non-peak conditions but showed limited effectiveness during peak demand. In the non-peak scenario, bus delay decreased from 24.73 s under the base plan to 16.85 s with green extension, indicating that extending the through phase allowed buses to clear the intersection more efficiently when spare capacity was available.

However, during peak conditions, green extension alone resulted in a slight increase in average bus delay (from 27.11 s to 28.02 s). This outcome suggests that when the intersection operates near saturation, extending green time for one approach can disrupt downstream phase sequencing and increase the likelihood of buses arriving later in the cycle. As a result, green extension without additional coordination may inadvertently increase bus delay under heavy traffic conditions.

7.2 Effectiveness of GE+Phase Rotation

Adding phase rotation to the green extension strategy substantially improved bus performance in both demand scenarios. Under non-peak conditions, average bus delay was reduced further to 8.09 s, representing a 67% reduction relative to the base plan. During peak conditions, the combined strategy reduced bus delay from 27.11 s to 11.13 s, demonstrating a clear benefit even under congested traffic conditions.

Phase rotation improves effectiveness by modifying the phase sequence at the barrier, allowing the bus-serving through movement to be served before the opposing left turn when a bus is present. This reduces the likelihood that a bus must wait through an entire left-turn phase after arriving during red, which is a common source of delay in leading-left operations.

7.3 Peak vs. Non-Peak Performance

The results indicate that transit signal priority strategies are more effective during non-peak periods, when available green time can be reallocated without significantly impacting other movements. In peak conditions, green extension alone provides limited benefit due to high competing demand.

However, the addition of phase rotation maintained strong bus delay reductions even in peak conditions, suggesting that reordering phases can be more robust than simply extending green time. This highlights the importance of adapting signal logic to operational context rather than relying on a single TSP tactic.

7.4 Impacts on General Traffic

While green extension increased average delay for general traffic under non-peak conditions (from 18.51 s to 24.02 s), the combined green extension and phase rotation strategy reduced overall vehicle delay relative to the base plan in both scenarios. Under peak conditions, average vehicle delay decreased from 26.80 s to 23.48 s with the combined strategy.

These results indicate that properly designed transit priority strategies can improve bus performance without adversely affecting general traffic and may even enhance overall intersection efficiency by reducing inefficient phase utilization.

8 Conclusion

This study evaluated two transit signal priority (TSP) strategies at the Washington Street–Bowdoin Street–Harvard Street intersection using a VISSIM microsimulation model and a calibrated actuated signal controller. A base signal timing plan was first implemented and used as a benchmark, followed by two alternatives: green extension only, and green extension combined with phase rotation.

The results show that green extension alone can reduce bus delay under non-peak conditions but may be ineffective or even detrimental during peak periods. Under heavy traffic demand, extending green time for buses without adjusting phase sequencing can increase bus delay by pushing arrivals later into the signal cycle.

In contrast, the combined green extension and phase rotation strategy consistently reduced bus delay under both peak and non-peak conditions. By serving the through movement before the opposing left turn when a bus was present, phase rotation prevented buses from waiting through a full left-turn phase and significantly improved reliability. This approach reduced average bus delay by more than 50% during peak conditions compared to the base plan.

Importantly, the combined strategy did not degrade general traffic performance. Average vehicle delay decreased relative to the base plan in both demand scenarios, indicating that well-designed TSP strategies can improve transit performance without negatively impacting overall intersection efficiency.

Overall, this project demonstrates that effective transit signal priority requires both temporal and structural adjustments to signal control. While green extension addresses short-term arrival uncertainty, phase rotation provides a more robust solution under congested conditions. These findings highlight the importance of tailoring TSP strategies to traffic demand and signal phasing characteristics rather than relying on a single priority mechanism.

A Video for GE and GE + Phase Rotation link

<https://drive.google.com/drive/u/0/folders/18e37gUlzFMg766qSYgPTnvJkApopW3K6>

B VAP Signal Controller Code

B.1 Base Signal Timing Plan

Listing 1: VAP code for Base Plan

```
1 PROGRAM actuated_8PHASES;
2 /* Actuated control for 8 phases (NEMA dual-ring) */
3 /* Version 3a corrected 2025. */
4 /* Uses intergreen to create visible red clearance. */

5
6 VAP_Frequency 10;      /* execute logic 10 times per second */

7
8 ****
9 /* CONSTANTS */
10 CONST
11 CALLDELAY = 2;          /* seconds before detector registers call */

12 ****
13 /* ARRAYS */
14 ARRAY
15
16 MAXGREEN[8] = [12, 26, 8, 32, 10, 26, 8, 30],
17 MINGAP10[8] = [30, 30, 30, 30, 30, 30, 30, 30],
18 RECALL[8] = [0,0,0,1,0,0,0,1],    /* recall on throughs */
19 has_call[8],
20 no_conflict[8],
21 confl_call[8],
22 imminent[8],
23 early_green[8],
24 ext_green[8],
25 rest_green[8];

26 ****
27 /* SUBROUTINES */
28
29
30 /* ----- Locking Call ----- */
31 SUBROUTINE Locking_call;
32 /* sets a locking call with a delay of CALLDELAY seconds */
33 IF Current_state(i, red) THEN
```

```

34     IF (RECALL[i] OR (Occupancy(dk) > CALLDELAY)) THEN
35         has_call[i] := 1;
36     END ;
37 END .

38
39
40
41 SUBROUTINE NONLocking_call;
42     /* Sets a locking call with a delay of CALLDELAY seconds */
43     /* Implicit inputs: i (signal group), dk (call detector), RECALL []
44         */
45 IF Current_state(i, red) THEN                                /* Current state()
46     is a VAP function */
47     IF (RECALL[i] OR (Occupancy(dk)> CALLDELAY)) THEN      /* if RECALL is
48         true, dk's value won't matter */
49         has_call[i] := 1;
50     ELSE
51         has_call[i] := 0;
52     END
53 END .    /* The period ends the subroutine */

54 ****
55 /* ----- Imminent-to-Rest Transition ----- */
56 SUBROUTINE Imminent_to_rest;
57 /* transitions between states from imminent to rest_green */
58 IF (imminent[i] AND no_conflict[i]) THEN
59     Sg_green(i);
60     early_green[i] := 1;
61     imminent[i] := 0;
62     has_call[i] := 0;
63 END ;

64
65 IF (early_green[i] AND confl_call[i] AND (T_green(i) >= T_green_min(i)))
66     THEN
67     early_green[i] := 0;
68     ext_green[i] := 1;
69 END ;

70 IF (ext_green[i]) THEN
71     IF ((Headway10(dk) > MINGAP10[i]) OR (T_green(i) >= MAXGREEN[i])) THEN
72         ext_green[i] := 0;

```

```

73     rest_green[i] := 1;
74   END;
75 END.

76

77

78 SUBROUTINE Imminent_to_rest_LPI4;
79 /* transitions between states from imminent to rest_green */
80 IF (imminent[i] AND no_conflict[i]) THEN
81   Start(LPI4);
82   IF (LPI4=4) THEN
83     Sg_green(i);
84     early_green[i] := 1;
85     imminent[i] := 0;
86     has_call[i] := 0;
87     Stop(LPI4);
88     Reset(LPI4);
89   END;
90 END;

91

92 IF (early_green[i] AND confl_call[i] AND (T_green(i) >= T_green_min(i)))
93   THEN
94   early_green[i] := 0;
95   ext_green[i] := 1;
96 END;

97 IF (ext_green[i]) THEN
98   IF ((Headway10(dk) > MINGAP10[i]) OR (T_green(i) >= MAXGREEN[i])) THEN
99     ext_green[i] := 0;
100    rest_green[i] := 1;
101  END;
102 END.

103

104

105

106

107 SUBROUTINE Imminent_to_rest_LPI8;
108 /* transitions between states from imminent to rest_green */
109 IF (imminent[i] AND no_conflict[i]) THEN
110   Start(LPI8);
111   IF (LPI8=4) THEN
112     Sg_green(i);
113     early_green[i] := 1;
114     imminent[i] := 0;

```

```

115     has_call[i] := 0;
116     Stop(LPI8);
117     Reset(LPI8);
118   END;
119 END;
120
121 IF (early_green[i] AND confl_call[i] AND (T_green(i) >= T_green_min(i)))
122 THEN
123   early_green[i] := 0;
124   ext_green[i] := 1;
125 END;
126
127 IF (ext_green[i]) THEN
128   IF ((Headway10(dk) > MINGAP10[i]) OR (T_green(i) >= MAXGREEN[i])) THEN
129     ext_green[i] := 0;
130     rest_green[i] := 1;
131   END;
132 END.
133
134
135
136
137 /* **** */
138 /* ----- Initialize Early Green ----- */
139 SUBROUTINE Initialize_early_green;
140 IF Current_state(i, green) THEN
141   early_green[i] := 1;
142 END.
143
144 /* **** */
145 /* ===== MAIN PROGRAM ===== */
146
147 /* ---- Initialization ---- */
148 IF sim_timer = 0 THEN
149   START(sim_timer);
150 END;
151
152 IF sim_timer <= 5 THEN
153   i := 1; GOSUB Initialize_early_green;
154   i := 2; GOSUB Initialize_early_green;
155   i := 3; GOSUB Initialize_early_green;
156   i := 4; GOSUB Initialize_early_green;

```

```

157     i := 5; GOSUB Initialize_early_green;
158         i := 6; GOSUB Initialize_early_green;
159     i := 7; GOSUB Initialize_early_green;
160         i := 8; GOSUB Initialize_early_green;
161 END;
162
163
164 /****** */
165 /* ---- A. Update Calls ---- */
166 i := 1; dk := 11; GOSUB NONLocking_call;
167 i := 2; dk := 21; GOSUB NONLocking_call;
168 i := 3; dk := 31; GOSUB NONLocking_call;
169 i := 4; dk := 41; GOSUB Locking_call;
170 i := 5; dk := 51; GOSUB NONLocking_call;
171 i := 6; dk := 61; GOSUB NONLocking_call;
172 i := 7; dk := 71; GOSUB NONLocking_call;
173 i := 8; dk := 81; GOSUB Locking_call;
174
175
176 /****** */
177 /* ---- B. Conflict-Status Logic (realistic NEMA) ---- */
178 no_conflict[1] :=
179     Current_state(2, red)*Current_state(3, red)*
180     Current_state(4, red)*Current_state(7, red)*
181     Current_state(8, red)*(Remaining_Intergreen(1)=0);
182
183 no_conflict[2] :=
184     Current_state(1, red)*Current_state(3, red)*
185     Current_state(4, red)*Current_state(7, red)*
186     Current_state(8, red)*(Remaining_Intergreen(2)=0);
187
188 no_conflict[3] :=
189     Current_state(1, red)*Current_state(2, red)*
190     Current_state(4, red)*Current_state(5, red)*
191     Current_state(6, red)*(Remaining_Intergreen(3)=0);
192
193 no_conflict[4] :=
194     Current_state(1, red)*Current_state(2, red)*
195     Current_state(3, red)*Current_state(5, red)*
196     Current_state(6, red)*(Remaining_Intergreen(4)=0);
197
198 no_conflict[5] :=
199     Current_state(6, red)*Current_state(7, red)*

```

```

200    Current_state(8, red)*Current_state(3, red)*
201    Current_state(4, red)*(Remaining_Intergreen(5)=0);
202
203 no_conflict[6] :=
204    Current_state(5, red)*Current_state(7, red)*
205    Current_state(8, red)*Current_state(3, red)*
206    Current_state(4, red)*(Remaining_Intergreen(6)=0);
207
208 no_conflict[7] :=
209    Current_state(5, red)*Current_state(6, red)*
210    Current_state(8, red)*Current_state(1, red)*
211    Current_state(2, red)*(Remaining_Intergreen(7)=0);
212
213 no_conflict[8] :=
214    Current_state(5, red)*Current_state(6, red)*
215    Current_state(7, red)*Current_state(1, red)*
216    Current_state(2, red)*(Remaining_Intergreen(8)=0);
217
218 /*****C. Conflicting Calls *****/
219 /* ---- C. Conflicting Calls ---- */
220 confl_call[1] := has_call[2] OR has_call[3] OR has_call[4] OR has_call[7]
221     OR has_call[8];
222 confl_call[2] := has_call[1] OR has_call[3] OR has_call[4] OR has_call[7]
223     OR has_call[8];
224 confl_call[3] := has_call[1] OR has_call[2] OR has_call[4] OR has_call[5]
225     OR has_call[6];
226 confl_call[4] := has_call[1] OR has_call[2] OR has_call[3] OR has_call[5]
227     OR has_call[6];
228 confl_call[5] := has_call[3] OR has_call[4] OR has_call[6] OR has_call[7]
229     OR has_call[8];
230 confl_call[6] := has_call[3] OR has_call[4] OR has_call[5] OR has_call[7]
231     OR has_call[8];
232 confl_call[7] := has_call[1] OR has_call[2] OR has_call[5] OR has_call[6]
233     OR has_call[8];
234 confl_call[8] := has_call[1] OR has_call[2] OR has_call[5] OR has_call[6]
235     OR has_call[7];
236
237 /*****D. Phase Transition Evaluation *****/
238 /* ---- D. Phase Transition Evaluation ---- */
239 i := 1; dk := 12; GOSUB Imminent_to_rest;
240 i := 2; dk := 22; GOSUB Imminent_to_rest;
241 i := 3; dk := 32; GOSUB Imminent_to_rest;

```

```

235 i := 4; dk := 42; GOSUB Imminent_to_rest_LPI4;
236 i := 5; dk := 52; GOSUB Imminent_to_rest;
237 i := 6; dk := 62; GOSUB Imminent_to_rest;
238 i := 7; dk := 72; GOSUB Imminent_to_rest;
239 i := 8; dk := 82; GOSUB Imminent_to_rest_LPI8;
240
241 /*************************************************************************/
242 /* ---- E. Phase Sequencing and Barrier Logic ---- */
243
244 /* within-ring transitions */
245 IF rest_green[1] THEN
246   rest_green[1] := 0;
247   Sg_red(1);
248   imminent[2] := 1;
249 END;
250
251 IF rest_green[3] THEN
252   rest_green[3] := 0;
253   Sg_red(3);
254   imminent[4] := 1;
255 END;
256
257 IF rest_green[5] THEN
258   rest_green[5] := 0;
259   Sg_red(5);
260   imminent[6] := 1;
261 END;
262
263 IF rest_green[7] THEN
264   rest_green[7] := 0;
265   Sg_red(7);
266   imminent[8] := 1;
267 END;
268
269 /* ---- Barrier 1: between 2 & 6 ---- */
270 IF (rest_green[2] AND rest_green[6]) THEN
271   rest_green[2] := 0;
272   rest_green[6] := 0;
273   Sg_red(2);
274   Sg_red(6);
275   IF has_call[3] THEN
276     imminent[3] := 1;
277   ELSE

```

```

278     imminent[4] := 1;
279 END;
280 IF has_call[7] THEN
281     imminent[7] := 1;
282 ELSE
283     imminent[8] := 1;
284 END;
285 END;

286 /* ---- Barrier 2: between 4 & 8 ---- */
287 IF (rest_green[4] AND rest_green[8]) THEN
288     rest_green[4] := 0;
289     rest_green[8] := 0;
290     Sg_red(4);
291     Sg_red(8);
292     IF has_call[1] THEN
293         imminent[1] := 1;
294     ELSE
295         imminent[2] := 1;
296     END;
297     IF has_call[5] THEN
298         imminent[5] := 1;
299     ELSE
300         imminent[6] := 1;
301     END;
302 END;
303 END.

304 ****
305 /* END PROGRAM */
306 END.

```

B.2 Alternative 1: Green Extension

Listing 2: VAP code for Alternative 1: GE

```

1 PROGRAM actuated_8PHASES;
2 /* Actuated control for 8 phases (NEMA dual-ring)
3 Simple TSP extension on phases 4 & 8 ONLY:
4 - if bus arrives within last 10s before MAXGREEN, allow up to MAXGREEN
5     +10
6 - terminate immediately when bus hits check-out detector
7 */

```

```

8 VAP_Frequency 10;      /* execute logic 10 times per second */
9
10 ****
11 /* CONSTANTS */
12 CONST
13 CALLDELAY = 2,          /* seconds before detector registers call */
14 BUS_WINDOW = 10,        /* seconds before MAXGREEN to become eligible */
15 BUS_EXT = 10;          /* max additional seconds allowed beyond MAXGREEN */
16
17 ****
18 /* ARRAYS */
19 ARRAY
20 MAXGREEN[8] = [12, 26, 8, 32, 10, 26, 8, 30],
21 MINGAP10[8] = [30, 30, 30, 30, 30, 30, 30, 30],
22 RECALL[8] = [0,0,0,1,0,0,0,1],    /* recall on throughs (4 & 8 in your
   plan) */
23
24 has_call[8],
25 no_conflict[8],
26 confl_call[8],
27 imminent[8],
28 early_green[8],
29 ext_green[8],
30 rest_green[8],
31
32 /* TSP state (arrays only) */
33 bus_req[8],            /* 1 if bus is between check-in and check-out */
34 bus_window_ok[8];      /* 1 if within BUS_WINDOW seconds of MAXGREEN */
35
36
37 ****
38 /* SUBROUTINES */
39
40 /* ----- Locking Call ----- */
41 SUBROUTINE Locking_call;
42 IF Current_state(i, red) THEN
43   IF (RECALL[i] OR (Occupancy(dk) > CALLDELAY)) THEN
44     has_call[i] := 1;
45   END;
46 END.
47
48 /* ----- Non-Locking Call ----- */
49 SUBROUTINE NONLocking_call;

```

```

50  IF Current_state(i, red) THEN
51      IF (RECALL[i] OR (Occupancy(dk) > CALLDELAY)) THEN
52          has_call[i] := 1;
53      ELSE
54          has_call[i] := 0;
55      END;
56  END.

57
58 /* ----- Imminent-to-Rest (normal phases) ----- */
59 SUBROUTINE Imminent_to_rest;
60 IF (imminent[i] AND no_conflict[i]) THEN
61     Sg_green(i);
62     early_green[i] := 1;
63     imminent[i] := 0;
64     has_call[i] := 0;
65 END;

66
67 IF (early_green[i] AND confl_call[i] AND (T_green(i) >= T_green_min(i)))
68     THEN
69     early_green[i] := 0;
70     ext_green[i] := 1;
71 END;

72 IF (ext_green[i]) THEN
73     IF ((Headway10(dk) > MINGAP10[i]) OR (T_green(i) >= MAXGREEN[i])) THEN
74         ext_green[i] := 0;
75         rest_green[i] := 1;
76     END;
77 END.

78
79 /* ----- Imminent-to-Rest with LPI + TSP (phase 4) ----- */
80 SUBROUTINE Imminent_to_rest_LPI4;
81
82 IF (imminent[i] AND no_conflict[i]) THEN
83     Start(LPI4);
84     IF (LPI4 = 4) THEN
85         Sg_green(i);
86         early_green[i] := 1;
87         imminent[i] := 0;
88         has_call[i] := 0;
89         Stop(LPI4);
90         Reset(LPI4);
91     END;

```

```

92    END;

93

94 IF (early_green[i] AND confl_call[i] AND (T_green(i) >= T_green_min(i)))
95   THEN
96   early_green[i] := 0;
97   ext_green[i] := 1;
98 END;

99 IF (ext_green[i]) THEN
100
101 /* 1) If bus clears (check-out 44), terminate immediately */
102 IF (Occupancy(44) > 0) THEN
103   ext_green[i] := 0;
104   rest_green[i] := 1;
105 END;

106
107 /* 2) Hard stop always at MAXGREEN + BUS_EXT */
108 IF (ext_green[i] AND (T_green(i) >= (MAXGREEN[i] + BUS_EXT))) THEN
109   ext_green[i] := 0;
110   rest_green[i] := 1;
111 END;

112
113 /* 3) Normal max-out at MAXGREEN only if NOT eligible for TSP */
114 IF (ext_green[i] AND (T_green(i) >= MAXGREEN[i])) THEN
115   IF NOT (bus_req[i] AND bus_window_ok[i]) THEN
116     ext_green[i] := 0;
117     rest_green[i] := 1;
118   END;
119 END;

120
121 /* 4) OPTIONAL: keep normal gap-out when no bus is active */
122 IF (ext_green[i] AND NOT bus_req[i]) THEN
123   IF (Headway10(dk) > MINGAP10[i]) THEN
124     ext_green[i] := 0;
125     rest_green[i] := 1;
126   END;
127 END;

128
129 END.

130
131 /* ----- Imminent-to-Rest with LPI + TSP (phase 8) ----- */
132 SUBROUTINE Imminent_to_rest_LPI8;
133

```

```

134 IF (imminent[i] AND no_conflict[i]) THEN
135   Start(LPI8);
136   IF (LPI8 = 4) THEN
137     Sg_green(i);
138     early_green[i] := 1;
139     imminent[i] := 0;
140     has_call[i] := 0;
141     Stop(LPI8);
142     Reset(LPI8);
143   END;
144 END;
145
146 IF (early_green[i] AND confl_call[i] AND (T_green(i) >= T_green_min(i)))
147   THEN
148   early_green[i] := 0;
149   ext_green[i] := 1;
150 END;
151
152 IF (ext_green[i]) THEN
153   /* 1) If bus clears (check-out 84), terminate immediately */
154   IF (Occupancy(84) > 0) THEN
155     ext_green[i] := 0;
156     rest_green[i] := 1;
157   END;
158
159   /* 2) Hard stop always at MAXGREEN + BUS_EXT */
160   IF (ext_green[i] AND (T_green(i) >= (MAXGREEN[i] + BUS_EXT))) THEN
161     ext_green[i] := 0;
162     rest_green[i] := 1;
163   END;
164
165   /* 3) Normal max-out at MAXGREEN only if NOT eligible for TSP */
166   IF (ext_green[i] AND (T_green(i) >= MAXGREEN[i])) THEN
167     IF NOT (bus_req[i] AND bus_window_ok[i]) THEN
168       ext_green[i] := 0;
169       rest_green[i] := 1;
170     END;
171   END;
172
173   /* 4) OPTIONAL: keep normal gap-out when no bus is active */
174   IF (ext_green[i] AND NOT bus_req[i]) THEN
175     IF (Headway10(dk) > MINGAP10[i]) THEN

```

```

176     ext_green[i] := 0;
177     rest_green[i] := 1;
178   END;
179 END;
180
181 END.
182
183 /* ----- Initialize Early Green ----- */
184 SUBROUTINE Initialize_early_green;
185 IF Current_state(i, green) THEN
186   early_green[i] := 1;
187 END.
188
189
190 /*************************************************************************/
191 /* ===== MAIN PROGRAM ===== */
192
193 /* ---- Initialization ---- */
194 IF sim_timer = 0 THEN
195   START(sim_timer);
196 END;
197
198 IF sim_timer <= 5 THEN
199   i := 1; GOSUB Initialize_early_green;
200   i := 2; GOSUB Initialize_early_green;
201   i := 3; GOSUB Initialize_early_green;
202   i := 4; GOSUB Initialize_early_green;
203   i := 5; GOSUB Initialize_early_green;
204   i := 6; GOSUB Initialize_early_green;
205   i := 7; GOSUB Initialize_early_green;
206   i := 8; GOSUB Initialize_early_green;
207 END;
208
209 /*************************************************************************/
210 /* ---- A. Update Calls ---- */
211 i := 1; dk := 11; GOSUB NONLocking_call;
212 i := 2; dk := 21; GOSUB NONLocking_call;
213 i := 3; dk := 31; GOSUB NONLocking_call;
214 i := 4; dk := 41; GOSUB Locking_call;
215 i := 5; dk := 51; GOSUB NONLocking_call;
216 i := 6; dk := 61; GOSUB NONLocking_call;
217 i := 7; dk := 71; GOSUB NONLocking_call;
218 i := 8; dk := 81; GOSUB Locking_call;

```

```

219
220 /* **** */
221 /* ---- A1. SIMPLE BUS REQUEST (must be BEFORE transitions) ---- */
222
223 /* clear arrays each step (safe) */
224 bus_req[1] := 0; bus_req[2] := 0; bus_req[3] := 0; bus_req[4] := 0;
225 bus_req[5] := 0; bus_req[6] := 0; bus_req[7] := 0; bus_req[8] := 0;
226
227 bus_window_ok[1] := 0; bus_window_ok[2] := 0; bus_window_ok[3] := 0;
228     bus_window_ok[4] := 0;
229 bus_window_ok[5] := 0; bus_window_ok[6] := 0; bus_window_ok[7] := 0;
230     bus_window_ok[8] := 0;
231
232 /* Phase 8 (NB): check-in 83, check-out 84 */
233 bus_req[8] := (Occupancy(83) > 0) AND (Occupancy(84) = 0);
234
235 /* Phase 4 (SB): check-in 43, check-out 44 */
236 bus_req[4] := (Occupancy(43) > 0) AND (Occupancy(44) = 0);
237
238 /* Eligible only if bus present AND within last BUS_WINDOW seconds before
   MAXGREEN */
239 bus_window_ok[4] := (bus_req[4]) AND ((MAXGREEN[4] - T_green(4)) <=
240     BUS_WINDOW);
241 bus_window_ok[8] := (bus_req[8]) AND ((MAXGREEN[8] - T_green(8)) <=
242     BUS_WINDOW);
243
244 /* **** */
245 /* ---- B. Conflict-Status Logic (realistic NEMA) ---- */
246
247 no_conflict[1] :=
248     Current_state(2, red)*Current_state(3, red)*
249     Current_state(4, red)*Current_state(7, red)*
250     Current_state(8, red)*(Remaining_Intergreen(1)=0);
251
252 no_conflict[2] :=
253     Current_state(1, red)*Current_state(3, red)*
254     Current_state(4, red)*Current_state(7, red)*
255     Current_state(8, red)*(Remaining_Intergreen(2)=0);
256
257 no_conflict[3] :=
258     Current_state(1, red)*Current_state(2, red)*
259     Current_state(4, red)*Current_state(5, red)*
260     Current_state(6, red)*(Remaining_Intergreen(3)=0);

```

```

257 no_conflict[4] :=
258     Current_state(1, red)*Current_state(2, red)*
259     Current_state(3, red)*Current_state(5, red)*
260     Current_state(6, red)*(Remaining_Intergreen(4)=0);
261
262 no_conflict[5] :=
263     Current_state(6, red)*Current_state(7, red)*
264     Current_state(8, red)*Current_state(3, red)*
265     Current_state(4, red)*(Remaining_Intergreen(5)=0);
266
267 no_conflict[6] :=
268     Current_state(5, red)*Current_state(7, red)*
269     Current_state(8, red)*Current_state(3, red)*
270     Current_state(4, red)*(Remaining_Intergreen(6)=0);
271
272 no_conflict[7] :=
273     Current_state(5, red)*Current_state(6, red)*
274     Current_state(8, red)*Current_state(1, red)*
275     Current_state(2, red)*(Remaining_Intergreen(7)=0);
276
277 no_conflict[8] :=
278     Current_state(5, red)*Current_state(6, red)*
279     Current_state(7, red)*Current_state(1, red)*
280     Current_state(2, red)*(Remaining_Intergreen(8)=0);
281
282 /*************************************************************************/
283 /* ---- C. Conflicting Calls ---- */
284 confl_call[1] := has_call[2] OR has_call[3] OR has_call[4] OR has_call[7]
285     OR has_call[8];
286 confl_call[2] := has_call[1] OR has_call[3] OR has_call[4] OR has_call[7]
287     OR has_call[8];
288 confl_call[3] := has_call[1] OR has_call[2] OR has_call[4] OR has_call[5]
289     OR has_call[6];
290 confl_call[4] := has_call[1] OR has_call[2] OR has_call[3] OR has_call[5]
291     OR has_call[6];
292 confl_call[5] := has_call[3] OR has_call[4] OR has_call[6] OR has_call[7]
293     OR has_call[8];
294 confl_call[6] := has_call[3] OR has_call[4] OR has_call[5] OR has_call[7]
295     OR has_call[8];
296 confl_call[7] := has_call[1] OR has_call[2] OR has_call[5] OR has_call[6]
297     OR has_call[8];
298 confl_call[8] := has_call[1] OR has_call[2] OR has_call[5] OR has_call[6]
299     OR has_call[7];

```

```

292
293 /* **** */
294 /* ---- D. Phase Transition Evaluation ---- */
295 i := 1; dk := 12; GOSUB Imminent_to_rest;
296 i := 2; dk := 22; GOSUB Imminent_to_rest;
297 i := 3; dk := 32; GOSUB Imminent_to_rest;
298 i := 4; dk := 42; GOSUB Imminent_to_rest_LPI4;
299 i := 5; dk := 52; GOSUB Imminent_to_rest;
300 i := 6; dk := 62; GOSUB Imminent_to_rest;
301 i := 7; dk := 72; GOSUB Imminent_to_rest;
302 i := 8; dk := 82; GOSUB Imminent_to_rest_LPI8;
303
304 /* **** */
305 /* ---- OPTIONAL: Dummy signals to verify TSP visually ---- */
306 /* SG 9: any bus check-in active */
307 IF ((Occupancy(83) > 0) OR (Occupancy(43) > 0)) THEN
308     Sg_green(9);
309 ELSE
310     Sg_red(9);
311 END;
312
313 /* SG10: bus_req active (bus between check-in and check-out) */
314 IF (bus_req[4] OR bus_req[8]) THEN
315     Sg_green(10);
316 ELSE
317     Sg_red(10);
318 END;
319
320 /* SG11: TSP eligible window true */
321 IF (bus_window_ok[4] OR bus_window_ok[8]) THEN
322     Sg_green(11);
323 ELSE
324     Sg_red(11);
325 END;
326
327 /* SG12: check-out hit */
328 IF ((Occupancy(84) > 0) OR (Occupancy(44) > 0)) THEN
329     Sg_green(12);
330 ELSE
331     Sg_red(12);
332 END;
333
334 /* **** */

```

```

335 /* ---- E. Phase Sequencing and Barrier Logic ---- */
336
337 /* within-ring transitions */
338 IF rest_green[1] THEN
339     rest_green[1] := 0;
340     Sg_red(1);
341     imminent[2] := 1;
342 END;
343
344 IF rest_green[3] THEN
345     rest_green[3] := 0;
346     Sg_red(3);
347     imminent[4] := 1;
348 END;
349
350 IF rest_green[5] THEN
351     rest_green[5] := 0;
352     Sg_red(5);
353     imminent[6] := 1;
354 END;
355
356 IF rest_green[7] THEN
357     rest_green[7] := 0;
358     Sg_red(7);
359     imminent[8] := 1;
360 END;
361
362 /* ---- Barrier 1: between 2 & 6 ---- */
363 IF (rest_green[2] AND rest_green[6]) THEN
364     rest_green[2] := 0;
365     rest_green[6] := 0;
366     Sg_red(2);
367     Sg_red(6);
368
369     IF has_call[3] THEN
370         imminent[3] := 1;
371     ELSE
372         imminent[4] := 1;
373     END;
374
375     IF has_call[7] THEN
376         imminent[7] := 1;
377     ELSE

```

```

378     imminent[8] := 1;
379   END;
380 END;
381
382 /* ---- Barrier 2: between 4 & 8 ---- */
383 IF (rest_green[4] AND rest_green[8]) THEN
384   rest_green[4] := 0;
385   rest_green[8] := 0;
386   Sg_red(4);
387   Sg_red(8);
388
389   IF has_call[1] THEN
390     imminent[1] := 1;
391   ELSE
392     imminent[2] := 1;
393   END;
394
395   IF has_call[5] THEN
396     imminent[5] := 1;
397   ELSE
398     imminent[6] := 1;
399   END;
400 END.
401
402 /*************************************************************************/
403 /* END PROGRAM */
404 END.

```

B.3 Alternative 2: Green Extension with Phase Rotation

Listing 3: VAP code for Alternative 2: GE + Phase Rotation

```

1 PROGRAM actuated_8PHASES;
2 /* Actuated 8-phase NEMA dual-ring
3 - TSP green extension on phases 4 & 8:
4   If bus is ACTIVE and within last BUS_WINDOW seconds of MAXGREEN,
5   allow green up to MAXGREEN + BUS_EXT
6   End immediately when bus hits check-out detector
7 - Phase rotation (lead -> lag) at barrier:
8   If bus is ACTIVE when Barrier 1 releases, serve through first:
9     ring1: 4 then 3 (barrier ends on 3)
10    ring2: 8 then 7 (barrier ends on 7)
11 */

```

```

12
13 VAP_Frequency 10;
14
15 /***** */
16 /* CONSTANTS */
17 CONST
18 CALLDELAY = 2,
19 BUS_WINDOW = 10,
20 BUS_EXT = 10;
21
22 /***** */
23 /* ARRAYS */
24 ARRAY
25 MAXGREEN[8] = [12, 26, 8, 32, 10, 26, 8, 30],
26 MINGAP10[8] = [30, 30, 30, 30, 30, 30, 30, 30],
27
28 /* recall: if phase 4 & 8 are on recall, set RECALL[4]=1 and RECALL[8]=1
   */
29 RECALL[8] = [0,0,0,1,0,0,0,1],
30
31 has_call[8],
32 no_conflict[8],
33 confl_call[8],
34 imminent[8],
35 early_green[8],
36 ext_green[8],
37 rest_green[8],
38
39 /* bus state */
40 bus_active[8], /* latched: 1 from check-in to check-out */
41 bus_window_ok[8], /* 1 if eligible to extend */
42
43 /* rotation flags (use arrays to avoid VAP variable errors) */
44 rotate34[1], /* 1 => serve 4 then 3, barrier ends on 3 */
45 rotate78[1]; /* 1 => serve 8 then 7, barrier ends on 7 */
46
47
48 /***** */
49 /* SUBROUTINES */
50
51 /* ----- Locking Call ----- */
52 SUBROUTINE Locking_call;
53 IF Current_state(i, red) THEN

```

```

54     IF (RECALL[i] OR (Occupancy(dk) > CALLDELAY)) THEN
55         has_call[i] := 1;
56     END ;
57 END .

58
59 /* ----- Non-Locking Call ----- */
60 SUBROUTINE NONLocking_call;
61 IF Current_state(i, red) THEN
62     IF (RECALL[i] OR (Occupancy(dk) > CALLDELAY)) THEN
63         has_call[i] := 1;
64     ELSE
65         has_call[i] := 0;
66     END ;
67 END .

68
69 /* ----- Initialize Early Green ----- */
70 SUBROUTINE Initialize_early_green;
71 IF Current_state(i, green) THEN
72     early_green[i] := 1;
73 END .

74
75 /* ----- Imminent-to-Rest (normal phases) ----- */
76 SUBROUTINE Imminent_to_rest;
77 IF (imminent[i] AND no_conflict[i]) THEN
78     Sg_green(i);
79     early_green[i] := 1;
80     imminent[i] := 0;
81     has_call[i] := 0;
82 END ;

83
84 IF (early_green[i] AND confl_call[i] AND (T_green(i) >= T_green_min(i)))
85     THEN
86     early_green[i] := 0;
87     ext_green[i] := 1;
88 END ;

89 IF (ext_green[i]) THEN
90     IF ((Headway10(dk) > MINGAP10[i]) OR (T_green(i) >= MAXGREEN[i])) THEN
91         ext_green[i] := 0;
92         rest_green[i] := 1;
93     END ;
94 END .
95

```

```

96  /* ----- Phase 4 with TSP extension ----- */
97  SUBROUTINE Imminent_to_rest_TSP4;
98  IF (imminent[i] AND no_conflict[i]) THEN
99    Sg_green(i);
100   early_green[i] := 1;
101   imminent[i] := 0;
102   has_call[i] := 0;
103 END;
104
105 IF (early_green[i] AND confl_call[i] AND (T_green(i) >= T_green_min(i)))
106   THEN
107   early_green[i] := 0;
108   ext_green[i] := 1;
109 END;
110
111 IF (ext_green[i]) THEN
112   /* end immediately when bus checks out */
113   IF (Occupancy(44) > 0) THEN
114     ext_green[i] := 0;
115     rest_green[i] := 1;
116   END;
117
118   /* hard max always */
119   IF (ext_green[i] AND (T_green(i) >= (MAXGREEN[i] + BUS_EXT))) THEN
120     ext_green[i] := 0;
121     rest_green[i] := 1;
122   END;
123
124   /* at MAXGREEN: end only if NOT eligible */
125   IF (ext_green[i] AND (T_green(i) >= MAXGREEN[i])) THEN
126     IF NOT bus_window_ok[i] THEN
127       ext_green[i] := 0;
128       rest_green[i] := 1;
129     END;
130   END;
131
132   /* optional: allow gap-out only if not active bus */
133   IF (ext_green[i] AND (bus_active[i] = 0)) THEN
134     IF (Headway10(dk) > MINGAP10[i]) THEN
135       ext_green[i] := 0;
136       rest_green[i] := 1;
137     END;

```

```

138     END ;
139
140 END .
141
142 /* ----- Phase 8 with TSP extension ----- */
143 SUBROUTINE Imminent_to_rest_TSP8;
144 IF (imminent[i] AND no_conflict[i]) THEN
145     Sg_green(i);
146     early_green[i] := 1;
147     imminent[i] := 0;
148     has_call[i] := 0;
149 END ;
150
151 IF (early_green[i] AND confl_call[i] AND (T_green(i) >= T_green_min(i)))
152     THEN
153     early_green[i] := 0;
154     ext_green[i] := 1;
155 END ;
156
157 IF (ext_green[i]) THEN
158     /* end immediately when bus checks out */
159     IF (Occupancy(84) > 0) THEN
160         ext_green[i] := 0;
161         rest_green[i] := 1;
162     END ;
163
164     /* hard max always */
165     IF (ext_green[i] AND (T_green(i) >= (MAXGREEN[i] + BUS_EXT))) THEN
166         ext_green[i] := 0;
167         rest_green[i] := 1;
168     END ;
169
170     /* at MAXGREEN: end only if NOT eligible */
171     IF (ext_green[i] AND (T_green(i) >= MAXGREEN[i])) THEN
172         IF NOT bus_window_ok[i] THEN
173             ext_green[i] := 0;
174             rest_green[i] := 1;
175         END ;
176     END ;
177
178     /* optional: allow gap-out only if not active bus */
179     IF (ext_green[i] AND (bus_active[i] = 0)) THEN

```

```

180     IF (Headway10(dk) > MINGAP10[i]) THEN
181         ext_green[i] := 0;
182         rest_green[i] := 1;
183     END;
184 END;
185
186 END.
187
188
189 /************************************************************************
190 /* ===== MAIN PROGRAM ===== */
191
192 /* ---- Initialization ---- */
193 IF sim_timer = 0 THEN
194     START(sim_timer);
195 END;
196
197 IF sim_timer <= 5 THEN
198     i := 1; GOSUB Initialize_early_green;
199     i := 2; GOSUB Initialize_early_green;
200     i := 3; GOSUB Initialize_early_green;
201     i := 4; GOSUB Initialize_early_green;
202     i := 5; GOSUB Initialize_early_green;
203     i := 6; GOSUB Initialize_early_green;
204     i := 7; GOSUB Initialize_early_green;
205     i := 8; GOSUB Initialize_early_green;
206
207 /* safe init */
208 rotate34[1] := 0;
209 rotate78[1] := 0;
210 bus_active[4] := 0;
211 bus_active[8] := 0;
212 END;
213
214
215 /************************************************************************
216 /* ---- A. Update Calls ---- */
217 i := 1; dk := 11; GOSUB NONLocking_call;
218 i := 2; dk := 21; GOSUB NONLocking_call;
219 i := 3; dk := 31; GOSUB NONLocking_call;
220 i := 4; dk := 41; GOSUB Locking_call;
221 i := 5; dk := 51; GOSUB NONLocking_call;
222 i := 6; dk := 61; GOSUB NONLocking_call;

```

```

223 i := 7; dk := 71; GOSUB NONLocking_call;
224 i := 8; dk := 81; GOSUB Locking_call;
225
226
227 /****** */
228 /* ---- A1. BUS LATCH + WINDOW ---- */
229
230 /* latch bus presence (SB: 43 in, 44 out) */
231 IF (Occupancy(43) > 0) THEN
232   bus_active[4] := 1;
233 END;
234 IF (Occupancy(44) > 0) THEN
235   bus_active[4] := 0;
236 END;
237
238 /* latch bus presence (NB: 83 in, 84 out) */
239 IF (Occupancy(83) > 0) THEN
240   bus_active[8] := 1;
241 END;
242 IF (Occupancy(84) > 0) THEN
243   bus_active[8] := 0;
244 END;
245
246 /* window eligibility only matters for phases 4 & 8 */
247 bus_window_ok[1] := 0; bus_window_ok[2] := 0; bus_window_ok[3] := 0;
248 bus_window_ok[5] := 0; bus_window_ok[6] := 0; bus_window_ok[7] := 0;
249
250 bus_window_ok[4] := (bus_active[4] = 1) * ((MAXGREEN[4] - T_green(4)) <=
251   BUS_WINDOW);
251 bus_window_ok[8] := (bus_active[8] = 1) * ((MAXGREEN[8] - T_green(8)) <=
252   BUS_WINDOW);
253
254 /****** */
255 /* ---- B. Conflict-Status Logic (same as your base) ---- */
256 no_conflict[1] :=
257   Current_state(2, red)*Current_state(3, red)*
258   Current_state(4, red)*Current_state(7, red)*
259   Current_state(8, red)*(Remaining_Intergreen(1)=0);
260
261 no_conflict[2] :=
262   Current_state(1, red)*Current_state(3, red)*
263   Current_state(4, red)*Current_state(7, red)*

```

```

264     Current_state(8, red)*(Remaining_Intergreen(2)=0);
265
266 no_conflict[3] :=
267     Current_state(1, red)*Current_state(2, red)*
268     Current_state(4, red)*Current_state(5, red)*
269     Current_state(6, red)*(Remaining_Intergreen(3)=0);
270
271 no_conflict[4] :=
272     Current_state(1, red)*Current_state(2, red)*
273     Current_state(3, red)*Current_state(5, red)*
274     Current_state(6, red)*(Remaining_Intergreen(4)=0);
275
276 no_conflict[5] :=
277     Current_state(6, red)*Current_state(7, red)*
278     Current_state(8, red)*Current_state(3, red)*
279     Current_state(4, red)*(Remaining_Intergreen(5)=0);
280
281 no_conflict[6] :=
282     Current_state(5, red)*Current_state(7, red)*
283     Current_state(8, red)*Current_state(3, red)*
284     Current_state(4, red)*(Remaining_Intergreen(6)=0);
285
286 no_conflict[7] :=
287     Current_state(5, red)*Current_state(6, red)*
288     Current_state(8, red)*Current_state(1, red)*
289     Current_state(2, red)*(Remaining_Intergreen(7)=0);
290
291 no_conflict[8] :=
292     Current_state(5, red)*Current_state(6, red)*
293     Current_state(7, red)*Current_state(1, red)*
294     Current_state(2, red)*(Remaining_Intergreen(8)=0);
295
296
297 /*****C. Conflicting Calls *****/
298 /* ---- C. Conflicting Calls ---- */
299 confl_call[1] := has_call[2] OR has_call[3] OR has_call[4] OR has_call[7]
300           OR has_call[8];
301 confl_call[2] := has_call[1] OR has_call[3] OR has_call[4] OR has_call[7]
302           OR has_call[8];
303 confl_call[3] := has_call[1] OR has_call[2] OR has_call[4] OR has_call[5]
304           OR has_call[6];
305 confl_call[4] := has_call[1] OR has_call[2] OR has_call[3] OR has_call[5]
306           OR has_call[6];

```

```

303 confl_call[5] := has_call[3] OR has_call[4] OR has_call[6] OR has_call[7]
      OR has_call[8];
304 confl_call[6] := has_call[3] OR has_call[4] OR has_call[5] OR has_call[7]
      OR has_call[8];
305 confl_call[7] := has_call[1] OR has_call[2] OR has_call[5] OR has_call[6]
      OR has_call[8];
306 confl_call[8] := has_call[1] OR has_call[2] OR has_call[5] OR has_call[6]
      OR has_call[7];
307
308
309 /************************************************************************
310 /* ---- D. Phase Transition Evaluation ---- */
311 i := 1; dk := 12; GOSUB Imminent_to_rest;
312 i := 2; dk := 22; GOSUB Imminent_to_rest;
313 i := 3; dk := 32; GOSUB Imminent_to_rest;
314
315 i := 4; dk := 42; GOSUB Imminent_to_rest_TSP4;
316
317 i := 5; dk := 52; GOSUB Imminent_to_rest;
318 i := 6; dk := 62; GOSUB Imminent_to_rest;
319 i := 7; dk := 72; GOSUB Imminent_to_rest;
320
321 i := 8; dk := 82; GOSUB Imminent_to_rest_TSP8;
322
323 /************************************************************************
324 /* ---- OPTIONAL: Dummy signals for TSP visualization ---- */
325 /************************************************************************
326
327 /* SG 9: Bus detected at upstream check-in */
328 IF (Occupancy(43) > 0) THEN
329   Sg_green(9);
330 ELSE
331   IF (Occupancy(83) > 0) THEN
332     Sg_green(9);
333   ELSE
334     Sg_red(9);
335   END;
336 END;
337
338
339 /* SG 10: Bus active (latched between check-in and check-out) */
340 IF (bus_active[4] = 1) THEN
341   Sg_green(10);

```

```

342 ELSE
343   IF (bus_active[8] = 1) THEN
344     Sg_green(10);
345   ELSE
346     Sg_red(10);
347   END;
348 END;

349

350

351 /* SG 11: Bus eligible for green extension (window OK) */
352 IF (bus_window_ok[4] = 1) THEN
353   Sg_green(11);
354 ELSE
355   IF (bus_window_ok[8] = 1) THEN
356     Sg_green(11);
357   ELSE
358     Sg_red(11);
359   END;
360 END;

361

362

363 /* SG 12: Bus cleared intersection (check-out hit) */
364 IF (Occupancy(44) > 0) THEN
365   Sg_green(12);
366 ELSE
367   IF (Occupancy(84) > 0) THEN
368     Sg_green(12);
369   ELSE
370     Sg_red(12);
371   END;
372 END;

373

374

375

376

377

378

379 ****
380 /* ---- E. Phase Sequencing and Barrier Logic ---- */
381
382 /* 1 -> 2 */
383 IF rest_green[1] THEN
384   rest_green[1] := 0;

```

```

385     Sg_red(1);
386     imminent[2] := 1;
387 END;
388
389 /* 5 -> 6 */
390 IF rest_green[5] THEN
391   rest_green[5] := 0;
392   Sg_red(5);
393   imminent[6] := 1;
394 END;
395
396 /* default lead-left transitions */
397 IF rest_green[3] THEN
398   IF (rotate34[1] = 0) THEN
399     rest_green[3] := 0;
400     Sg_red(3);
401     imminent[4] := 1;
402   END;
403 END;
404
405 IF rest_green[7] THEN
406   IF (rotate78[1] = 0) THEN
407     rest_green[7] := 0;
408     Sg_red(7);
409     imminent[8] := 1;
410   END;
411 END;
412
413 /* rotated lag-left transitions */
414 IF rest_green[4] THEN
415   IF (rotate34[1] = 1) THEN
416     rest_green[4] := 0;
417     Sg_red(4);
418     imminent[3] := 1;
419   END;
420 END;
421
422 IF rest_green[8] THEN
423   IF (rotate78[1] = 1) THEN
424     rest_green[8] := 0;
425     Sg_red(8);
426     imminent[7] := 1;
427   END;

```

```

428 END;
429
430
431 /* ---- Barrier 1: between 2 & 6 ---- */
432 IF (rest_green[2] AND rest_green[6]) THEN
433   rest_green[2] := 0;
434   rest_green[6] := 0;
435   Sg_red(2);
436   Sg_red(6);
437
438 /* decide rotation ONCE at barrier release */
439 rotate34[1] := (bus_active[4] = 1) * (has_call[3] = 1);
440 rotate78[1] := (bus_active[8] = 1) * (has_call[7] = 1);
441
442 /* ring1 start */
443 IF (rotate34[1] = 1) THEN
444   imminent[4] := 1;
445 ELSE
446   IF has_call[3] THEN
447     imminent[3] := 1;
448   ELSE
449     imminent[4] := 1;
450   END;
451 END;
452
453 /* ring2 start */
454 IF (rotate78[1] = 1) THEN
455   imminent[8] := 1;
456 ELSE
457   IF has_call[7] THEN
458     imminent[7] := 1;
459   ELSE
460     imminent[8] := 1;
461   END;
462 END;
463
464 END;
465
466
467 /* ---- Barrier 2 (dynamic end) ---- */
468 IF ( ((rotate34[1] = 1) AND rest_green[3]) OR ((rotate34[1] = 0) AND
469   rest_green[4]) ) THEN

```

```

469  IF ((rotate78[1] = 1) AND rest_green[7]) OR ((rotate78[1] = 0) AND
470    rest_green[8]) ) THEN
471
472    /* clear barrier-ending phases */
473    IF (rotate34[1] = 1) THEN
474      rest_green[3] := 0;
475      Sg_red(3);
476    ELSE
477      rest_green[4] := 0;
478      Sg_red(4);
479    END;
480
481    IF (rotate78[1] = 1) THEN
482      rest_green[7] := 0;
483      Sg_red(7);
484    ELSE
485      rest_green[8] := 0;
486      Sg_red(8);
487    END;
488
489    /* reset rotation */
490    rotate34[1] := 0;
491    rotate78[1] := 0;
492
493    /* next starts (same as base) */
494    IF has_call[1] THEN
495      imminent[1] := 1;
496    ELSE
497      imminent[2] := 1;
498    END;
499
500    IF has_call[5] THEN
501      imminent[5] := 1;
502    ELSE
503      imminent[6] := 1;
504    END;
505  END;
506 END.
507
508 ****
509 END.

```