ОГЛАВЛЕНИЕ

**Элементы оглавления не найдены.**

# 1 ПОДХОДЫ К ЭМУЛЯЦИИ МНОГОЗАДАЧНОСТИ В ARDUINO

## ФУНКЦИЯ YIELD()

Если объявить функцию yield(), то расположенный внутри неё код будет выполняться во время работы любой задержки delay() в программе. Это позволяет быстро реализовать пару параллельно выполняющихся задач. Рассмотрим простой пример: стандартный мигающий светодиод, но с опросом кнопки (листинг 1.1) [1].



Листинг 1.1 – Пример многозадачности с использованием функции yield()

Такой подход позволяет достаточно просто и быстро реализовать многозадачность, однако в данном случае она будет ограничена только двумя одновременно выполняющимися задачами.

## МНОГОЗАДАЧНОСТЬ С MILLIS()

При помощи функций времени millis() или micros() можно организовать программный таймер, по которому и выполнять нужные действия. Схема такая:

1. Заводим переменную для таймера типа unsigned long (uint32\_t) – именно этот тип возвращает millis().
2. Ищем разницу между текущим временем работы программы и переменной таймера.
3. Если разница больше необходимого периода – выполняем нужный код и сбрасываем таймер.

Например, если мы хотим выполнять одно действие два раза в секунду, второе – три, и третье – 10. Нам понадобится 3 переменные таймера и 3 конструкции с условием (листинг 1.2) [2].



Листинг 1.2 – Пример многозадачности с использованием функции millis()

Такой подход также достаточно прост и быстр в реализации, однако плох тем, что мы задаём достаточно грубо задаём временные рамки выполнения задач, что в некоторых случаях может привести к простоям процессора.

## ОПЕРАЦИОННАЯ СИСТЕМА

Операционные системы (ОС), которые работают внутри встраиваемой электроники (embedded devices), в большинстве случаев являются операционными системами реального времени (RTOS – Real-Time Operating System). В этих системах критически важно, чтобы задачи исполнялись в строго отведенное для них время. Задачи реального времени в этих системах подразумевают под собой то, что время ответа на определенное событие всегда занимает строго фиксированное время. Операционные системы реального времени (RTOS) спроектированы таким образом, чтобы обеспечивать запуск приложений с максимально точным таймингом и гарантировать им высокую надежность работы. Также ОС реального времени помогают реализовать многозадачность в микроконтроллерах (микропроцессорах) с одним ядром [3].

Данный подход позволяет сделать многозадачность максимально эффективной, и в сложных случаях без него не обойтись. Однако использование операционной системы требует большого количества ресурсов, которые в случае применения микроконтроллеров достаточно ограничены. Кроме этого, создание подобной операционной потребует от программиста достаточной осведомлённости о архитектуре выбранного микроконтроллера и высоких навыков в низкоуровневом программировании, а использование готовых решений может потребовать долгого разбирательства в чужом коде.

## ПРЕРЫВАНИЯ ПО АППАРАТНОМУ ТАЙМЕРУ

Как в реальной жизни, в микроконтроллерах таймер — это нечто, что вы устанавливаете для запуска предупреждения в определенный момент в будущем. Когда этот момент наступает, предупреждение прерывает микропроцессор, напоминая ему что-то сделать, например, выполнить определенный кусок кода.

Таймеры работают, увеличивая переменную-счетчик, известную как регистр счетчика. Регистр счетчика может считать до определенного значения в зависимости от его размера (обычно 8 или 16 бит). Таймер увеличивает этот счетчик шаг за шагом до достижения его максимального значения, после чего счетчик переполняется и сбрасывается обратно на ноль. Таймер обычно устанавливает флаг, чтобы предупредить вас о том, что произошло переполнение. Этот флаг можно проверить вручную, или вы можете настроить таймер на вызов прерывания сразу после установки флага. Как и в случае с любым другим прерыванием, вы можете указать обработчик прерывания (ISR), чтобы выполнить свой собственный код, когда происходит переполнение таймера. ISR автоматически сбросит флаг переполнения, поэтому использование прерываний обычно является лучшим вариантом для простоты и скорости.

Для увеличения значения счетчика через регулярные интервалы таймеру необходим источник тактового сигнала. Этот источник предоставляет постоянный сигнал. Каждый раз, когда таймер обнаруживает этот сигнал, он увеличивает свой счетчик на единицу.

Вы можете подавать внешний тактовый сигнал для использования с таймерами, но обычно используется внутренний тактовый сигнал микросхемы [4].

Кодом, который будет выступать в качестве таймерного прерывания, может стать простая функция переключения задач или некоторый более сложный планировщик задач.

Положительной стороной такого подхода является то, что основной код программы не будет нагружен лишними конструкциями, так как код таймерного прерывания существует и исполняется отдельно. По этой же причине работа данного метода будет более чёткой и стабильной, чем в случае использования программного таймера, описанного в пункте 1.2.

Отрицательной стороной является то, что нужно хранить большую таблицу временных интервалов вызова прерываний. Можно, конечно, строить таблицу динамически: каждый раз, когда приходит срок очередного события, программа обработки прерывания спрашивает у обслуживаемого устройства, каким будет следующее событие и когда оно должно произойти. В этом случае достаточно будет таблицы с одной строчкой на каждый канал.

Однако при этом подходе процедура обслуживания события явно устанавливает величину интервала ожидания следующего события по тому же каналу. Это значит, что мы не можем запустить отсчет следующего интервала до завершения текущей процедуры обслуживания события, поскольку еще не знаем, какой из интервалов минимален. Из этого следует, что очередной интервал удлиняется на время выполнения процедуры, а это снижает точность отсчета времени, особенно в случаях, когда обслуживание события длится долго.

Также может возникнуть ситуация, когда до завершения обработки прерывания возникает еще одно прерывание, если оно не запрещено аппаратно. Поскольку у процедур обработки прерывания обычно проблемы с реентерабельностью, результат такой вложенности непредсказуем. Можно запретить повторные прерывания на время их обработки, но в этом случае есть риск их потерять, если они следуют достаточно часто [5].

## МНОГОЗАДАЧНОСТЬ НА КОНЕЧНЫХ АВТОМАТАХ

Детерминированный конечный автомат состоит из следующих компонентов.

1. Конечное множество состояний, обозначаемое обычно как .

2. Конечное множество входных символов, обозначаемое обычно как .

3. Функция переходов, аргументами которой являются текущее состояние и входной символ, а значением — новое состояние. Функция переходов обычно обозначается как .

4. Начальное состояние, одно из состояний в .

5. Множество заключительных, или допускающих, состояний . Множество является подмножеством .

То есть ДКА – это пятёрка где — имя ДКА, — множество состояний, — множество входных символов, — функция переходов, — начальное состояние и — множество допускающих состояний [6 с. 62].

Любую задачу, которую будет решать встроенное приложение можно описать языком теории конечных автоматов. После этого можно будет создать некоторый цикл, который будет периодически заставлять полученные автоматы переходить в новое состояние. Также нужно будет создать систему сообщений, которая будет передавать последовательность выходных символов одного автомата на вход другому, позволяя задачам работать синхронно.

Главным достоинством автоматного подхода к созданию встроенного приложения является то, что подобную программу легко модифицировать и документировать. Ещё одним преимуществом является повторное использование кода. Фактически, программа в данном случае состоит из компонентов, являющихся в высокой степени автономными «сущностями». Компонент имеет ограниченное количество связей с остальной программой, его можно разрабатывать и тестировать отдельно, а применять многократно, и именно это свойство подобных систем делает разработку быстрой и удобной [7].

Негативной стороной является то, что если один из автоматов медленно функционирует, или вовсе впадает в бесконечный цикл или бесконечное ожидание, то это отрицательно сказывается на производительности всего приложения. Однако, эта проблема имеет решение, которое будет рассмотрено далее.

Таким образом, автоматный подход к созданию встроенных приложений является наиболее оптимальным для большинства задач, так как имеет неоспоримые плюсы для программиста, и его минусы достаточно легко исправимы. Поэтому теоретическому описанию данного подхода и будет посвящена следующая глава.

# ТЕОРЕТИЧЕСКОЕ ОПИСАНИЕ МНОГОЗАДАЧНОГО ВСТРОЕННОГО ПРИЛОЖЕНИЯ НА ДКА

## 2.1 ОБЩАЯ СТРУКТУРА

Каждый конечный автомат описан своим программным модулем и имеет как минимум две внешние функции – функция инициализации и функция, отвечающая за работу автомата.

Главная особенность подхода в том, что функция работы не должна выполнять продолжительных во времени действий, связанных с ожиданием какого-либо флага или с истечением временного интервала.

Задачей главного цикла программы является поочередный вызов функций работы всех автоматов, составляющих программу.

Многозадачность системы обеспечивается тем, что в каждой итерации главного цикла поочередно вызываются функции работы каждого автомата – каждому автомату выделяется время для выполнения какого-либо элементарного действия (или, что тоже может быть, просто для передачи управления далее по списку). То есть если какой-либо из автоматов «зависает», то «зависает» вся система. Именно для предотвращения подобной ситуации вводится явный запрет на действия в автоматах, которые занимают продолжительное или неопределенное время [7].

## ВИДЫ КОНЕЧНЫХ АВТОМАТОВ

Автомат Мура был впервые описан Эдвардом Ф. Муром в [8].

Автомат Мура может быть определён как кортеж из 6 элементов, включающий:

* множество внутренних состояний (внутренний алфавит);
* начальное состояние ;
* множество входных сигналов (входной алфавит);
* множество выходных сигналов (выходной алфавит);
* функция переходов ;
* функция вывода .

Автомат Мили был впервые описан Джорджем Мили в [9].

Автомат Мили – совокупность , где

* — конечное непустое множество состояний автомата;
* — конечное непустое множество входных символов;
* — конечное непустое множество выходных символов;
* — функция переходов, отображающая пары состояние/входной символ на соответствующее следующее состояние;
* — функция выходов, отображающая пары состояние/входной символ на соответствующий выходной символ;
* — начальное состояние.

Основное отличие автомата Мили состоит в том, что в нём выходная последовательность зависит от состояния автомата и входных сигналов, тогда как в автомате Мура она зависит лишь от текущего состояния автомата.

Другими словами, в автомате Мура выходная последовательность определяется самими состояниями, а в автомате Мили — переходами между ними.

Также существуют смешанные автоматы, в которых выходные состояния могут определяться как самими состояниями, так и переходами между ними.

## АВТОМАТ КАК ЗАДАЧА В ВСТРОЕННОМ ПРИЛОЖЕНИИ

В данном случае под входом понимается сообщение, срабатывание таймера, результат выражения, которое имеет логическое значение (в том числе возвращаемое функцией логическое значение). Доступ к аппаратным ресурсам микроконтроллера (регистрам периферийных устройств, портам ввода–вывода и т. п.) выполняется также путем вызова функций.

Под выходом автомата понимается отправка сообщения; запуск, останов или сброс таймера; вызов функции.

При этом выделяется особая переменная состояния – переменная, которая определяет текущее состояние автомата. Эта переменная должна быть доступна только своему автомату, ее изменение из других автоматов недопустимо.

Автомат может осуществлять действия (action) и деятельности (activity) автомата. И деятельность, и действие автомата относятся к его выходной активности.

При этом действием называется выходное воздействие, выполняемое однократно при входе в состояние или на переходе, а деятельностью – выходное воздействие, выполняющееся непрерывно в течение всего времени нахождения автомата в определенном состоянии. Также возможна реализация действий, выполняющихся на выходе из состояния.

Автомат Мура и автомат смешанного типа могут выполнять как действия, так и деятельности, а автомат Мили может выполнять только действия.

Автомат в минимальном виде представлен на Рисунке 2.1.

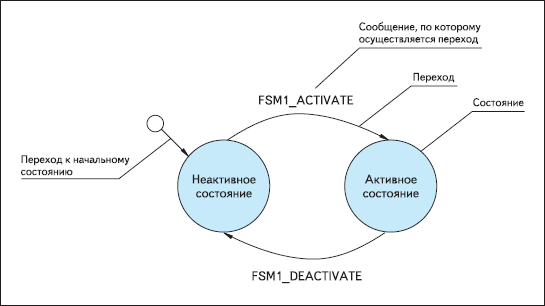


Рисунок 2.1 — Автомат в минимальном виде.

Данный автомат выполняет действия на переходах из одного состояния в другое (именно поэтому это автомат Мили), однако его достаточно легко преобразовать в автомат Мура или в смешанный автомат. Представленный на рисунке автомат не делает ничего полезного, это просто шаблон, на основе которого можно строить автоматы с более сложным поведением [7].

## 2.4 МЕНЕДЖЕР И ГЕНЕРАТОР СООБЩЕНИЙ

Для отправки сообщения другому автомату действующий автомат должен решить следующие вопросы: содержание, канал передачи, получатель, протокол передачи.

Во встроенном приложении мы не можем позволить доверить каждому автомату решать данные вопросы самостоятельно, так как в общем случае время их решения не является константным, а значит операции, связанные с этим вопросом, могут сильно замедлить работу приложения.

Следовательно, формирование и пересылка сообщений должна быть централизована. Для этого предлагается ввести в рассматриваемую модель встроенного приложения новые сущности: генератор и менеджер сообщений.

Генератор сообщений будет работать по следующему алгоритму. Он принимает на вход символ из множества выходных последовательностей одного из автоматов. С помощью этого символа по некоторому заданному условиями прикладной задачи набору правил он определяет содержание и получателя сообщения. Далее он формирует сообщение по заданному программистом протоколу и помещает его в очередь сообщений.

Менеджер сообщений занимается распределением сообщений в очереди по получателям. Алгоритм работы менеджера сообщений следующий. Во время инициализации работы приложения автоматы, которые будут принимать сообщения, должны быть зарегистрированы в менеджере сообщений. Сообщения, созданные генератором, помещаются в очередь сообщений. Менеджер выбирает по некому алгоритму, какое сообщение будет послано следующим. Далее он определяет получателя и отправляет ему сообщение.

Для единообразия рассматриваемой модели встроенного приложения генератор и менеджер сообщений можно описать как ДКА.

## 2.5 ГЕНЕРАТОР СООБЩЕНИЙ КАК АВТОМАТ МУРА

1. Множеством входных символов данного автомата является объединения выходных множеств всех автоматов и «пустой символ», говорящий о том, что на данный момент выходов от автоматов-процессов нет.
2. Выходом является вызов функции формирования сообщения.
3. Состояний у данного автомата два: «Ожидание выходного символа автомата-процесса» и «Формирование сообщения».
4. Начальное состояние «Ожидание выходного символа автомата-процесса».
5. Функция переходов:
   1. В состоянии «Ожидание выходного символа автомата-процесса»:

* Если поступивший символ говорит о необходимости формирования сообщения, автомат переходит в состояние «Формирование сообщения».
* Если не говорит, автомат остаётся в состоянии «Ожидание выходного символа автомата-процесса»

б) В состоянии «Формирование сообщения» автомат выполняет действие вызова функцию формирования сообщения. После поступления «пустого символа» автомат переходит в состояние «Ожидание выходного символа автомата-процесса».

Граф ДКА, реализующего генератор сообщений изображён на Рисунке 2.2.

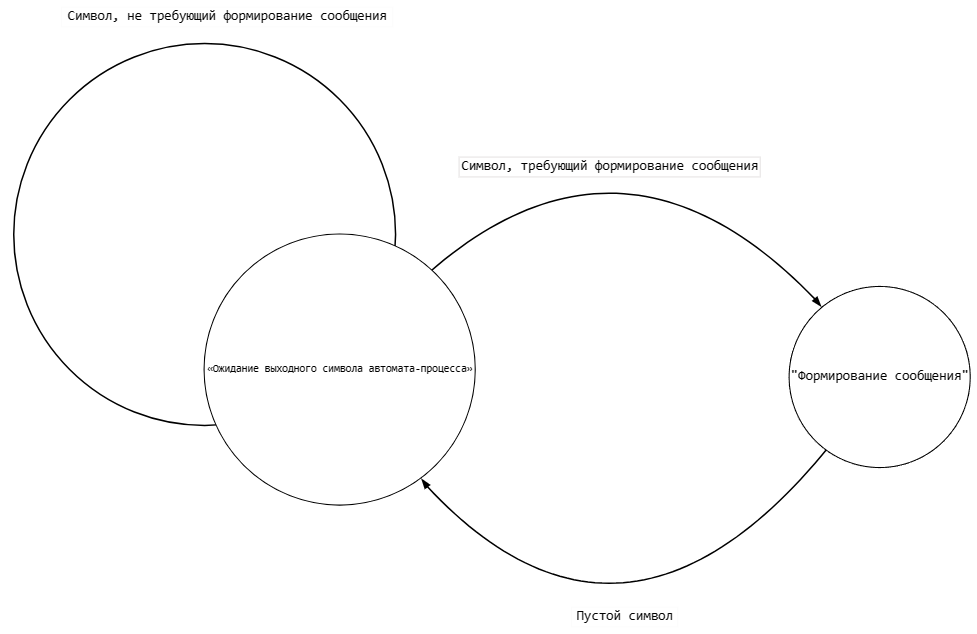


Рисунок 2.2 – ДКА, реализующий генератор сообщений.

# ЛИТЕРАТУРА

1. Функции времени [Электронный ресурс]. Режим доступа: <https://alexgyver.ru/lessons/time/> (Дата обращения 05.11.23).
2. Многозадачность в Arduino [Электронный ресурс]. Режим доступа: <https://alexgyver.ru/lessons/how-to-sketch/> (Дата обращения 05.11.23).
3. Как использовать FreeRTOS в Arduino – руководство для начинающих [Электронный ресурс]. Режим доступа: <https://microkontroller.ru/arduino-projects/kak-ispolzovat-freertos-v-arduino-rukovodstvo-dlya-nachinayushhih/> (Дата обращения 07.11.23).
4. Timer interrupts [Электронный ресурс]. Режим доступа: <https://arduinodiy.wordpress.com/2012/02/28/timer-interrupts/> (Дата обращения 07.11.23).
5. Многозадачность во встроенном приложении. Часть 1. [Электронный ресурс]. Режим доступа: <https://club.shelek.ru/viewart.php?id=362> Дата обращения 07.11.23).
6. Хопкрофт Д., Мотвани Р., Ульман Д. Введение в теорию автоматов, языков и вычислений, 2-е изд. : Пер. с англ. — М.:Издательский дом «Вильямс», 2002. — ISBN 5-8459-0261-4
7. Конечные автоматы в микроконтроллерах AVR [Электронный ресурс]. Режим доступа: <https://microkontroller.ru/programmirovanie-mikrokontrollerov-avr/konechnye-avtomaty-v-mikrokontrollerah-avr/> Дата обращения 21.11.23).
8. Moore, Edward F. Gedanken-experiments on Sequential Machines // Automata Studies,Annals of Mathematical Studies. — Princeton, N.J.: Princeton University Press, 1956. — No. 34. — P. 129—153.
9. Mealy, George H. A Method to Synthesizing Sequential Circuits — Bell Systems Technical Journal, 1955. — P. 1045—1079.