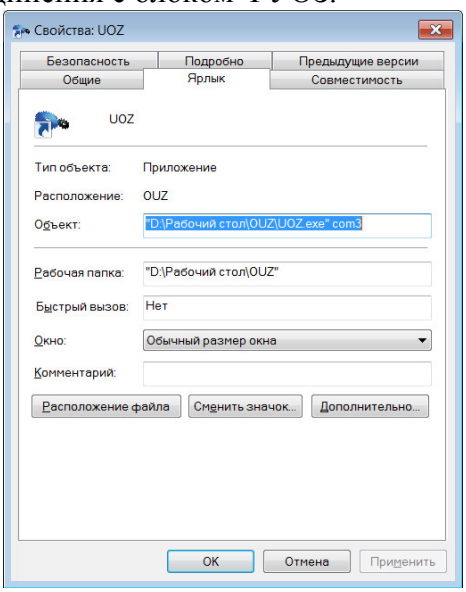
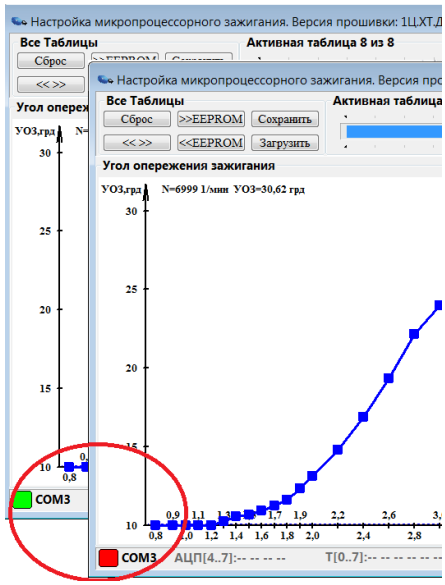
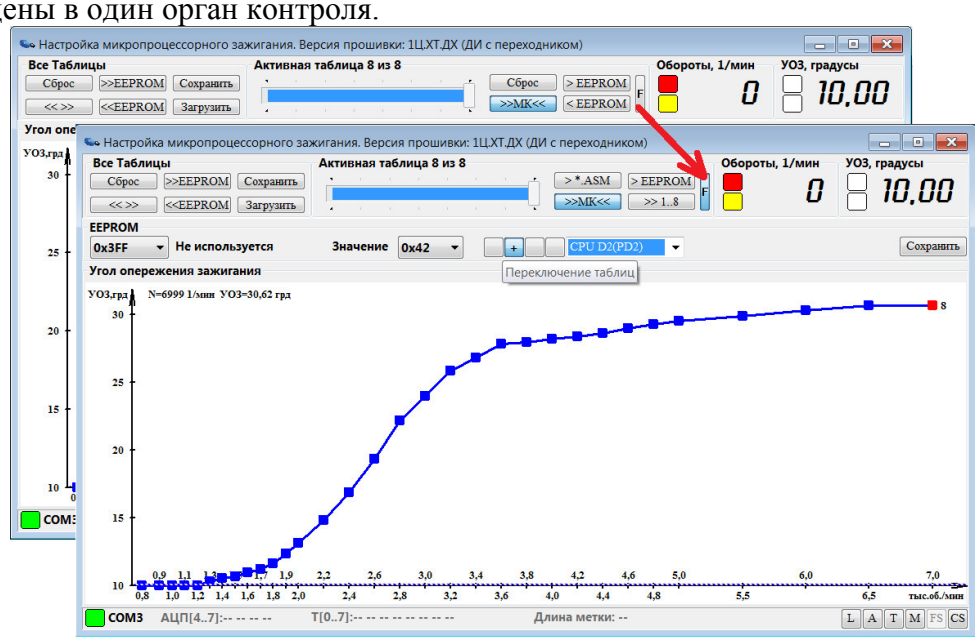
**Версия от 01.10.2021**
**(с учетом изменений от 28.08.2021)**

1. В программу UOZ добавлена поддержка номера COM порта, указанного как параметр при запуске. Ускоряет запуск при больших номерах COM портов. Поддержка – до COM15. Добавлен индикатор соединения с блоком ФУОЗ.




2. Размер Flash памяти (энергонезависимой памяти EEPROM) микроконтроллера 1024 байта. На каждую страницу отводится 32 байта. В случае, если используется менее максимального количества (32) таблиц, есть свободная половина EEPROM, а то и больше. Даже в случае использования максимального количества страниц можно найти точку в наборе таблиц, выбор которой Ядром будет маловероятным. Для достижения этой цели введены несколько новых макросов в Ядро, а так же создана система команд по управлению Flash памяти – система Flash команд по интерфейсу UART. В Ядро добавлен признак поддержки этих команд. Система Flash команд приведена в файле «команды.ods» - выделено зеленным цветом. К сожалению, из-за необходимости сохранения обратной совместимости версий и нежелательности увеличения трафика по интерфейсу UART работы с этими командами носят весьма сложный характер. Внесены изменения и в программу OUZ. Она теперь различает Ядра с поддержкой Flash команд и при включении дополнительного функционала, описанного в предыдущем посте (кнопка F), дает пользователю доступ к содержанию всего адресного пространства памяти EEPROM. Пользователю остается выбрать ячейку энергонезависимой памяти с неиспользуемым Ядром адресом и пользоваться её битами для управления параметрами Ядра (**EEPROM аналог DIP переключателей**). Четыре правых биты предназначены для управления светодиода на плате и для удобства объедены в один орган контроля.



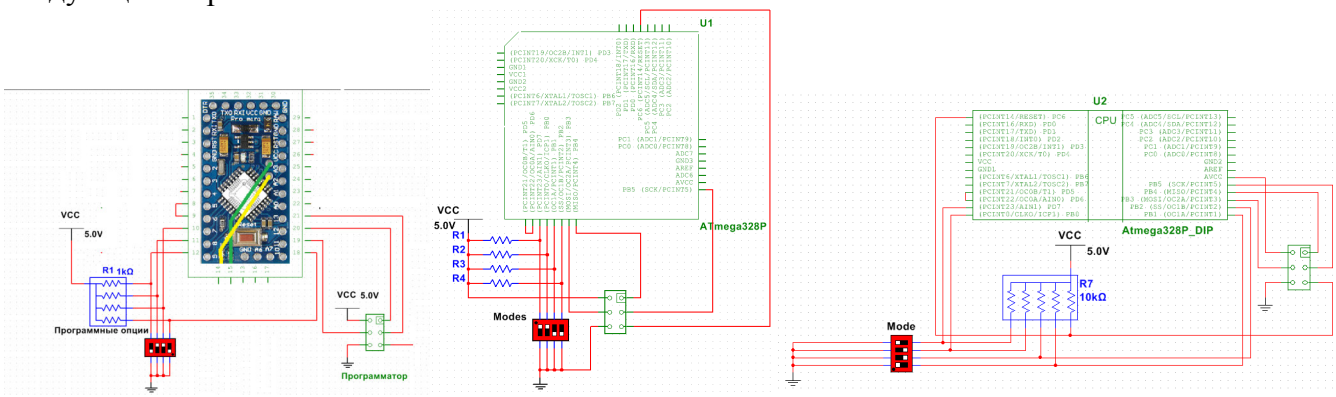
Для чтения EEPROM аналогов DIP переключателей служат макросы: FLDS Reg, Addr или FLD Reg, X(Y)(Z). Для корректной установки режимов следует воспользоваться макросом: SetLEDandDIPCtrl Reg. Пример:

```
FLDS            A, 0x3FF
SetLEDandDIPCtrl A
```

Назначение EEPROM аналогов DIP переключателей следующее:

| | |
|---|---|
| 7 | Если бит равен 1 (+ в программе), то ручной корректор включен |
| 6 | Если бит равен 1 (+ в программе), то переключатель таблиц включен. |
| 5 | Если бит равен 1 (+ в программе), то корректор нагрузки включен. |
| 4 | Если бит равен 1 (+ в программе), то корректор нагрузки будет продолжать работу на оборотах, выше табличных. |
| 3-0 | Выбор режима работы установленного на плате Arduino светодиода. |

3. Добавлена поддержка и **DIP переключателей (перемычек).** Анализ схем коллег дал возможность постулировать общий подход к построению вариантов управления с помощью DIP перемычек на свободных входах микроконтроллера. Так DIP переключатели при желании следуют подключить следующим образом:




Назначение DIP переключателей следующее:

| | |
|---|---|
| PD7(ain1) | Если уровень на входе равен 0, то ручной корректор отключен |
| PB0(icp) | Если уровень на входе равен 0, то переключатель таблиц отключен. |
| PB1(osc1) | Если уровень на входе равен 0, то корректор нагрузки отключен. |
| PB2(ss) | Резерв |

Для корректного заполнения любого регистра для последующей установки в качестве параметров, также как и в случае с EEPROM аналогом DIP переключателей, создан макрос CopyLEDandDIP Reg,Value. Здесь Value - выбор режима работы установленного на плате Arduino светодиода.

Пример:

```
CopyLEDandDIP    A, 2              ; Светодиод отражает состояние линии CPU D2(PD2)
SetLEDandDIPCtrl A
```

Если нет необходимости использования ни DIP переключателей, и их EEPROM аналогов, то все параметры устанавливаются в теле программы, кроме выбора режима работы установленного на плате Arduino светодиода. Для этого создан макрос SetCoreSetup Value.

Пример:

```
SetCoreSetup   0b00000100        ; Отключен корректор нагрузки
```

Назначение битов

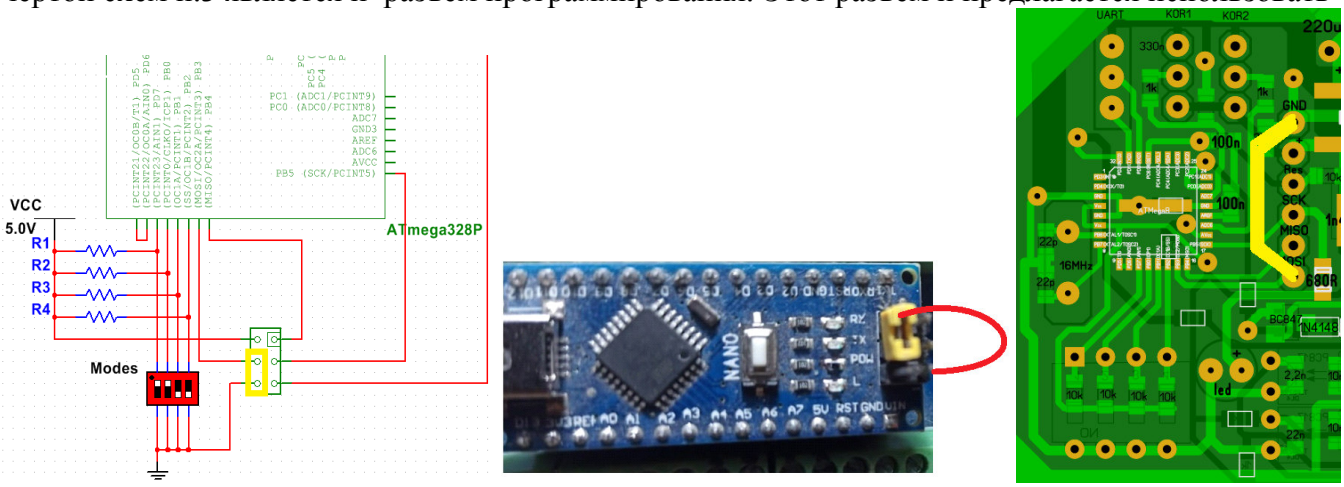
| | |
|---|---|
| 7-4 | Служебный флаг - при настройке может иметь произвольное значение |
| 3 | Режим работы корректора нагрузки на оборотах выше табличных (1 корректор будет продолжать работу при высоких оборотах) |
| 2 | Отключен корректор нагрузки |
| 1 | Отключен АЦП |
| 0 | Отключен UART |

Начальное значение и значение «по умолчанию» - **все включено.** Обращаю внимание, что если соответствующие каналы АЦП для обеспечения функций ручного корректора и переключателя таблиц не включены, то все манипуляции с DIP переключателем, в том числе с их EEPROM аналогами, не будут приводить ни к каким результатам. То есть - **выключатель любой может отключить или включить, только включенную ранее макросом SetCoreSetup функцию!**

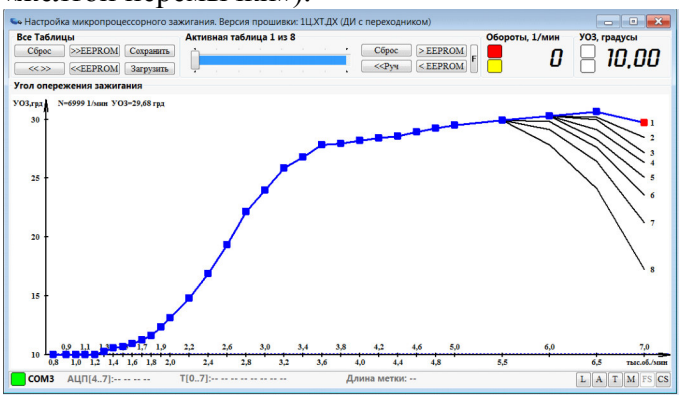
Для отдельного управления им создан макрос SetLEDCtrl Value
Пример:

```
SetLEDCtrl      2          ; Светодиод отражает состояние линии CPU D2(PD2)
```

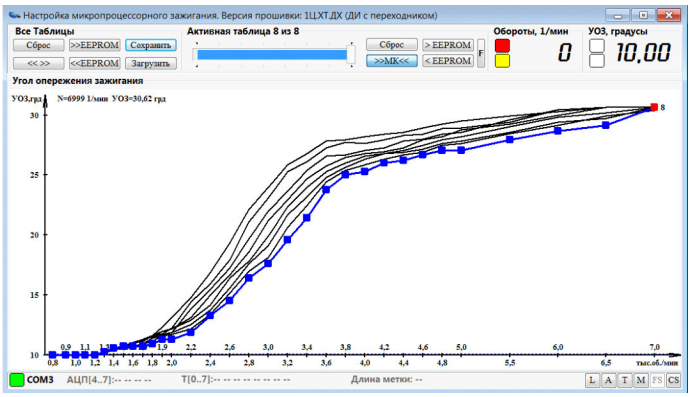
4. В Ядро добавлена функция «**желтая перемычка**» - при наличии этой перемычки набор таблиц будет считан из области программы, а не из области EEPROM как обычно. Данную функцию можно использовать или как выбор альтернативного набора таблиц или выбор источника таблиц при аварии внутренней EERPOM (аварийный режим восстановления работы без компьютера). Видно, что общей чертой схем п.3 является и разъем программирования. Этот разъем и предлагается использовать




При включении «желтой перемычки» Ядро сверит контрольную сумму сохраненного набора таблиц в области программ. В случае ошибки таблицы будут загружены как обычно из EEPROM (как без «желтой перемычки»).




Загрузка из EEPROM




Загрузка из области программы («желтая» перемычка)

Для обеспечения возможности такого режима расширил действие программы UOZ при нажатии на кнопку “ > *.ASM ”. После отладки набора таблиц следует нажать на эту кнопку и произвести повторное программирование МК измененной прошивкой. В конец листинга исходного файла теперь сохраняется и весь набор таблиц (с контрольной суммой) для восстановления без компьютера.

5. Применена преобразованная функция округления при задании констант:
$round\left(\frac{A}{B}\right) = int\left(\frac{A}{B} + 0{,}5\right) = int\left(2\frac{A}{B} + 1\right) \gg 1$, где “>>1“ – логический сдвиг в право (деление на 2). В противном случае иногда имеет место потери в младшем разряде из-за «глюков» avrmasm2.

6. Программное деление столбиком на константу условную К заменено на многоразрядное умножение с помощью встроенного умножителя mul с учетом сдвига результата:

$$int\left(\frac{A}{\text{К}}\right) = \text{int}\left(\frac{A}{2^P} * \frac{2^P}{K}\right) = int\left(\frac{A * C}{2^P}\right) = int(A * C) \gg p$$

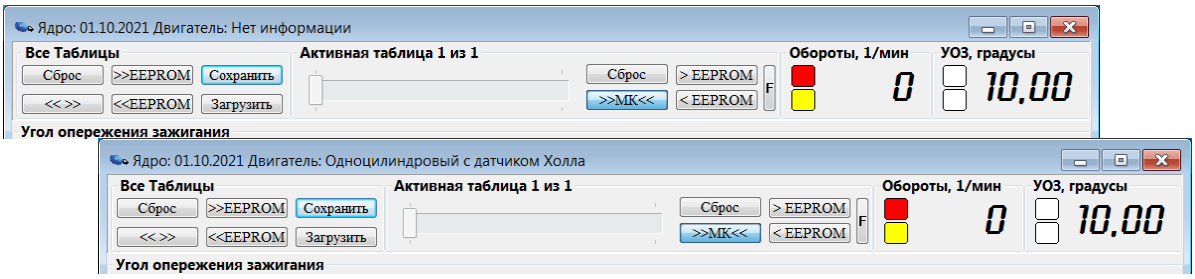
Сдвиг вправо на p разрядов осуществляется автоматически при суммировании после умножения. Константа $C = int\left(\frac{2^P}{K}\right)$ при необходимости предварительно размещается в ОЗУ. Степень p выбирается исходя из требуемой точности.

7. Применен механизм условной компиляции по средствам директив avrasm2 (.if, #ifdef, #define ....). **Цель - сокращение объемов ОЗУ, ускорение работы прошивки при не использовании отдельных узлов за счет их автоматического изъятия из кода при компиляции**. Минимизировано количество блоков, которых требуется включать или отключать по флагам. Это введение позволило **кардинально изменить подход к заданию параметров прошивки**.

8. Длина строки версии прошивки увеличена до 127 символов. Строка "изъята" из ОЗУ. Освобожденное ОЗУ используется для хранения чисел механизма п.6. Сама строка версии разделена на две части – Дата Ядра и Модель или описание двигателя. Первая часть сохранена в Ядре. Вторая часть **может быть добавлена** с помощью механизма п.7. пользователем в моторной части строкой:

**#define _EngineInfo "Одноцилиндровый с датчиком Холла ".**

Ключевые слова - «**может быть добавлена**». Иными словами, такой строки может не быть совсем. На рисунке приведено отличие в отображении программой UOZ в этих случаях.




9. Откорректирована (оптимизирована) функция поиска номера диапазона Selectr - номер ячейки в таблице.

10. Исправлен математический аппарат по проведению линейной интерполяции и вычисления кода для таймера «задержки» между узлами. Математический аппарат приведен в соответствие формулам в описании предыдущей версии:

$$\text{OCR1A} = \frac{N \cdot \left[fmul\left\{B_k^{fp8} \cdot A_{k+1}^{fp8}\right\} + fmul\left\{128 - B_k^{fp8},\ A_k^{fp8}\right\} + C^{fp16}\right]}{\text{Ka} \cdot 32768} - \Delta\text{T}$$

$$A_k^{fp8} = int\left[\frac{\Delta - \alpha(M, \ldots)}{Base - \Delta} \cdot Ka \cdot 128\right],\ B_k^{fp8} = int\left[\frac{N - n_k}{n_{k+1} - n_k} \cdot 128\right],\qquad k = Selectr,$$

$$C^{fp16} = -round\left[\frac{\text{Апертура} \cdot Ka \cdot 32768}{Base - \Delta}\right] + 2 \cdot round\left[\frac{\text{Апертура} \cdot Ka \cdot 32768}{Base - \Delta} \cdot \frac{ADC_j}{256}\right] \qquad (1)$$

$$\Delta T = round\left[(ProtectCount \cdot\ ProtectClk + 6)\frac{8}{1 + Fn}\right]$$

$$TblSelectr = int\left[ADC_i \cdot \frac{TblCount}{256}\right]$$

i – номер канала АЦП для выбор таблицы, j – номер канала АЦП ручного корректора УОЗ
Ранее в прошивке были технической ошибке использованы эквивалентные на 100% формы.

11. С помощью механизма п.6 применена альтернативная функция деления произведения $\frac{N\cdot\left[fmul\left\{B_k^{fp8}\cdot A_{k+1}^{fp8}\right\}+fmul\left\{128-B_k^{fp8},\ A_k^{fp8}\right\}+C^{fp16}\right]}{32768}$ из (1) на константу Ka через многоразрядное умножение - $C = int\left(\frac{2^{16}}{Ka}\right)$. Для учета частных случаев, когда Ka есть 2 в степени создан макрос DivKa.

12. Деление в выражении $B_k^{fp8} = int\left[\frac{N-n_k}{n_{k+1}-n_k}\cdot 128\right]$ (1) также заменено на многоразрядное умножение на заранее рассчитанные константы $C_k = int\left(\frac{2^{24}}{n_{k+1}-n_k}\right)$

$$B_k^{fp8} = (N-n_k)\cdot C_k \gg (24-7)$$

Вычисление трехбайтовых констант $C_k$ происходит в Главной разметочной таблице. Макрос FTW в Ядре откорректирован для заполнения ОЗУ этими константами FTW $n_k,\ C_k$ -

**FTW 0x1BDC, 0x007760**

**Все вышеперечисленное позволило значительно снизить время вычисления**. Следует отметить что, из-за особенности функции поиска номера диапазона Selectr имеет место быть полезная особенность - чем выше обороты, тем меньше время выполнения математических расчетов. В таблице сведены результаты сравнения скорости работы предыдущей и новой версии прошивки.

| Версия прошивки | Затраты на вычисления, мкс/град | | |
|---|---|---|---|
| | Минимальные обороты по таблице | Максимальные обороты по таблице | Обороты выше табличных |
| Предыдущая | 76/0,4 | 56/2,4 | 30/1,3 |
| Новая | 41/0,2 | 21/0,9 | 16/0,7 |

В знаменателе указаны потери времени в углах поворота маховика. Обращаю внимание на значении 2,4 градуса. Это именно те потерянные ~3 градуса, про которые сказано в Главной разметочной таблице. предыдущей версии прошивки. **Теперь это менее 1 градуса без потери точности вычисления!** Безусловно, объем используемых ресурсов ОЗУ возрастает. Так при максимальных параметрах системы ОЗУ используется на 97%.

13. Все параметры Ядра теперь имеют значения по «умолчанию». Если пользователь в моторной части «забыл» задать параметр с помощью директивы **#define,** то будут действовать следующие значения по умолчанию:

| Параметр | Назначение | Значение по умолчанию |
|---|---|---|
| **_EngineInfo** | "Нет информации" | - |
| **_Base** | Угол-база в градусах | 360 |
| **_Delta** | Угловой размер метки Delta | 30 |
| **_Betta** | Угловое расстояние от ВМТ до метки Delta. | 0 |
| **_UOZMax** | Максимальный угол опережения зажигания | _Betta + _Delta-1 |
| **_UOZMin** | Минимальный угол опережения зажигания | _Betta |
| **_Alfa** | Угловое расстояние отключения сигнала зажигания после метки Delta | 0.5 |
| **_Fn** | Частота тактовых импульсов, поступающих на таймер T1 | 8 |
| **_Load** | Угловая апертура корректора нагрузки в "-", град. | 0 |
| **_LoadStartSelectr** | Номер начального диапазона работы корректора нагрузки | 31 |
| **_LoadBufferSize** | Длина кольцевого буфера корректора нагрузки | 64 |
| **_ProtectClk** | Интервал дискретизации протокола защиты от пульсаций в мкс | 6 |
| **_ProtectCount** | Количество повторных чтений протоколом защиты с интервалом ProtectClk после первого изменения линии датчика | 2 |
| **_ADCBufferSize** | Длина буферов АЦП | 8 |

| Параметр | Назначение | Значение по умолчанию |
|---|---|---|
| **_StopTime** | Временной интервал для расчета времени установления режима СТОП, сек. | 1 |
| **_Manual** | Константа для ручного корректора, град. | 0 |
| **_ADCCanCorSelectr** | Номер канала АЦП Ручного корректора УОЗ | 0 |
| **_DoInvertADCCorSelectr** | Инверсия кода АЦП канала _ADCCanCorSelectr код = (255-код) | отключена |
| **_TblCount** | Количество таблиц УОЗ | 1 |
| **_ADCCanTblSelectr** | Номер канала АЦП выбора активной таблицы УОЗ | 0 |
| **_DefaultTbl** | Номер активной таблицы в случае отключенного канала АЦП _ADCCanTblSelectr и количества таблиц более 1 | _TblCount-1 |
| **_DoInvertADCCorSelectr** | Инверсия кода АЦП канала _ADCCanTblSelectr код = (255-код) | отключена |
| **_ADCCanA** | Номер канала АЦП свободного канала | 0 |
| **_ADCCanB** | Номер канала АЦП свободного канала | 0 |
| **_DoAverage** | Включено усреднение по двум отсчетам оценки скорости вращения | отключено |

Пример полного набора параметров с учетом значений по умолчанию:

```
#define _EngineInfo "1Ц.4Т Индуктивный датчик с удлинением метки….."
#define _Delta              25.0
#define _Betta              6.0
#define _UOZMax             30.0
#define _UOZMin              6
#define _Fn                  18
#define _Load               3.0
#define _LoadStartSelectr 28
#define _LoadBufferSize   128
#define _DoAverage
```

14. Расширена логика управления корректора нагрузки. Введен начальный номер диапазона оборотов (Selectr), выше которых корректор работает. Введены два режима его работы: только для верхней границы таблиц УОЗ, и для всех оборотов.

15. Вспомогательные функции для событий ServiceA, ServiceB, Loop переведены в режим подключаемых с помощью механизма п.7. Если в моторной части будет строка

```
#define _ServiceA  ProgName
```

и где-то в листинге будет описана подпрограмма

```
ProgName:
        do something
        do something
        ret
```

то эта подпрограмма будет вызываться в соответствующий момент времени функционирования Ядра. То же справедливо и для событий ServiceB и Loop. Напомню, что ни какие регистры (кроме temp, ZL, ZH) изменять в подпрограмме нельзя без предварительного сохранения в стеке и последующего их восстановления в стеке. Подпрограмма RESET, по прежнему обязательная должна быть в моторной части.

16. Весь механизм сравнения регистра событий Events с заранее известными кодами перенесен из моторной части в Ядро. В исходном коде Ядра определены 4 **потенциальных вызова** подпрограмм EventX (X=0..3). Для активации нужного событие следует указать код события. Если в тексте моторной части будет подпрограмма

```
Event0: do something
        do something
        ret
```

то для того чтобы Ядро могло вызвать эту подпрограмму следует указать

```
#define  _Event0  0b00001010
```

Тем самым укажем Ядру, что при появления события с кодом 0b00001010 следует вызвать подпрограмму. Аналогично и с Event1, Event2, Event3. Если в процессе работы будут использованы только две подпрограммы, то лучше использовать Event0 и Event3. Механизм сравнения регистра событий первым сравнивает _Event0, а подпрограмма Event3 имеет сокращенный код, поскольку она крайняя в цепочке сравнения.

Следующим новшество – введение для каждого события **отдельной маски** _EventMaskX( X=0..3). Этот механизм позволяет расширить логику входных сигналов за счет сокращения контролируемых состояний во времени. В случае , когда _EventMaskX не определена, то сравнение происходит по четырем моментам времени. Установив маску можно отключить отдельные моменты времени или вообще весь вход контроллера:

```
#define _EventMask0  0b00110011
#define _Event0      0b00110010
#define _EventMask3  0b00111000
#define _Event3      0b00101000
.
.
Event0:
  StartDelta
  ret

Event3:
  StopDelta
  ret
```

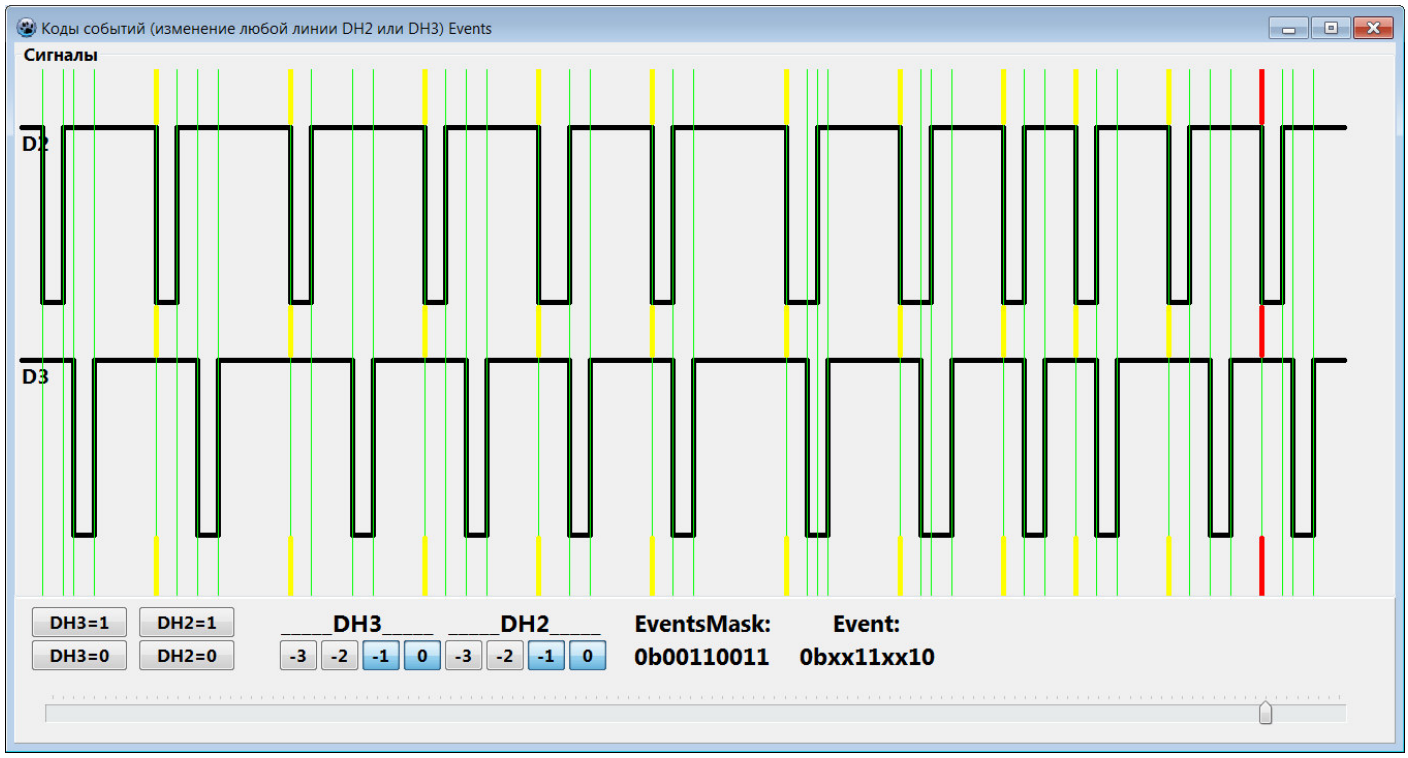
В приведенном примере событие Event0 произойдет если при неизменном входном сигнале на входе Int1/PD3/3 и равным 1, на другом входе Int0/PD2/2 **в это время** произойдет переход из 1 в 0. В свою очередь, событие **Event3** произойдет если «три изменения состояний входных линий» назад на входе Int0/PD2/2 есть спад логического уровня на входе Int1/PD3/3. Понятно, что такое временное кодирование весьма трудноосуществимо. Для создания таких кодов была разработана программа EventMaker. В программе следует нарисовать диаграммы входных сигналов, «поиграться» с масками и кодами для обнаружения уникальных комбинаций.

Приведенный пример соответствует случаю применения штатного индуктивного датчика, положительные и отрицательные импульсы которого включают диодную часть двух оптопар с триггером Шмидта на выходе.

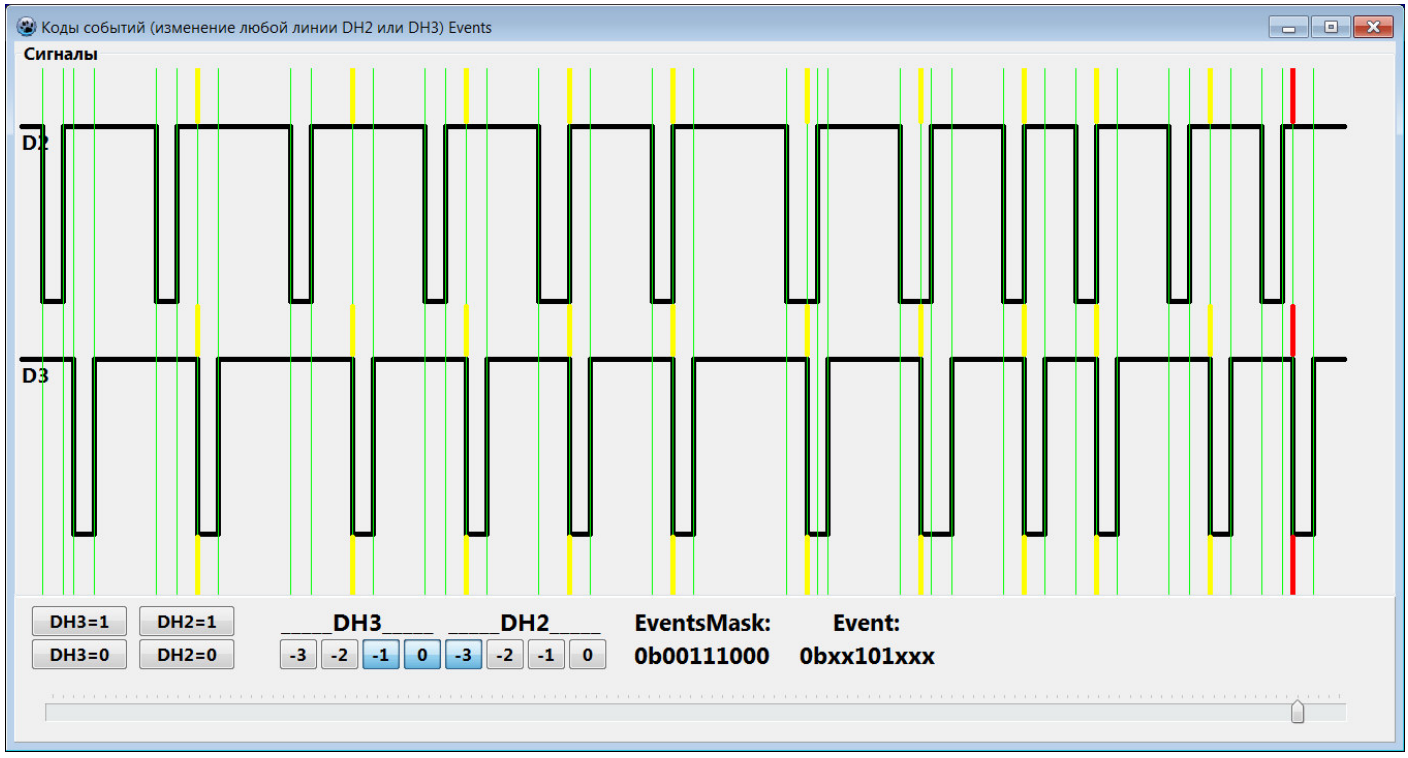
Справа представлена «собранная» из двух выборок осциллограмма этого случая. Данные любезно предоставил коллега fcftdbx. Из-за короткого расстояния между этими импульсами микроконтроллер может не успеть обработать входные линии с помощью протокола защиты от импульсной помехи на высоких оборотах без маскирования.

На оборотах 10000 1/мин расстояние между фронтом первого импульса и спадом второго всего ~40 мкс.

Ниже приведены скриншоты для вышеприведенного примера.




На диаграммах специально создается ситуация наложения импульсов. Красный маркер указывает на рассматриваемое событие. Желтые маркеры, указывают на моменты времени, где есть такие же события. Первые две пары изменений слева не учитываются. Зелеными вертикальными линиями обозначены моменты времени, когда происходит любое изменение на любом из входов.




Вместе с программой выложен и файл «event128.dat», в котором программа EventMaker хранит результаты своей работы. Можно повторить мои изыскания и возможно найти еще комбинации кодов и масок.
Отмечу, что рассматриваемый случай - чуть ли на самый сложный. В простейшем случае одного датчика Холла  все будет гораздо проще.

17. Файл «Главная разметочная таблица» тоже не мог не поменяться. Сократил многие поля, добавил информационные.