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

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.

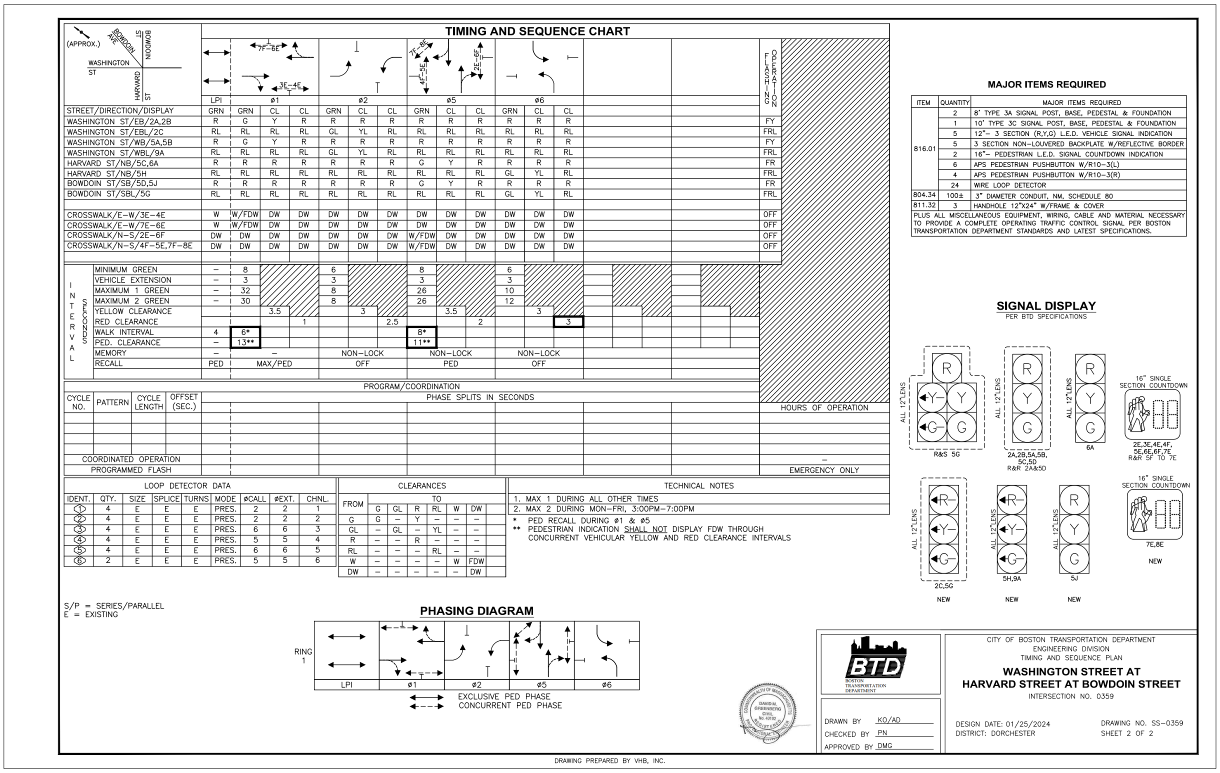

Figure 1: City of Boston Timing and Sequence Chart for the Washington Street–Bowdoin Street–Harvard Street intersection

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

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

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

/*******************************************************/
/* CONSTANTS */
CONST
CALLDELAY = 2;      /* seconds before detector registers call */

/*******************************************************/
/* ARRAYS */
ARRAY
MAXGREEN[8] = [12, 26, 8, 32, 10, 26, 8, 30],
MINGAP10[8] = [30, 30, 30, 30, 30, 30, 30, 30],
RECALL[8]   = [0,0,0,1,0,0,0,1],   /* recall on throughs */
has_call[8],
no_conflict[8],
confl_call[8],
imminent[8],
early_green[8],
ext_green[8],
rest_green[8];

/*******************************************************/
/* SUBROUTINES */

/* ---------- Locking Call ---------- */
SUBROUTINE Locking_call;
/* sets a locking call with a delay of CALLDELAY seconds */
IF Current_state(i, red) THEN
```

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



SUBROUTINE NONLocking_call;
        /* Sets a locking call with a delay of CALLDELAY seconds */
        /* Implicit inputs: i (signal group), dk (call detector), RECALL[] */

IF Current_state(i, red)  THEN                          /* Current state() is a VAP function  */
   IF (RECALL[i] OR (Occupancy(dk)> CALLDELAY))  THEN   /* if RECALL is true, dk's␣value␣won't matter */
      has_call[i] := 1;
   ELSE
      has_call[i] :=0;
   END
END.   /* The period ends the subroutine */


/*************************************************************/
/* ---------- Imminent-to-Rest Transition ---------- */
SUBROUTINE Imminent_to_rest;
/* transitions between states from imminent to rest_green */
IF (imminent[i] AND no_conflict[i]) THEN
   Sg_green(i);
   early_green[i] := 1;
   imminent[i] := 0;
   has_call[i] := 0;
END;

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

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

```
      rest_green[i] := 1;
   END;
END.


SUBROUTINE Imminent_to_rest_LPI4;
/* transitions between states from imminent to rest_green */
IF (imminent[i] AND no_conflict[i]) THEN
   Start(LPI4);
   IF (LPI4=4) THEN
        Sg_green(i);
        early_green[i] := 1;
        imminent[i] := 0;
        has_call[i] := 0;
        Stop(LPI4);
        Reset(LPI4);
   END;
END;

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

IF (ext_green[i]) THEN
   IF ((Headway10(dk) > MINGAP10[i]) OR (T_green(i) >= MAXGREEN[i])) THEN
      ext_green[i] := 0;
      rest_green[i] := 1;
   END;
END.




SUBROUTINE Imminent_to_rest_LPI8;
/* transitions between states from imminent to rest_green */
IF (imminent[i] AND no_conflict[i]) THEN
   Start(LPI8);
   IF (LPI8=4) THEN
        Sg_green(i);
        early_green[i] := 1;
        imminent[i] := 0;
```

```
        has_call[i] := 0;
        Stop(LPI8);
        Reset(LPI8);
   END;
END;

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

IF (ext_green[i]) THEN
   IF ((Headway10(dk) > MINGAP10[i]) OR (T_green(i) >= MAXGREEN[i])) THEN
      ext_green[i] := 0;
      rest_green[i] := 1;
   END;
END.



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

/*************************************************************/
/* =============== MAIN PROGRAM ============================ */

/* ---- Initialization ---- */
IF sim_timer = 0 THEN
   START(sim_timer);
END;

IF sim_timer <= 5 THEN
   i := 1; GOSUB Initialize_early_green;
        i := 2; GOSUB Initialize_early_green;
   i := 3; GOSUB Initialize_early_green;
        i := 4; GOSUB Initialize_early_green;
```

```
    i := 5; GOSUB Initialize_early_green;
         i := 6; GOSUB Initialize_early_green;
    i := 7; GOSUB Initialize_early_green;
         i := 8; GOSUB Initialize_early_green;
END;


/************************************************************/
/* ---- A. Update Calls ---- */
i := 1; dk := 11; GOSUB NONLocking_call;
i := 2; dk := 21; GOSUB NONLocking_call;
i := 3; dk := 31; GOSUB NONLocking_call;
i := 4; dk := 41; GOSUB Locking_call;
i := 5; dk := 51; GOSUB NONLocking_call;
i := 6; dk := 61; GOSUB NONLocking_call;
i := 7; dk := 71; GOSUB NONLocking_call;
i := 8; dk := 81; GOSUB Locking_call;


/************************************************************/
/* ---- B. Conflict-Status Logic (realistic NEMA) ---- */
no_conflict[1] :=
    Current_state(2, red)*Current_state(3, red)*
    Current_state(4, red)*Current_state(7, red)*
    Current_state(8, red)*(Remaining_Intergreen(1)=0);

no_conflict[2] :=
    Current_state(1, red)*Current_state(3, red)*
    Current_state(4, red)*Current_state(7, red)*
    Current_state(8, red)*(Remaining_Intergreen(2)=0);

no_conflict[3] :=
    Current_state(1, red)*Current_state(2, red)*
    Current_state(4, red)*Current_state(5, red)*
    Current_state(6, red)*(Remaining_Intergreen(3)=0);

no_conflict[4] :=
    Current_state(1, red)*Current_state(2, red)*
    Current_state(3, red)*Current_state(5, red)*
    Current_state(6, red)*(Remaining_Intergreen(4)=0);

no_conflict[5] :=
    Current_state(6, red)*Current_state(7, red)*
```

```
    Current_state(8, red)*Current_state(3, red)*
    Current_state(4, red)*(Remaining_Intergreen(5)=0);

no_conflict[6] :=
    Current_state(5, red)*Current_state(7, red)*
    Current_state(8, red)*Current_state(3, red)*
    Current_state(4, red)*(Remaining_Intergreen(6)=0);

no_conflict[7] :=
    Current_state(5, red)*Current_state(6, red)*
    Current_state(8, red)*Current_state(1, red)*
    Current_state(2, red)*(Remaining_Intergreen(7)=0);

no_conflict[8] :=
    Current_state(5, red)*Current_state(6, red)*
    Current_state(7, red)*Current_state(1, red)*
    Current_state(2, red)*(Remaining_Intergreen(8)=0);

/************************************************************/
/* ---- C. Conflicting Calls ---- */
confl_call[1] := has_call[2] OR has_call[3] OR has_call[4] OR has_call[7]
   OR has_call[8];
confl_call[2] := has_call[1] OR has_call[3] OR has_call[4] OR has_call[7]
   OR has_call[8];
confl_call[3] := has_call[1] OR has_call[2] OR has_call[4] OR has_call[5]
   OR has_call[6];
confl_call[4] := has_call[1] OR has_call[2] OR has_call[3] OR has_call[5]
   OR has_call[6];
confl_call[5] := has_call[3] OR has_call[4] OR has_call[6] OR has_call[7]
   OR has_call[8];
confl_call[6] := has_call[3] OR has_call[4] OR has_call[5] OR has_call[7]
   OR has_call[8];
confl_call[7] := has_call[1] OR has_call[2] OR has_call[5] OR has_call[6]
   OR has_call[8];
confl_call[8] := has_call[1] OR has_call[2] OR has_call[5] OR has_call[6]
   OR has_call[7];


/************************************************************/
/* ---- D. Phase Transition Evaluation ---- */
i := 1; dk := 12; GOSUB Imminent_to_rest;
i := 2; dk := 22; GOSUB Imminent_to_rest;
i := 3; dk := 32; GOSUB Imminent_to_rest;
```

```
i := 4; dk := 42; GOSUB Imminent_to_rest_LPI4;
i := 5; dk := 52; GOSUB Imminent_to_rest;
i := 6; dk := 62; GOSUB Imminent_to_rest;
i := 7; dk := 72; GOSUB Imminent_to_rest;
i := 8; dk := 82; GOSUB Imminent_to_rest_LPI8;

/*********************************************************/
/* ---- E. Phase Sequencing and Barrier Logic ---- */

/* within-ring transitions */
IF rest_green[1] THEN
   rest_green[1] := 0;
   Sg_red(1);
   imminent[2] := 1;
END;

IF rest_green[3] THEN
   rest_green[3] := 0;
   Sg_red(3);
   imminent[4] := 1;
END;

IF rest_green[5] THEN
   rest_green[5] := 0;
   Sg_red(5);
   imminent[6] := 1;
END;

IF rest_green[7] THEN
   rest_green[7] := 0;
   Sg_red(7);
   imminent[8] := 1;
END;

/* ---- Barrier 1: between 2 & 6 ---- */
IF (rest_green[2] AND rest_green[6]) THEN
   rest_green[2] := 0;
   rest_green[6] := 0;
   Sg_red(2);
   Sg_red(6);
   IF has_call[3] THEN
      imminent[3] := 1;
   ELSE
```

```
        imminent[4] := 1;
     END;
     IF has_call[7] THEN
        imminent[7] := 1;
     ELSE
        imminent[8] := 1;
     END;
END;

/* ---- Barrier 2: between 4 & 8 ---- */
IF (rest_green[4] AND rest_green[8]) THEN
     rest_green[4] := 0;
     rest_green[8] := 0;
     Sg_red(4);
     Sg_red(8);
     IF has_call[1] THEN
        imminent[1] := 1;
     ELSE
        imminent[2] := 1;
     END;
     IF has_call[5] THEN
        imminent[5] := 1;
     ELSE
        imminent[6] := 1;
     END;
END.

/*************************************************************/
/* END PROGRAM */
END.
```

B.2 Alternative 1: Green Extension

Listing 2: VAP code for Alternative 1: GE

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

```

```
VAP_Frequency 10;   /* execute logic 10 times per second */

/*************************************************************/
/* CONSTANTS */
CONST
CALLDELAY = 2,      /* seconds before detector registers call */
BUS_WINDOW = 10,    /* seconds before MAXGREEN to become eligible */
BUS_EXT = 10;       /* max additional seconds allowed beyond MAXGREEN */

/*************************************************************/
/* ARRAYS */
ARRAY
MAXGREEN[8] = [12, 26, 8, 32, 10, 26, 8, 30],
MINGAP10[8] = [30, 30, 30, 30, 30, 30, 30, 30],
RECALL[8]   = [0,0,0,1,0,0,0,1],   /* recall on throughs (4 & 8 in your
   plan) */

has_call[8],
no_conflict[8],
confl_call[8],
imminent[8],
early_green[8],
ext_green[8],
rest_green[8],

/* TSP state (arrays only) */
bus_req[8],          /* 1 if bus is between check-in and check-out */
bus_window_ok[8];    /* 1 if within BUS_WINDOW seconds of MAXGREEN */


/*************************************************************/
/* SUBROUTINES */

/* ---------- Locking Call ---------- */
SUBROUTINE Locking_call;
IF Current_state(i, red) THEN
   IF (RECALL[i] OR (Occupancy(dk) > CALLDELAY)) THEN
      has_call[i] := 1;
   END;
END.

/* ---------- Non-Locking Call ---------- */
SUBROUTINE NONLocking_call;
```

```
IF Current_state(i, red) THEN
   IF (RECALL[i] OR (Occupancy(dk) > CALLDELAY)) THEN
      has_call[i] := 1;
   ELSE
      has_call[i] := 0;
   END;
END.

/* ---------- Imminent-to-Rest (normal phases) ---------- */
SUBROUTINE Imminent_to_rest;
IF (imminent[i] AND no_conflict[i]) THEN
   Sg_green(i);
   early_green[i] := 1;
   imminent[i] := 0;
   has_call[i] := 0;
END;

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

IF (ext_green[i]) THEN
   IF ((Headway10(dk) > MINGAP10[i]) OR (T_green(i) >= MAXGREEN[i])) THEN
      ext_green[i] := 0;
      rest_green[i] := 1;
   END;
END.

/* ---------- Imminent-to-Rest with LPI + TSP (phase 4) ---------- */
SUBROUTINE Imminent_to_rest_LPI4;

IF (imminent[i] AND no_conflict[i]) THEN
   Start(LPI4);
   IF (LPI4 = 4) THEN
      Sg_green(i);
      early_green[i] := 1;
      imminent[i] := 0;
      has_call[i] := 0;
      Stop(LPI4);
      Reset(LPI4);
   END;
```

```
END;

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

IF (ext_green[i]) THEN

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

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

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

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

END.

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

```

```
IF (imminent[i] AND no_conflict[i]) THEN
   Start(LPI8);
   IF (LPI8 = 4) THEN
      Sg_green(i);
      early_green[i] := 1;
      imminent[i] := 0;
      has_call[i] := 0;
      Stop(LPI8);
      Reset(LPI8);
   END;
END;

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

IF (ext_green[i]) THEN

   /* 1) If bus clears (check-out 84), terminate immediately */
   IF (Occupancy(84) > 0) THEN
      ext_green[i] := 0;
      rest_green[i] := 1;
   END;

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

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

   /* 4) OPTIONAL: keep normal gap-out when no bus is active */
   IF (ext_green[i] AND NOT bus_req[i]) THEN
      IF (Headway10(dk) > MINGAP10[i]) THEN
```

```
            ext_green[i] := 0;
            rest_green[i] := 1;
         END;
      END;

END.

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


/*************************************************************/
/* =============== MAIN PROGRAM ============================ */

/* ---- Initialization ---- */
IF sim_timer = 0 THEN
   START(sim_timer);
END;

IF sim_timer <= 5 THEN
   i := 1; GOSUB Initialize_early_green;
   i := 2; GOSUB Initialize_early_green;
   i := 3; GOSUB Initialize_early_green;
   i := 4; GOSUB Initialize_early_green;
   i := 5; GOSUB Initialize_early_green;
   i := 6; GOSUB Initialize_early_green;
   i := 7; GOSUB Initialize_early_green;
   i := 8; GOSUB Initialize_early_green;
END;

/*************************************************************/
/* ---- A. Update Calls ---- */
i := 1; dk := 11; GOSUB NONLocking_call;
i := 2; dk := 21; GOSUB NONLocking_call;
i := 3; dk := 31; GOSUB NONLocking_call;
i := 4; dk := 41; GOSUB Locking_call;
i := 5; dk := 51; GOSUB NONLocking_call;
i := 6; dk := 61; GOSUB NONLocking_call;
i := 7; dk := 71; GOSUB NONLocking_call;
i := 8; dk := 81; GOSUB Locking_call;
```

```
/*******************************************************/
/* ---- A1. SIMPLE BUS REQUEST (must be BEFORE transitions) ---- */

/* clear arrays each step (safe) */
bus_req[1] := 0; bus_req[2] := 0; bus_req[3] := 0; bus_req[4] := 0;
bus_req[5] := 0; bus_req[6] := 0; bus_req[7] := 0; bus_req[8] := 0;

bus_window_ok[1] := 0; bus_window_ok[2] := 0; bus_window_ok[3] := 0;
    bus_window_ok[4] := 0;
bus_window_ok[5] := 0; bus_window_ok[6] := 0; bus_window_ok[7] := 0;
    bus_window_ok[8] := 0;

/* Phase 8 (NB): check-in 83, check-out 84 */
bus_req[8] := (Occupancy(83) > 0) AND (Occupancy(84) = 0);

/* Phase 4 (SB): check-in 43, check-out 44 */
bus_req[4] := (Occupancy(43) > 0) AND (Occupancy(44) = 0);

/* Eligible only if bus present AND within last BUS_WINDOW seconds before
    MAXGREEN */
bus_window_ok[4] := (bus_req[4]) AND ((MAXGREEN[4] - T_green(4)) <=
    BUS_WINDOW);
bus_window_ok[8] := (bus_req[8]) AND ((MAXGREEN[8] - T_green(8)) <=
    BUS_WINDOW);

/*******************************************************/
/* ---- B. Conflict-Status Logic (realistic NEMA) ---- */
no_conflict[1] :=
     Current_state(2, red)*Current_state(3, red)*
     Current_state(4, red)*Current_state(7, red)*
     Current_state(8, red)*(Remaining_Intergreen(1)=0);

no_conflict[2] :=
     Current_state(1, red)*Current_state(3, red)*
     Current_state(4, red)*Current_state(7, red)*
     Current_state(8, red)*(Remaining_Intergreen(2)=0);

no_conflict[3] :=
     Current_state(1, red)*Current_state(2, red)*
     Current_state(4, red)*Current_state(5, red)*
     Current_state(6, red)*(Remaining_Intergreen(3)=0);

```

```
no_conflict[4] :=
    Current_state(1, red)*Current_state(2, red)*
    Current_state(3, red)*Current_state(5, red)*
    Current_state(6, red)*(Remaining_Intergreen(4)=0);

no_conflict[5] :=
    Current_state(6, red)*Current_state(7, red)*
    Current_state(8, red)*Current_state(3, red)*
    Current_state(4, red)*(Remaining_Intergreen(5)=0);

no_conflict[6] :=
    Current_state(5, red)*Current_state(7, red)*
    Current_state(8, red)*Current_state(3, red)*
    Current_state(4, red)*(Remaining_Intergreen(6)=0);

no_conflict[7] :=
    Current_state(5, red)*Current_state(6, red)*
    Current_state(8, red)*Current_state(1, red)*
    Current_state(2, red)*(Remaining_Intergreen(7)=0);

no_conflict[8] :=
    Current_state(5, red)*Current_state(6, red)*
    Current_state(7, red)*Current_state(1, red)*
    Current_state(2, red)*(Remaining_Intergreen(8)=0);

/*********************************************************/
/* ---- C. Conflicting Calls ---- */
confl_call[1] := has_call[2] OR has_call[3] OR has_call[4] OR has_call[7]
   OR has_call[8];
confl_call[2] := has_call[1] OR has_call[3] OR has_call[4] OR has_call[7]
   OR has_call[8];
confl_call[3] := has_call[1] OR has_call[2] OR has_call[4] OR has_call[5]
   OR has_call[6];
confl_call[4] := has_call[1] OR has_call[2] OR has_call[3] OR has_call[5]
   OR has_call[6];
confl_call[5] := has_call[3] OR has_call[4] OR has_call[6] OR has_call[7]
   OR has_call[8];
confl_call[6] := has_call[3] OR has_call[4] OR has_call[5] OR has_call[7]
   OR has_call[8];
confl_call[7] := has_call[1] OR has_call[2] OR has_call[5] OR has_call[6]
   OR has_call[8];
confl_call[8] := has_call[1] OR has_call[2] OR has_call[5] OR has_call[6]
   OR has_call[7];
```

```
/*********************************************************/
/* ---- D. Phase Transition Evaluation ---- */
i := 1; dk := 12; GOSUB Imminent_to_rest;
i := 2; dk := 22; GOSUB Imminent_to_rest;
i := 3; dk := 32; GOSUB Imminent_to_rest;
i := 4; dk := 42; GOSUB Imminent_to_rest_LPI4;
i := 5; dk := 52; GOSUB Imminent_to_rest;
i := 6; dk := 62; GOSUB Imminent_to_rest;
i := 7; dk := 72; GOSUB Imminent_to_rest;
i := 8; dk := 82; GOSUB Imminent_to_rest_LPI8;

/*********************************************************/
/* ---- OPTIONAL: Dummy signals to verify TSP visually ---- */
/* SG 9: any bus check-in active */
IF ((Occupancy(83) > 0) OR (Occupancy(43) > 0)) THEN
   Sg_green(9);
ELSE
   Sg_red(9);
END;

/* SG10: bus_req active (bus between check-in and check-out) */
IF (bus_req[4] OR bus_req[8]) THEN
   Sg_green(10);
ELSE
   Sg_red(10);
END;

/* SG11: TSP eligible window true */
IF (bus_window_ok[4] OR bus_window_ok[8]) THEN
   Sg_green(11);
ELSE
   Sg_red(11);
END;

/* SG12: check-out hit */
IF ((Occupancy(84) > 0) OR (Occupancy(44) > 0)) THEN
   Sg_green(12);
ELSE
   Sg_red(12);
END;

/*********************************************************/
```

```
/* ---- E. Phase Sequencing and Barrier Logic ---- */

/* within-ring transitions */
IF rest_green[1] THEN
   rest_green[1] := 0;
   Sg_red(1);
   imminent[2] := 1;
END;

IF rest_green[3] THEN
   rest_green[3] := 0;
   Sg_red(3);
   imminent[4] := 1;
END;

IF rest_green[5] THEN
   rest_green[5] := 0;
   Sg_red(5);
   imminent[6] := 1;
END;

IF rest_green[7] THEN
   rest_green[7] := 0;
   Sg_red(7);
   imminent[8] := 1;
END;

/* ---- Barrier 1: between 2 & 6 ---- */
IF (rest_green[2] AND rest_green[6]) THEN
   rest_green[2] := 0;
   rest_green[6] := 0;
   Sg_red(2);
   Sg_red(6);

   IF has_call[3] THEN
      imminent[3] := 1;
   ELSE
      imminent[4] := 1;
   END;

   IF has_call[7] THEN
      imminent[7] := 1;
   ELSE
```

```
      imminent[8] := 1;
   END;
END;

/* ---- Barrier 2: between 4 & 8 ---- */
IF (rest_green[4] AND rest_green[8]) THEN
   rest_green[4] := 0;
   rest_green[8] := 0;
   Sg_red(4);
   Sg_red(8);

   IF has_call[1] THEN
      imminent[1] := 1;
   ELSE
      imminent[2] := 1;
   END;

   IF has_call[5] THEN
      imminent[5] := 1;
   ELSE
      imminent[6] := 1;
   END;
END.

/*************************************************************/
/* END PROGRAM */
END.
```

B.3 Alternative 2: Green Extension with Phase Rotation

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

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

```
VAP_Frequency 10;

/*******************************************************/
/* CONSTANTS */
CONST
CALLDELAY  = 2,
BUS_WINDOW = 10,
BUS_EXT    = 10;

/*******************************************************/
/* ARRAYS */
ARRAY
MAXGREEN[8] = [12, 26, 8, 32, 10, 26, 8, 30],
MINGAP10[8] = [30, 30, 30, 30, 30, 30, 30, 30],

/* recall: if phase 4 & 8 are on recall, set RECALL[4]=1 and RECALL[8]=1
   */
RECALL[8]   = [0,0,0,1,0,0,0,1],

has_call[8],
no_conflict[8],
confl_call[8],
imminent[8],
early_green[8],
ext_green[8],
rest_green[8],

/* bus state */
bus_active[8],       /* latched: 1 from check-in to check-out */
bus_window_ok[8],    /* 1 if eligible to extend */

/* rotation flags (use arrays to avoid VAP     variable     errors) */
rotate34[1],         /* 1 => serve 4 then 3, barrier ends on 3 */
rotate78[1];         /* 1 => serve 8 then 7, barrier ends on 7 */


/*******************************************************/
/* SUBROUTINES */

/* ---------- Locking Call ---------- */
SUBROUTINE Locking_call;
IF Current_state(i, red) THEN
```

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

/* ---------- Non-Locking Call ---------- */
SUBROUTINE NONLocking_call;
IF Current_state(i, red) THEN
    IF (RECALL[i] OR (Occupancy(dk) > CALLDELAY)) THEN
        has_call[i] := 1;
    ELSE
        has_call[i] := 0;
    END;
END.

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

/* ---------- Imminent-to-Rest (normal phases) ---------- */
SUBROUTINE Imminent_to_rest;
IF (imminent[i] AND no_conflict[i]) THEN
    Sg_green(i);
    early_green[i] := 1;
    imminent[i] := 0;
    has_call[i] := 0;
END;

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

IF (ext_green[i]) THEN
    IF ((Headway10(dk) > MINGAP10[i]) OR (T_green(i) >= MAXGREEN[i])) THEN
        ext_green[i] := 0;
        rest_green[i] := 1;
    END;
END.

```

```
/* ---------- Phase 4 with TSP extension ---------- */
SUBROUTINE Imminent_to_rest_TSP4;
IF (imminent[i] AND no_conflict[i]) THEN
   Sg_green(i);
   early_green[i] := 1;
   imminent[i] := 0;
   has_call[i] := 0;
END;

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

IF (ext_green[i]) THEN

   /* end immediately when bus checks out */
   IF (Occupancy(44) > 0) THEN
      ext_green[i] := 0;
      rest_green[i] := 1;
   END;

   /* hard max always */
   IF (ext_green[i] AND (T_green(i) >= (MAXGREEN[i] + BUS_EXT))) THEN
      ext_green[i] := 0;
      rest_green[i] := 1;
   END;

   /* at MAXGREEN: end only if NOT eligible */
   IF (ext_green[i] AND (T_green(i) >= MAXGREEN[i])) THEN
      IF NOT bus_window_ok[i] THEN
         ext_green[i] := 0;
         rest_green[i] := 1;
      END;
   END;

   /* optional: allow gap-out only if not active bus */
   IF (ext_green[i] AND (bus_active[i] = 0)) THEN
      IF (Headway10(dk) > MINGAP10[i]) THEN
         ext_green[i] := 0;
         rest_green[i] := 1;
      END;
```

```
    END;

END.

/* ---------- Phase 8 with TSP extension ---------- */
SUBROUTINE Imminent_to_rest_TSP8;
IF (imminent[i] AND no_conflict[i]) THEN
    Sg_green(i);
    early_green[i] := 1;
    imminent[i] := 0;
    has_call[i] := 0;
END;

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

IF (ext_green[i]) THEN

    /* end immediately when bus checks out */
    IF (Occupancy(84) > 0) THEN
        ext_green[i] := 0;
        rest_green[i] := 1;
    END;

    /* hard max always */
    IF (ext_green[i] AND (T_green(i) >= (MAXGREEN[i] + BUS_EXT))) THEN
        ext_green[i] := 0;
        rest_green[i] := 1;
    END;

    /* at MAXGREEN: end only if NOT eligible */
    IF (ext_green[i] AND (T_green(i) >= MAXGREEN[i])) THEN
        IF NOT bus_window_ok[i] THEN
            ext_green[i] := 0;
            rest_green[i] := 1;
        END;
    END;

    /* optional: allow gap-out only if not active bus */
    IF (ext_green[i] AND (bus_active[i] = 0)) THEN
```

```
      IF (Headway10(dk) > MINGAP10[i]) THEN
         ext_green[i] := 0;
         rest_green[i] := 1;
      END;
   END;

END.


/*******************************************************/
/* =============== MAIN PROGRAM ============================ */

/* ---- Initialization ---- */
IF sim_timer = 0 THEN
   START(sim_timer);
END;

IF sim_timer <= 5 THEN
   i := 1; GOSUB Initialize_early_green;
   i := 2; GOSUB Initialize_early_green;
   i := 3; GOSUB Initialize_early_green;
   i := 4; GOSUB Initialize_early_green;
   i := 5; GOSUB Initialize_early_green;
   i := 6; GOSUB Initialize_early_green;
   i := 7; GOSUB Initialize_early_green;
   i := 8; GOSUB Initialize_early_green;

   /* safe init */
   rotate34[1] := 0;
   rotate78[1] := 0;
   bus_active[4] := 0;
   bus_active[8] := 0;
END;


/*******************************************************/
/* ---- A. Update Calls ---- */
i := 1; dk := 11; GOSUB NONLocking_call;
i := 2; dk := 21; GOSUB NONLocking_call;
i := 3; dk := 31; GOSUB NONLocking_call;
i := 4; dk := 41; GOSUB Locking_call;
i := 5; dk := 51; GOSUB NONLocking_call;
i := 6; dk := 61; GOSUB NONLocking_call;
```

```
i := 7; dk := 71; GOSUB NONLocking_call;
i := 8; dk := 81; GOSUB Locking_call;


/*********************************************************/
/* ---- A1. BUS LATCH + WINDOW ---- */

/* latch bus presence (SB: 43 in, 44 out) */
IF (Occupancy(43) > 0) THEN
   bus_active[4] := 1;
END;
IF (Occupancy(44) > 0) THEN
   bus_active[4] := 0;
END;

/* latch bus presence (NB: 83 in, 84 out) */
IF (Occupancy(83) > 0) THEN
   bus_active[8] := 1;
END;
IF (Occupancy(84) > 0) THEN
   bus_active[8] := 0;
END;

/* window eligibility only matters for phases 4 & 8 */
bus_window_ok[1] := 0; bus_window_ok[2] := 0; bus_window_ok[3] := 0;
bus_window_ok[5] := 0; bus_window_ok[6] := 0; bus_window_ok[7] := 0;

bus_window_ok[4] := (bus_active[4] = 1) * ((MAXGREEN[4] - T_green(4)) <=
   BUS_WINDOW);
bus_window_ok[8] := (bus_active[8] = 1) * ((MAXGREEN[8] - T_green(8)) <=
   BUS_WINDOW);


/*********************************************************/
/* ---- B. Conflict-Status Logic (same as your base) ---- */
no_conflict[1] :=
    Current_state(2, red)*Current_state(3, red)*
    Current_state(4, red)*Current_state(7, red)*
    Current_state(8, red)*(Remaining_Intergreen(1)=0);

no_conflict[2] :=
    Current_state(1, red)*Current_state(3, red)*
    Current_state(4, red)*Current_state(7, red)*
```

```
    Current_state(8, red)*(Remaining_Intergreen(2)=0);

no_conflict[3] :=
    Current_state(1, red)*Current_state(2, red)*
    Current_state(4, red)*Current_state(5, red)*
    Current_state(6, red)*(Remaining_Intergreen(3)=0);

no_conflict[4] :=
    Current_state(1, red)*Current_state(2, red)*
    Current_state(3, red)*Current_state(5, red)*
    Current_state(6, red)*(Remaining_Intergreen(4)=0);

no_conflict[5] :=
    Current_state(6, red)*Current_state(7, red)*
    Current_state(8, red)*Current_state(3, red)*
    Current_state(4, red)*(Remaining_Intergreen(5)=0);

no_conflict[6] :=
    Current_state(5, red)*Current_state(7, red)*
    Current_state(8, red)*Current_state(3, red)*
    Current_state(4, red)*(Remaining_Intergreen(6)=0);

no_conflict[7] :=
    Current_state(5, red)*Current_state(6, red)*
    Current_state(8, red)*Current_state(1, red)*
    Current_state(2, red)*(Remaining_Intergreen(7)=0);

no_conflict[8] :=
    Current_state(5, red)*Current_state(6, red)*
    Current_state(7, red)*Current_state(1, red)*
    Current_state(2, red)*(Remaining_Intergreen(8)=0);


/**************************************************************/
/* ---- C. Conflicting Calls ---- */
confl_call[1] := has_call[2] OR has_call[3] OR has_call[4] OR has_call[7]
    OR has_call[8];
confl_call[2] := has_call[1] OR has_call[3] OR has_call[4] OR has_call[7]
    OR has_call[8];
confl_call[3] := has_call[1] OR has_call[2] OR has_call[4] OR has_call[5]
    OR has_call[6];
confl_call[4] := has_call[1] OR has_call[2] OR has_call[3] OR has_call[5]
    OR has_call[6];
```

```
confl_call[5] := has_call[3] OR has_call[4] OR has_call[6] OR has_call[7]
    OR has_call[8];
confl_call[6] := has_call[3] OR has_call[4] OR has_call[5] OR has_call[7]
    OR has_call[8];
confl_call[7] := has_call[1] OR has_call[2] OR has_call[5] OR has_call[6]
    OR has_call[8];
confl_call[8] := has_call[1] OR has_call[2] OR has_call[5] OR has_call[6]
    OR has_call[7];


/***********************************************************/
/* ---- D. Phase Transition Evaluation ---- */
i := 1; dk := 12; GOSUB Imminent_to_rest;
i := 2; dk := 22; GOSUB Imminent_to_rest;
i := 3; dk := 32; GOSUB Imminent_to_rest;

i := 4; dk := 42; GOSUB Imminent_to_rest_TSP4;

i := 5; dk := 52; GOSUB Imminent_to_rest;
i := 6; dk := 62; GOSUB Imminent_to_rest;
i := 7; dk := 72; GOSUB Imminent_to_rest;

i := 8; dk := 82; GOSUB Imminent_to_rest_TSP8;

/***********************************************************/
/* ---- OPTIONAL: Dummy signals for TSP visualization ---- */
/***********************************************************/

/* SG 9: Bus detected at upstream check-in */
IF (Occupancy(43) > 0) THEN
   Sg_green(9);
ELSE
   IF (Occupancy(83) > 0) THEN
      Sg_green(9);
   ELSE
      Sg_red(9);
   END;
END;


/* SG 10: Bus active (latched between check-in and check-out) */
IF (bus_active[4] = 1) THEN
   Sg_green(10);
```

```
ELSE
   IF (bus_active[8] = 1) THEN
      Sg_green(10);
   ELSE
      Sg_red(10);
   END;
END;


/* SG 11: Bus eligible for green extension (window OK) */
IF (bus_window_ok[4] = 1) THEN
   Sg_green(11);
ELSE
   IF (bus_window_ok[8] = 1) THEN
      Sg_green(11);
   ELSE
      Sg_red(11);
   END;
END;


/* SG 12: Bus cleared intersection (check-out hit) */
IF (Occupancy(44) > 0) THEN
   Sg_green(12);
ELSE
   IF (Occupancy(84) > 0) THEN
      Sg_green(12);
   ELSE
      Sg_red(12);
   END;
END;






/**********************************************************/
/* ---- E. Phase Sequencing and Barrier Logic ---- */

/* 1 -> 2 */
IF rest_green[1] THEN
   rest_green[1] := 0;
```

```
      Sg_red(1);
      imminent[2] := 1;
END;

/* 5 -> 6 */
IF rest_green[5] THEN
      rest_green[5] := 0;
      Sg_red(5);
      imminent[6] := 1;
END;

/* default lead-left transitions */
IF rest_green[3] THEN
      IF (rotate34[1] = 0) THEN
            rest_green[3] := 0;
            Sg_red(3);
            imminent[4] := 1;
      END;
END;

IF rest_green[7] THEN
      IF (rotate78[1] = 0) THEN
            rest_green[7] := 0;
            Sg_red(7);
            imminent[8] := 1;
      END;
END;

/* rotated lag-left transitions */
IF rest_green[4] THEN
      IF (rotate34[1] = 1) THEN
            rest_green[4] := 0;
            Sg_red(4);
            imminent[3] := 1;
      END;
END;

IF rest_green[8] THEN
      IF (rotate78[1] = 1) THEN
            rest_green[8] := 0;
            Sg_red(8);
            imminent[7] := 1;
      END;
```

```
END;


/* ---- Barrier 1: between 2 & 6 ---- */
IF (rest_green[2] AND rest_green[6]) THEN
   rest_green[2] := 0;
   rest_green[6] := 0;
   Sg_red(2);
   Sg_red(6);

   /* decide rotation ONCE at barrier release */
   rotate34[1] := (bus_active[4] = 1) * (has_call[3] = 1);
   rotate78[1] := (bus_active[8] = 1) * (has_call[7] = 1);

   /* ring1 start */
   IF (rotate34[1] = 1) THEN
      imminent[4] := 1;
   ELSE
      IF has_call[3] THEN
         imminent[3] := 1;
      ELSE
         imminent[4] := 1;
      END;
   END;

   /* ring2 start */
   IF (rotate78[1] = 1) THEN
      imminent[8] := 1;
   ELSE
      IF has_call[7] THEN
         imminent[7] := 1;
      ELSE
         imminent[8] := 1;
      END;
   END;

END;


/* ---- Barrier 2 (dynamic end) ---- */
IF ( ((rotate34[1] = 1) AND rest_green[3]) OR ((rotate34[1] = 0) AND
   rest_green[4]) ) THEN
```

```
    IF ( ((rotate78[1] = 1) AND rest_green[7]) OR ((rotate78[1] = 0) AND
        rest_green[8]) ) THEN

        /* clear barrier-ending phases */
        IF (rotate34[1] = 1) THEN
            rest_green[3] := 0;
            Sg_red(3);
        ELSE
            rest_green[4] := 0;
            Sg_red(4);
        END;

        IF (rotate78[1] = 1) THEN
            rest_green[7] := 0;
            Sg_red(7);
        ELSE
            rest_green[8] := 0;
            Sg_red(8);
        END;

        /* reset rotation */
        rotate34[1] := 0;
        rotate78[1] := 0;

        /* next starts (same as base) */
        IF has_call[1] THEN
            imminent[1] := 1;
        ELSE
            imminent[2] := 1;
        END;

        IF has_call[5] THEN
            imminent[5] := 1;
        ELSE
            imminent[6] := 1;
        END;

    END;
END.

/************************************************************/
END.
```