Лабораторная работа N2

Реализация конечного автомата (автомобильный светофор) на контроллере двумя методами

Цель работы: разработать и отладить конечный автомат (КДА) на платформе Arduino, моделирующий управление светофором автомобильного перекрёстка. Реализовать функционал двумя способами, определить какой метод позволяет получить наименьший код:

1. Классический — с использованием switch-case по текущему состоянию.

2. Табличный (table-driven) — с использованием двумерной таблицы (состояние × событие), где каждая ячейка содержит обработчик для комбинации состояние/событие.

Список оборудования

- Плата макетная с контроллером Arduino Uno
- Модуль светофора (Green, Yellow, Red)
  (или два модуля — опционально)
- Кнопка (пешеходный запрос или переключение режима)
- Провода, соединительный кабель USB

Рекомендованное подключение к контроллеру (одно направление):

- pin 13 — Red LED
- pin 12 — Yellow LED
- pin 11 — Green LED
- pin 2 — Button (INPUT_PULLUP)

Функциональные требования (поведение)

1. Базовая последовательность циклов: Green → Yellow → Red → Green.
2. Время по умолчанию:
    - Green = 10 с

- Yellow = 3 с
- Red = 10 с

3. Поддержка события PEDESTRIAN_REQUEST (кнопка): при срабатывании — переход к безопасному завершению текущего зелёного цикла (через Yellow → Red), затем красный дополнительный интервал для пешеходов (например, +5 с), после чего обычный цикл возобновляется.
4. Поддержка события EMERGENCY (внешний вход/симуляция) — немедленный переход в режим мигающего желтого (WARNING), пока событие активно.
5. Ночной режим (опция): если активирован (например, длительное нажатие кнопки) — мигание желтого с периодом 1 с.
6. Логирование в Serial для отладки — вывод состояния и событий.
7. Требование по надежности кнопки — антидребезг (debounce).

Состояния и события

Состояния (пример, возможно реализация более сложного варианта, например плюс зеленый мигающий и красно-желтый):

- S_GREEN
- S_YELLOW
- S_RED
- S_WARNING (мигающий желтый)
- S_OFF (опционально)

События:

- E_TIMER_EXPIRE — таймер состояния завершился
- E_PED_REQ — нажатие кнопки (пешеход)
- E_EMERGENCY_ON — авария включена
- E_EMERGENCY_OFF — авария выключена
- E_NIGHT_MODE_TOGGLE — переключение ночного режима

Графы процесса управления приведены на рис.1:

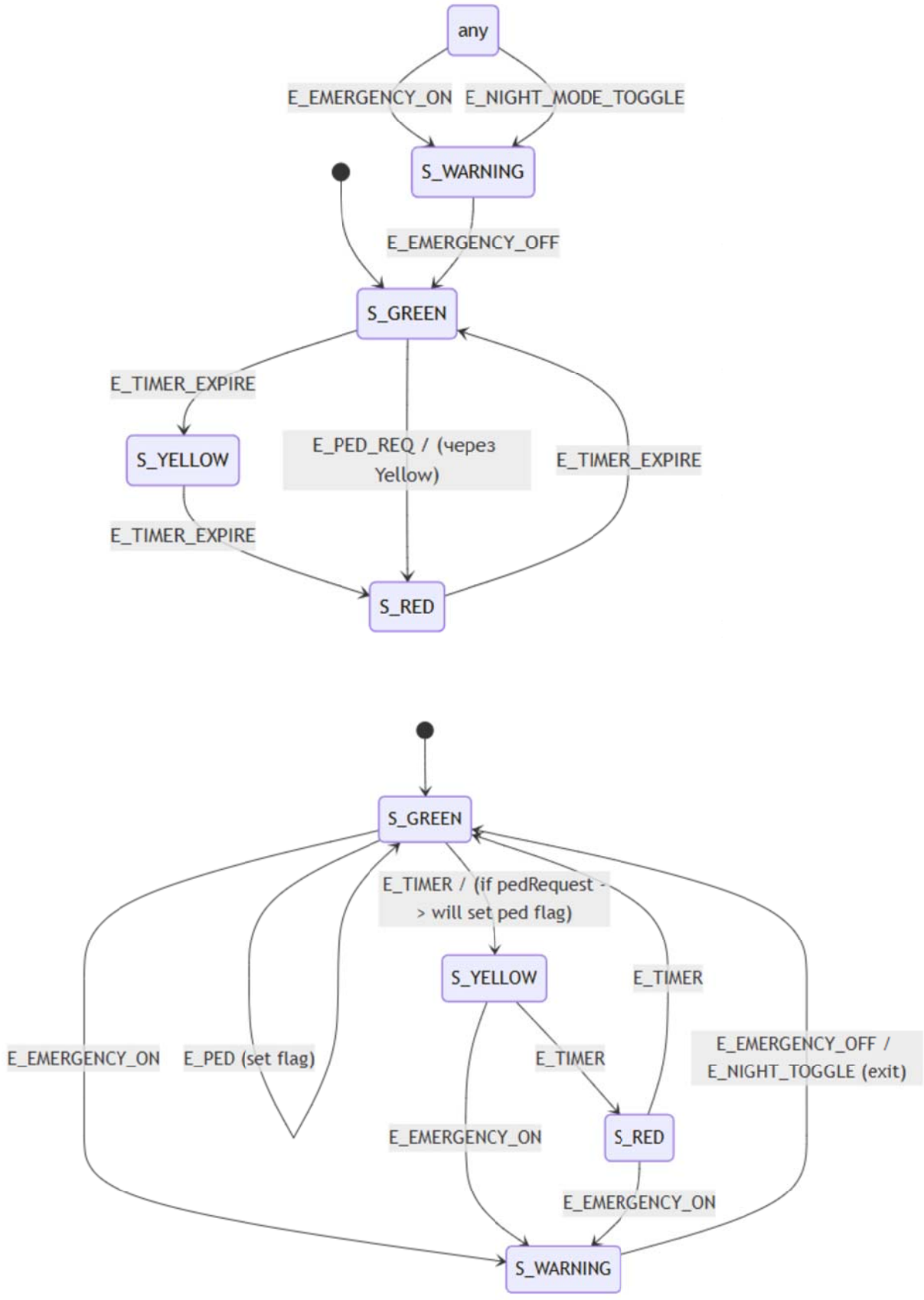

Рис.1 Варианты графов управления

Обзор метода 1 — классический (switch-case)

Описание: основной цикл опрашивает события и хранит одно переменное currentState. Для перехода между состояниями используется конструкция switch(currentState); в каждом состоянии — обработка входов, установка выходов (LED), запуск таймера и переход по истечении времени.

Основные элементы:

- enum State { S_GREEN, S_YELLOW, S_RED, S_WARNING };
- unsigned long stateStartMillis; unsigned long stateDuration;
- функция handleEventsInState(State s) — читает кнопки/сенсоры, устанавливает флаги событий
- Внутри loop() — проверяем флаги, таймер, и делаем switch(currentState).

Скелет кода (Arduino C++ — сокращённый):

```
enum State { S_GREEN, S_YELLOW, S_RED, S_WARNING };
State currentState = S_GREEN;
unsigned long stateStart = 0;
unsigned long duration = 0;
bool pedRequest = false;
bool emergency = false;

void goToState(State s, unsigned long dur) {
  currentState = s;
  stateStart = millis();
  duration = dur;
  setOutputsForState(s);
}

void setup() {
  // pinMode, Serial.begin, init
  goToState(S_GREEN, 10000);
}

void loop() {
  readInputs(); // обновляет pedRequest, emergency и т.д.

  if (emergency) {
    if (currentState != S_WARNING) goToState(S_WARNING, 500); // мигающий
  }

  switch (currentState) {
```

```
    case S_GREEN:
      if (millis() - stateStart >= duration) {
        if (pedRequest) {
          // сначала короткий переход: Yellow -> Red с удлиненным Red
          goToState(S_YELLOW, 3000);
        } else goToState(S_YELLOW, 3000);
      }
      break;

    case S_YELLOW:
      if (millis() - stateStart >= duration) {
        if (pedRequest) {
          pedRequest = false;
          goToState(S_RED, 15000); // удлинённый red для пешехода
        } else goToState(S_RED, 10000);
      }
      break;

    case S_RED:
      if (millis() - stateStart >= duration) {
        goToState(S_GREEN, 10000);
      }
      break;

    case S_WARNING:
      // мигание: переключение желтого в on/off каждые duration
      // возврат при выключении emergency — обработка в readInputs()
      break;
  }

  // доп. поведение, логирование
}
```

Подсказки:

- Не блокируйте loop() на длительное время — используйте millis() вместо delay().
- Реализуйте антидребезг для кнопки (например, 50 ms стабильности).
- Для мигания желтого можно переключать LED каждые 500 ms (внутри S_WARNING).

Обзор метода 2 — таблица переходов (table-driven FSM)

Описание: определите таблицу, индексируемую текущим состоянием и типом события. Таблица хранит обработчики (указатели на функции) — один обработчик выполняется при совпадении состояния и события. События формируются/генерируются вне таблицы (таймер истёк, кнопка, авария). Табличный подход удобен для расширяемости и тестирования.

Структура:

- enum State { ... }
- enum Event { E_NONE, E_TIMER, E_PED, E_EMERGENCY_ON, E_EMERGENCY_OFF, E_NIGHT_TOGGLE }
- typedef void (*Handler)(void);
- Handler fsmTable[NUM_STATES][NUM_EVENTS];

Принцип:

1. В loop() генерируем событие (или список событий).
2. Для каждого события вызываем handler = fsmTable[currentState][event]; если handler != NULL — вызываем.
3. Handler сам может менять currentState и устанавливать выходы и длительности.

Скелет кода:

```
enum State { S_GREEN, S_YELLOW, S_RED, S_WARNING, NUM_STATES };
enum Event { E_NONE, E_TIMER, E_PED, E_EMERGENCY_ON,
E_EMERGENCY_OFF, NUM_EVENTS };

State currentState = S_GREEN;
unsigned long stateStart = 0;
unsigned long stateDur = 0;

typedef void (*Handler)(void);
Handler fsmTable[NUM_STATES][NUM_EVENTS];

void onGreenTimer() {
  // если есть pedRequest -> перейти в S_YELLOW, иначе обычный переход
  if (pedRequest) { currentState = S_YELLOW; stateDur = 3000; }
  else { currentState = S_YELLOW; stateDur = 3000; }
  stateStart = millis();
  setOutputsForState(currentState);
}
```

```
void onAnyEmergencyOn() {
  currentState = S_WARNING;
  stateDur = 500;
  stateStart = millis();
  setOutputsForState(currentState);
}

void setupTable() {
  for (int s=0; s<NUM_STATES; ++s)
    for (int e=0; e<NUM_EVENTS; ++e) fsmTable[s][e] = NULL;

  fsmTable[S_GREEN][E_TIMER] = onGreenTimer;
  fsmTable[S_GREEN][E_PED]   = onGreenPedButton; // можно отдельный обработчик
  // ... заполнить остальные ячейки
  // обработчики E_EMERGENCY_ON для всех состояний:
  for (int s=0; s<NUM_STATES; ++s) fsmTable[s][E_EMERGENCY_ON] = onAnyEmergencyOn;
}

void loop() {
  Event ev = pollEvent(); // получить одиночное событие или очередь
  if (ev != E_NONE) {
    Handler h = fsmTable[currentState][ev];
    if (h) h();
  }

  // проверка таймера -> если истёк -> ev = E_TIMER, и т.д.
}
```

Подсказки:

- Таблица может содержать NULL — значит событие игнорируется в данном состоянии.
- Можно хранить в таблице структуры с полями { handler, nextStateIfAny, mandatoryDuration } для более декларативного подхода.
- Для множественных событий используйте очередь событий (ring buffer).

Вспомогательные требования к программе и пункты для реализации

- Обязательное использование неблокирующей логики (millis(), без delay() > 50 ms).
- Реализация debounce для кнопки (например, 50 ms).
- Логирование через Serial: при каждом переходе состояния выводить название состояния, timestamp, причина перехода.
- Документирование кода: комментарии, краткое README.
- Обработка нештатных состояний: при ошибке — переход в безопасный режим (мигающий жёлтый).

Тестовые сценарии и критерии проверки перед сдачей задания

Обязательные тесты:

1. Базовый цикл без нажатий: 10s Green → 3s Yellow → 10s Red → и т.д.
2. Нажатие кнопки во время Green: после завершения Green через Yellow переходим в Red с дополнительным временем для пешехода.
3. Аварийный режим: подать EMERGENCY_ON (замыкание входа) — немедленный переход в WARNING; снять — возврат в предыдущий нормальный цикл.
4. Ночной режим: переключить — мигание жёлтого; повторное переключение — возврат.
5. Надёжность кнопки: быстрое многократное нажатие — только одно событие после дебаунса.

Время выполнения: 4 часа.

Вопросы к зачету по лабораторной работе

1. Какова основная разница между классическим FSM (switch-case) и table-driven FSM?
2. Почему в реальном времени следует избегать длительных delay() в loop()?
3. Объясните принцип работы debounce и почему он нужен для кнопки.
4. Как реализован пешеходный запрос (pedestrian request) в обоих вариантах?

5. Почему мы используем millis() вместо таймеров delay()? Назовите преимущества.

6. Что происходит, если при активном EMERGENCY входе поступает пешеходный запрос?

7. Как реализован ночной режим и чем он отличается от аварийного?

8. Как должна вести себя система при заполненной очереди событий (в таблице)?

9. Какие состояния у вас есть — и какие события их переводят в S_WARNING?

10. Какая точность времени ожидается и от чего она зависит?

11. Покажите, где и как вы предотвращаете гонки (race conditions) при доступе к флагам (pedRequest, emergency).

12. Объясните, как восстановление после EMERGENCY реализовано — вернётся ли система в то же место цикла? Какие компромиссы есть?

13. Как улучшить таблицу так, чтобы handlers были декларативными (handler + nextState + duration) и почему это полезно?

14. Как расширить реализацию до двух направлений с контролем конфликтов (mutual exclusion)? Опишите логику.

15. Если очередь событий переполняется, какие стратегии обработки событий вы предложите?

16. Предложите тесты для проверки корректности обработки long-press vs short-press.

17. Какие метрики/логирование вы бы добавили для production-мониторинга светофора?

18. Как вы бы реализовали адаптивное время зелёного на основе потока машин (датчик/ультразвук)?

19. Какие проблемы может дать использование Serial.print в каждой итерации loop() и как уменьшить нагрузку?

20. Как бы вы добавили unit-тесты для FSM (симуляция времени и событий) на хост-машине?

21. Светофор застрял в одном состоянии — какие шаги для поиска причины?

22. Нажатие кнопки не регистрируется — как локализовать проблему (аппаратура/софт)?

23. Мигающий жёлтый неравномерный (дрожит) — какие возможные причины?

24. При включении EMERGENCY система не возвращается в норму — что проверить первым делом?

25. Как измерить реальные интервалы времени и сравнить с ожидаемыми (инструменты/метод)?