МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ
НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ УКРАИНЫ
«КИЕВСКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ»

КАФЕДРА КЭВА

РАСЧЕТНО-ГРАФИЧЕСКАЯ РАБОТА

По курсу: «Электронно-вычислительные устройства и системы»

Выполнил:

Студент группы ДК-41

Белаш Б.О.

Проверил:

Лысенко А.Н.

Киев – 2018

СОДЕРЖАНИЕ:

Анализ проблемы……..3

Решение проблемы……4

Реализация решения……..6

АНАЛИЗ ПРОБЛЕМЫ.

При изучении новых контроллеров PIC32 изначально достаточным был один из двух подходов к проектированию устройств на базе микроконтроллеров. Этот метод достаточно прост и известен – суперцикл. Это просто бесконечный цикл, который выполняет по кругу вложенные в него команды. Выглядит он следующим образом:

```
int main() {
	while(1) {
		doSomething();
		doSomethingElse();
		doSomethingMore();
	}
}
```

Для лабораторных работ начального уровня этого было более, чем достаточно. Однако далее, по мере нарастания сложности необходимых условий работы контроллера суперцикл мог проседать в быстродействии выполнимой работы. Конечно, в лабораторных условиях и поставленных задачах это практически незаметно. Плюс прерывания, которые используются были сложными и выполнялись в суперцикле. Также существует определенная аксиома, что любой контроллер можно заставить работать так как нужно без применения ОС.

Но не всегда это удобно. Ведь в условиях, где быстродействие крайне важно, в момент, когда пришел внешний источник прерывания, пока в суперцикле не пришла очередь проверки этого прерывания – оно не выполнится. И на это тратится время.

То есть можно сделать вывод. Что одним из главных минусов суперцикла идет его последовательность, и проверки определенных условий если прошли свою очередь, то будут по кругу ожидать своего следующего решения.

На помощь в таком случае приходит операционная система реального времени ОСРВ (Real-time operation system RTOS).

РЕШЕНИЕ ПРОБЛЕМЫ

ОСРВ в своей специфике и основе довольно проста. Для новичка, который привык работать в суперцикле, достаточно трудно самостоятельно прийти к ОРСВ, однако увидев ее реализацию, сразу становится ясна ее простота и основная идея. Для примера рассмотрен одновременный опрос датчика с заданной частотой и вывод на консоль длинной отладочной строки. В основе ОСРВ лежит следующий принцип:

```
void SensorPollingTask() {
      while (1) {
            value = SensorRead();
            if (value > LIMIT) {
                  doSomething();
            }
            taskDelay(10_MILLISECOND_DELAY);
      }
}
void DebugTask() {
      dbg_task_queue = os_queue_create();
      while (1) {
            char * str = os_queue_read(dbg_task_queue);
            foreach (ch in str) {
                  put_ch(ch);
            }
      }
}
void OtherTask() {
      other_task_init(); …
      while(1) {
            …
            // we want to do a dbg_printout here
            os_queue_put("Long Debug Output String");
```

```
            …
        }
}
int main() {
        os_task_create(SensorPollingTask, HIGH_PRIORITY);
        os_task_create(DebugTask, LOW_PRIORITY);
        os_task_create(OtherTask, OTHER_PRIORITY);
        os_start_sheduler();
}
```

Как видно, в главной функции больше нет одного главного бесконечного цикла. Вместо него – отдельный бесконечный цикл в каждой задаче. (функция os_start_sheduler(); никогда не вернет управление!). Что самое главное – у этих задач есть приоритеты. ОСРВ сама обеспечит то, что нам нужно – чтобы задача с высоким приоритетом выполнялась прежде всего, а с низким – только лишь тогда, когда ей остается время.

И если время реакции на, например, прерывание в дизайне с суперциклом будет равно в худшем случае времени выполнения всего цикла (прерывание случится сразу же, но далеко не всегда необходимые действия можно сделать непосредственно в обработчике), то время реакции в случае ОСРВ будет равно времени переключения между задачами (которое достаточно мало, чтобы считать, что это происходит сразу же!). То есть прерывание произойдет в одной задаче, а сразу по его завершению переключится на другую задачу, ожидающую события, «запущенного» из прерывания.

Также если изначально не будет замечена определенная ошибка, то «упадет» только та задача, в которой будет присутствовать ошибка (возможно также, что и все задачи с более низким приоритетом тоже), но задачи с более высоким приоритетом продолжат выполняться, обеспечивая хотя бы минимальные жизненно важные функции устройства, например, защиту от перегрузки.

И подводя итог: если система очень простая и нетребовательная ко времени реакции, ее проще сделать по образцу «суперцикл». Если же система собирается стать большой, соединяющей в себе много разных действий и

реакций, которые к тому же критичны ко времени – то альтернативы использования ОСРВ.

Кроме этого, плюс использования ОСРВ – более простой и понятный код, поскольку можно группировать код по задачам, избегая глобальных переменных, машин состояний и прочего мусора, необходимого при использовании дизайна с суперциклом.

Минус использования ОСРВ – для ее использования требуется больше места, памяти, опыта и знаний (хотя ничего сложного там и нет, все же многозадачность изначально сложнее и непредсказуемее, чем последовательно выполняющийся код).

РЕАЛИЗАЦИЯ РЕШЕНИЯ

Изначально следует отметить, что помимо ОСРВ, которая является ОС «жесткого» реального времени, существуют другие ОС, «мягкого» реального времени. К ним относятся привычные Windows, Linux. Пользователь может видеть, что, например, нажав кнопку с символом, он видит введенный символ, а если же он нажал кнопку, и спустя время не увидел реакции, то ОС будет считать задачу «не отвечающей» (по аналогии с Windows — «Программа не отвечает»), но ОС остается пригодной для использования. Таким образом, ОСРВ мягкого времени просто определяет предполагаемое время ответа, и если оно истекло, то ОС относит задачу к не отвечающим.

К ОСРВ жесткого типа, как было сказано ранее, как раз относят ОСРВ во встраиваемых устройствах. Они имеют главное отличие — каждая задача должна выполняться за отведенный квант времени, не выполнение данного условия ведет к краху все системы.

Многозадачность. ОСРВ предоставляет программисту готовый, отлаженный механизм многозадачности. Теперь каждую отдельную задачу можно программировать по отдельности так, как будто остальных задач не существует. Например, можно разработать архитектуру программы, то есть разбить ее на отдельные задачи и распределить их между командой программистов. Программисту не нужно заботиться о переключении между задачами: за него это сделает ОСРВ в соответствии с алгоритмом работы

планировщика. Временная база. Необходимо отмерять интервалы времени? Пожалуйста, любая ОСРВ имеет удобный программный интерфейс для отсчета интервалов времени и выполнения каких-либо действий в определенные моменты времени. Обмен данными между задачами. Необходимо передать информацию от одной задачи к другой без потерь? Используйте очередь, которая гарантирует, что сообщения дойдут до адресата в том объеме и в той последовательности, в которой были отправлены. Синхронизация. Разные задачи обращаются к одному и тому же аппаратному ресурсу? Используйте мьютексы или критические секции для организации совместного доступа к ресурсам. Необходимо выполнять задачи в строгой последовательности или по наступлении определенного события? Используйте семафоры или сигналы для синхронизации задач. Кроме этого, одна и та же ОСРВ для МК может выполняться на множестве архитектур микроконтроллеров.