Секундный таймер с нулевой погрешностью

Введение

Простая и быстрая система получения точных временных периодов на микроконтроллере PIC.

Эта система (ассемблерный код для PIC прилагается) даёт простой, быстрый способ генерировать достоверные периоды на микроконтроллере PIC с любой тактовой частотой. Особенно для односекундных событий, таких как простые часы. Вы можете использовать любой кварц, какой у вас есть: 4.0 МГц, 12.137253 МГц; ЛЮБОЙ кварц и ЛЮБОЕ значение предделителя, и всё равно будете получать идеальную односекундную синхронизацию.

Система будет генерировать достоверные периоды от миллисекунд до многих секунд при очень быстром исполнении кода.

Замечание! Эта система может иметь колебания, то есть каждый период может быть частично дольше или короче, но полное функционирование таймера даёт нулевую погрешность, идеальную для часов и регистраторов данных. Процент дребезжания можно легко минимизировать, если это важно для вас.

Теория

Здесь используется модифицированная версия алгоритма Брезенхама, который сначала использовался для обеспечения быстрой обработки движения пера в перьевых графопостроителях.

Система Брезенхама идеальна для пошагового перемещения на точное расстояние с нулевой погрешностью (однако при этом важно понимать, что каждый шаг будет немного длиннее или короче).

Представьте, что вы можете двигаться только по "клеточкам" сетки, то есть вы можете сдвинуться только на 3 или на 4 клеточки, но в целом вам нужно сдвигаться на 3.37629 клеточки при каждом движении. Всё, что вы делаете - это сдвигаетесь иногда на 3, а иногда на 4 клеточки. Система обеспечивает итоговую нулевую погрешность и всегда выбирает ближайшую "клеточку" (или 3 или 4) для этого движения.

Красота системы в том, что вы можете использовать любую частоту и любую длину периода. И всё равно будете получать периоды с нулевой погрешностью.

Вместо того чтобы объяснять алгоритм Брезенхама и его многочисленные вариации, обратимся к конкретному примеру.

Процедура

Пример

PIC тактовой с частотой 4 МГц = 1 000 000 отсчётов в секунду.
Прерывание каждые 256 отсчётов.
Мы хотим сгенерировать 1-секундное событие.
(24-битная переменная требует три регистра PIC-микроконтроллера.)

Процедура

- При загрузке мы добавляем 1 000 000 в вашу 24-битную переменную.
- Прерывание происходит каждые 256 отсчётов.
- При каждом прерывании мы вычитаем 256 из нашей 24-битной переменной.
- Когда значение переменной становится меньше нуля, мы генерируем событие и снова **прибавляем** 1 000 000 к переменной.

Это довольно просто - когда переменная становится меньше нуля, она содержит погрешность, таким образом, ПРИБАВЛЕНИЕ 1 000 000 к ней удаляет эту ошибку из следующей секунды, давая нулевую погрешность для любого более длительного периода времени.

Замечание по колебанию! Максимальное отклонение - это число, которое мы вычитаем при каждом прерывании, таким образом, в данном случае максимальная погрешность - 256 отсчётов или 0.0256%. Это около 1/40 максимума процента ошибки для каждой секунды, полная же ошибка для длительного срока равна НУЛЮ.

Улучшенная процедура

Улучшения, которые я произвёл в моём проекте, базируются на аппаратных особенностях микроконтроллера PIC. Очень легко вычитать 256 из 24-битной переменной - вы просто вычитаете «1» из среднего байта переменной, и если знак получился отрицательный, вычитаем «1» из старшего байта.

Эта операция эквивалентна обычному 24-х битному вычитанию, а реализуется всего несколькими командами.

Другое улучшение связано с перемещением точки "события" от генерации события при "меньше нуля" к при "меньше 256". Когда старший и средний байты равны нулю, наступает событие "меньше 256".

Нам не нужно постоянно заботится о младшем байте и обрабатывать отрицательные значения.

Также было исправлено "прибавление 1 000 000", потому что у нас теперь нет отрицательных значений, и старший и средний байты будут всегда равны нулю, когда мы выполняем прибавление. Нам только нужно прибавить МЛАДШИЙ байт, проверить на перенос в средний байт, затем просто загрузить средний и старший байты (это намного быстрее, чем обычное 24-битное сложение).

Пример

PIC тактовой с частотой 4 МГц = 1 000 000 отсчётов в секунду.
Прерывание каждые 256 отсчётов.
Мы хотим сгенерировать 1-секундное событие.
(24-битная переменная требует три регистра PIC-микроконтроллера.)

Процедура

- При загрузке мы добавляем 1 000 000 + 256 в нашу 24-битную переменную.
- Прерывание происходит каждые 256 отсчётов.
- При каждом прерывании мы декрементируем наш средний байт (вычитаем 256)

- Проверяем на знак и декрементируем старший байт, если нужно.
- Когда старший и младший байты оба равны нулю, мы генерируем событие и прибавляем 1 000 000 в 24-битную переменную.

Помните, что "прибавить 1 000 000" теперь очень просто - мы просто **загружаем** старший и средний байты, затем **прибавляем** младший байт и, если он переполняется, инкрементируем средний байт.

Пример кода

Без использования прерывания

Этот пример подходит для любого PIC - он будет работать с 12-разрядными устройствами, подобными 12C508, с 14-разрядными устройствами, подобными 16F84 и т.д.

Он не требует какого-либо прерывания или вызовов, поскольку стек не используется.

Он генерирует период в одну секунду при тактовой частоте PIC 4 МГц, бит 7 таймера TMR0 используется как фиктивное "прерывание".

С использованием прерывания

Этот пример подходит для микроконтроллеров PIC с прерыванием по переполнению таймера TMR0. Односекундный таймер в данном случае автоматический и не требует опроса.

Он генерирует период в одну секунду при тактовой частоте PIC 4 МГц, прерывание таймера TMR0 устанавливается на возникновение каждые 256 команд.

Улучшения и предложения

Оба примера кода, приведённые выше, имеют частоту возникновения "прерывания" каждые 256 команд. Это сильно упрощает прибавление и вычитание и делает их очень быстрыми.

Для приложений, где вы не возражаете против немного большего колебания, вы можете делать прерывания каждые 65536 (256x256) команд, это будет также просто и быстро, вам только нужно будет делать проверку каждые 65536 отсчётов, что оставляет больше времени для другого вашего кода. Итоговая ошибка останется равной нулю, но колебание в каждой отдельной секунде увеличится. Максимальное колебание составит около 0.066 секунды или около 7 %. Все равно это даёт точный отсчёт времени, так как никого не волнует, что секунда выдаётся на 7 % раньше или позднее, и в итоге часы обеспечивают точное время благодаря тому, что итоговая ошибка равна нулю.

Как заявлено выше, эта система будет работать с любой тактовой частотой и любой длиной периода, однако, если ваше прерывание не происходит с множителем 256, вам нужно будет использовать немного более длинное 24-битное сложение и вычитание, хотя вы можете оптимизировать код сложения и вычитания до более быстрого, чем обычное.

Примеры кода снабжены комментариями и могут быть легко адаптированы под ваши нужды.

Окончательные примеры

Пример 1

Кварц в 4.43 МГц можно найти почти в каждом старом телевизоре или видеомагнитофоне. Он отлично подходит для «домашних» проектов, где нужна синхронизация, и если вы не хотите покупать другой нормальный кварц.

4.43 МГц / 4 = 1 107 500 команд в секунду.

Просто используйте коды примеров, приведённые выше, и загрузите в 24-битную переменную значение 1 107 500 = 10 E6 2C (в шестнадцатеричном формате). Это даст односекундное событие.

Конечно, вы можете использовать любой высокоскоростной кварц и просто менять добавляемое значение.

Пример 2

Сгенерировать 1200 Гц при кварце 16 МГц.

Это может быть полезным для PIC приложений, в которых нужна реализация последовательного протокола связи с заданной скоростью передачи (в нашем случае 1200 бод).

16 МГц / 4 = 4 000 000 команд в секунду

4 000 000 / 1200 = 3333.3 отсчётов в секунду.

Просто используйте код примеров, приведённых выше, и загрузите в 24-битную переменную значение 3333 = 00 0D 05 (в шестнадцатеричном формате).

На б*о*льших частотах могут возникнуть проблемы с колебаниями, максимальное колебание будет около 256 отсчётов, которое составляет около 8 % на частоте 1200 Гц. Это ещё довольно-таки неплохо.

Заключение

Недостатки

- Эта система добавляет небольшой сдвиг каждому периоду (плавает продолжительность каждого периода)

Достоинства

- Нулевая итоговая погрешность, идеальная для часов и регистраторов данных.
- Очень быстрое выполнение
- Может быть добавлена в существующую разработку с любой частотой кварца.
- Даёт фактически любой период на любом кристалле.
- Периоды могут быть изменены программно и всё равно останутся точными.
- Может быть добавлена в ваше существующее прерывание.
- Прекрасно подходит для дешёвых PIC
- В большинстве (односекундных) случаев колебание очень маленькое, меньше 0.05 %.