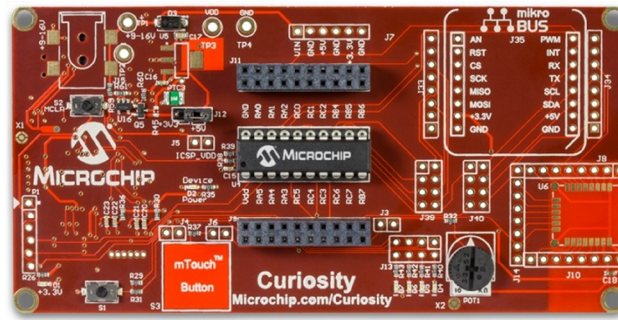


## Лабораторные работы MASTERS 2015 Russia

Лабораторные работы проводим на плате Curiosity, на контроллере PIC16F1619



Цель: Изучение Периферии Независимой от Ядра (ПНЯ) – Code Independent Peripheral (CIP).

Необходимо иметь на компьютере следующее ПО:

MPLAB X IDE (среда разработки)		<a href="http://www.microchip.com/mplabx">www.microchip.com/mplabx</a>
Плагины к MPLAB X IDE:	MPLAB Code Configurator (MCC)	Устанавливается из MPLAB X
	Data Monitor Control Interface (DMCI)	Устанавливается из MPLAB X
MPLAB XC8 (компилятор)		<a href="http://www.microchip.com/xc8">www.microchip.com/xc8</a>

### Лабораторная работа 1

Изучаем Модуль Конфигурируемых Логических Ячеек (CLC), используем Таймера и модули ШИМ.

Аппаратно (без участия программы) формируем сигнал вида

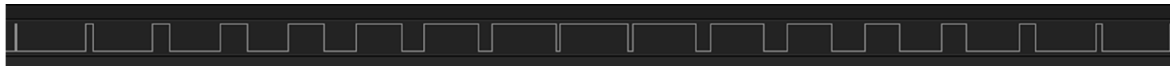
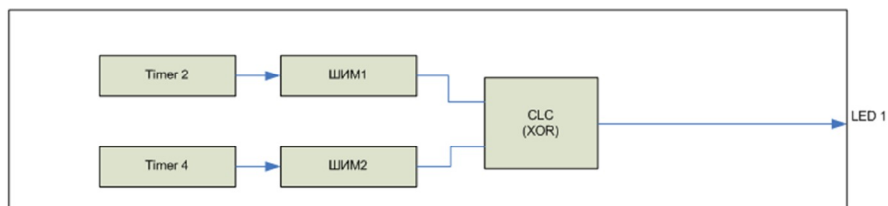


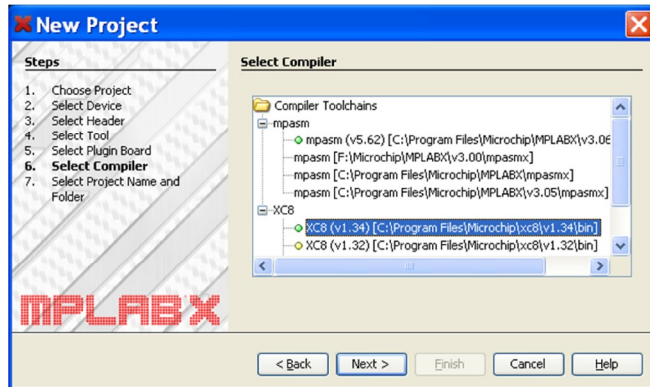
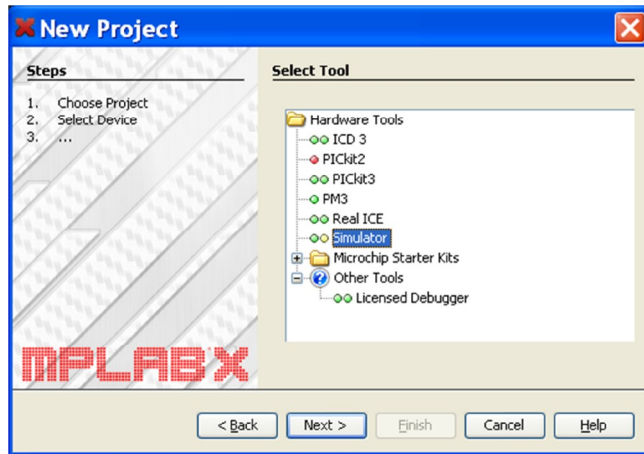
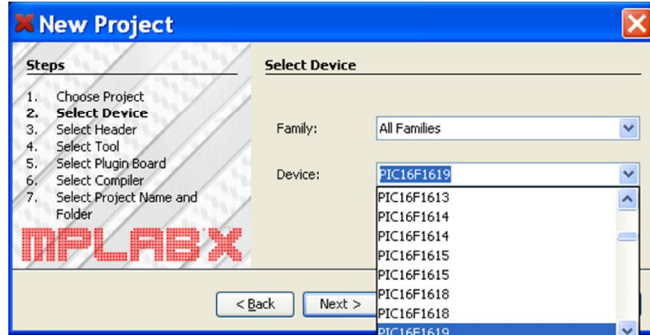
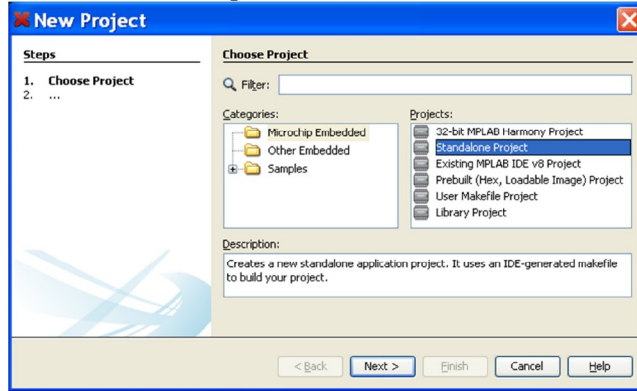
Схема:

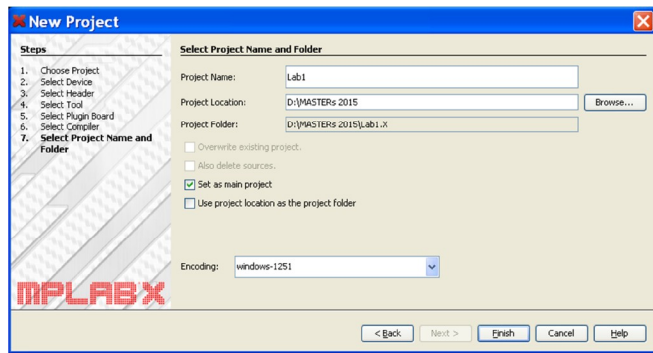


1. Запускаем MPLAB X IDE

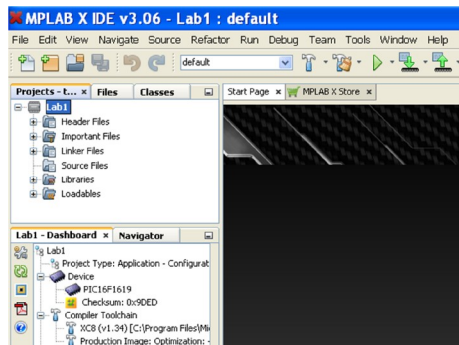


## 2. Создаем проект в MPLAB X IDE

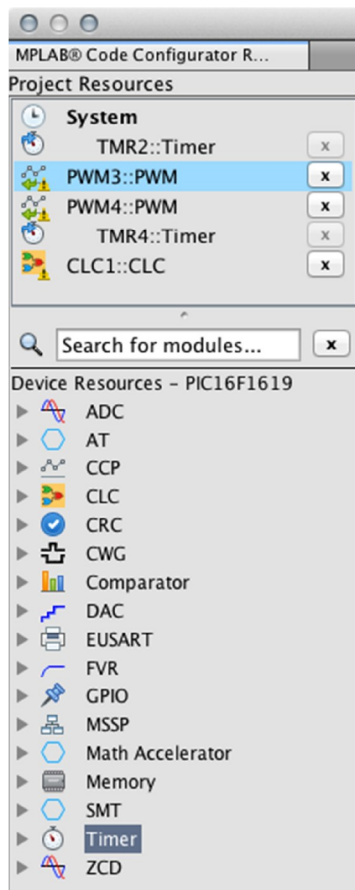
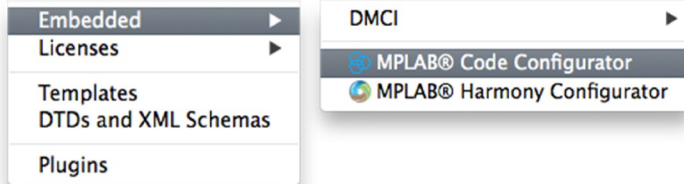




Получили пустой (пока проект)



Запускаем MCC (Mplab Code Configurator):



Нам понадобится конфигурировать:

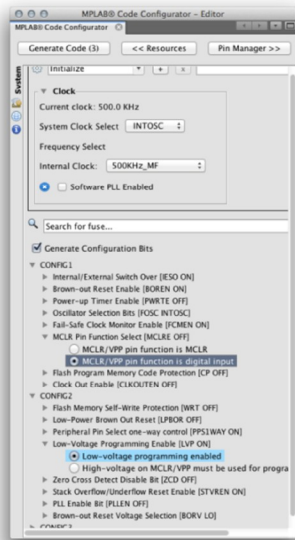
System: Источник тактирования, частота внутреннего генератора, биты конфигурации

Таймера: TMR2, TMR4

Модули ШИМ: PWM3, PWM4

Модули конфигурируемых логических ячеек CLC

Прим.: Двойной клик по ресурсу микроконтроллера (Device Resources) переводит ресурс в проект (Project Resources), т.е. периферию из *доступной* переводит в ранг *используемой* в проекте.



## System

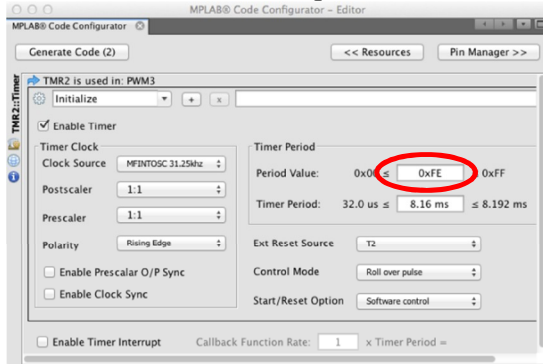
Внутренний генератор INTOSC

Тактовая частота 500кГц (не принципиально)

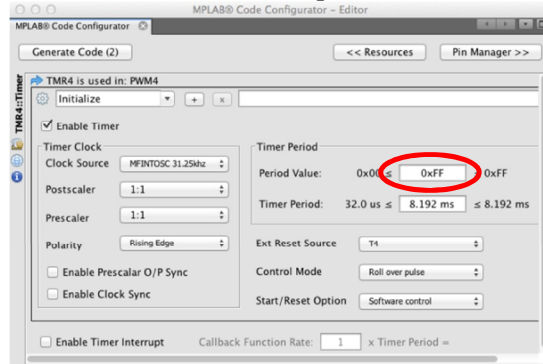
Биты конфигурации:

- MCLR – возможно использовать как I/O
- Low-voltage programming Enabled

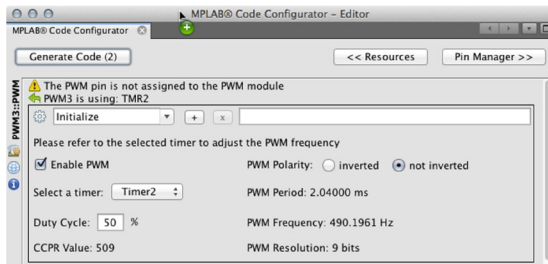
## Таймер 2



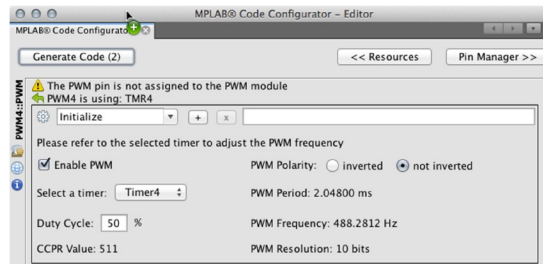
## Таймер 4



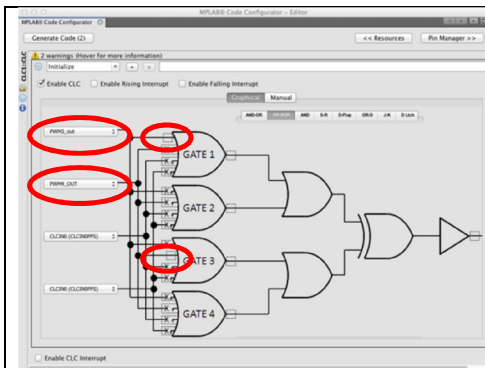
## ШИМ 3



## ШИМ 4



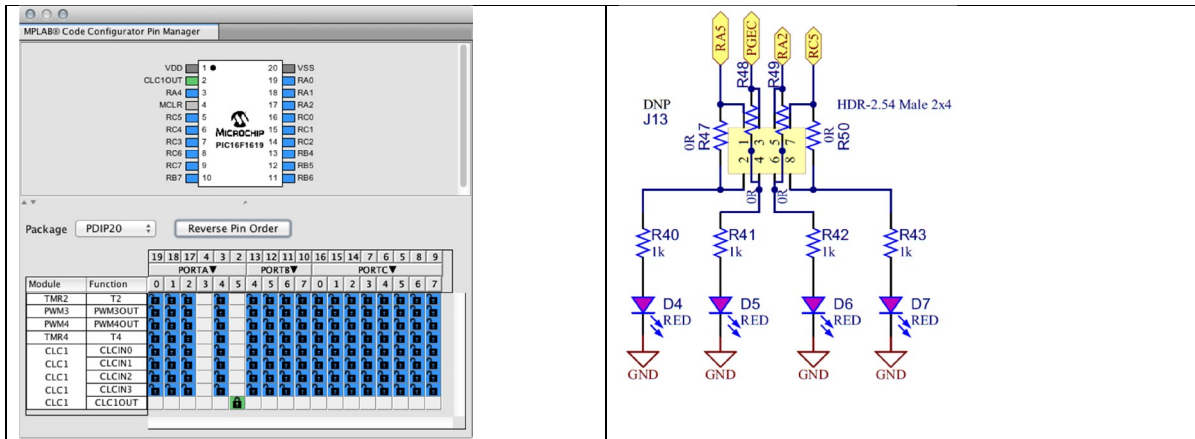
Сигналы ШИМ с разной частотой подаем на один из CLC (в конфигурации XOR)



## CLC элемент OR-XOR

Один вход PWM3\_out  
Второй вход PWM4\_out

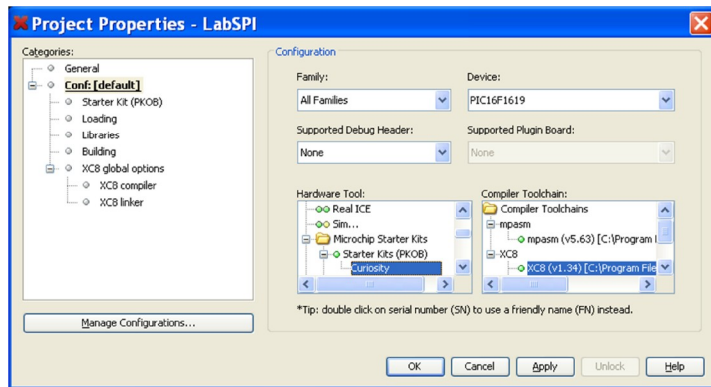
Конфигурируем выход CLC на один из светодиодов (порты микроконтроллера RA5 или RA2 или RC5)



Генерируем код



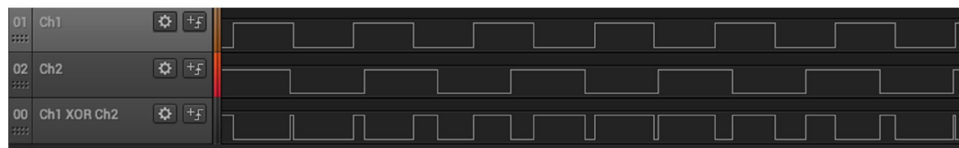
В свойствах проекта в качестве отладочной платы выбрать Curiosity



Программируем микроконтроллер.



На выбранном светодиоде должны видеть периодическое изменение яркости.



## Лабораторная работа №2

Изучение модуля вычисления CRC, таймера измерения сигналов SMT, оконного сторожевого таймера WWDT.

Пример использует функции библиотеки CLASS B (стандарт IEC 60730) программного сканирования памяти и вычисления CRC.

Современные бытовые приборы как правило содержат электронику, к которой, в свою очередь, предъявляются требования по безопасности. Стандарты типа IEC 60335 используются производителями как основа для обеспечения безопасности разработок на системном уровне. Стандарт IEC 60730, на который ссылается IEC 60335, рассматривает безопасность блоков электронного управления в бытовой технике. В настоящее время обеспечение стандарта IEC 60730 обязательно для бытовой техники, продаваемой в Европе. Приложение H стандарта IEC 60730 определяет требования для электронного управления и меры по выявлению ошибок.

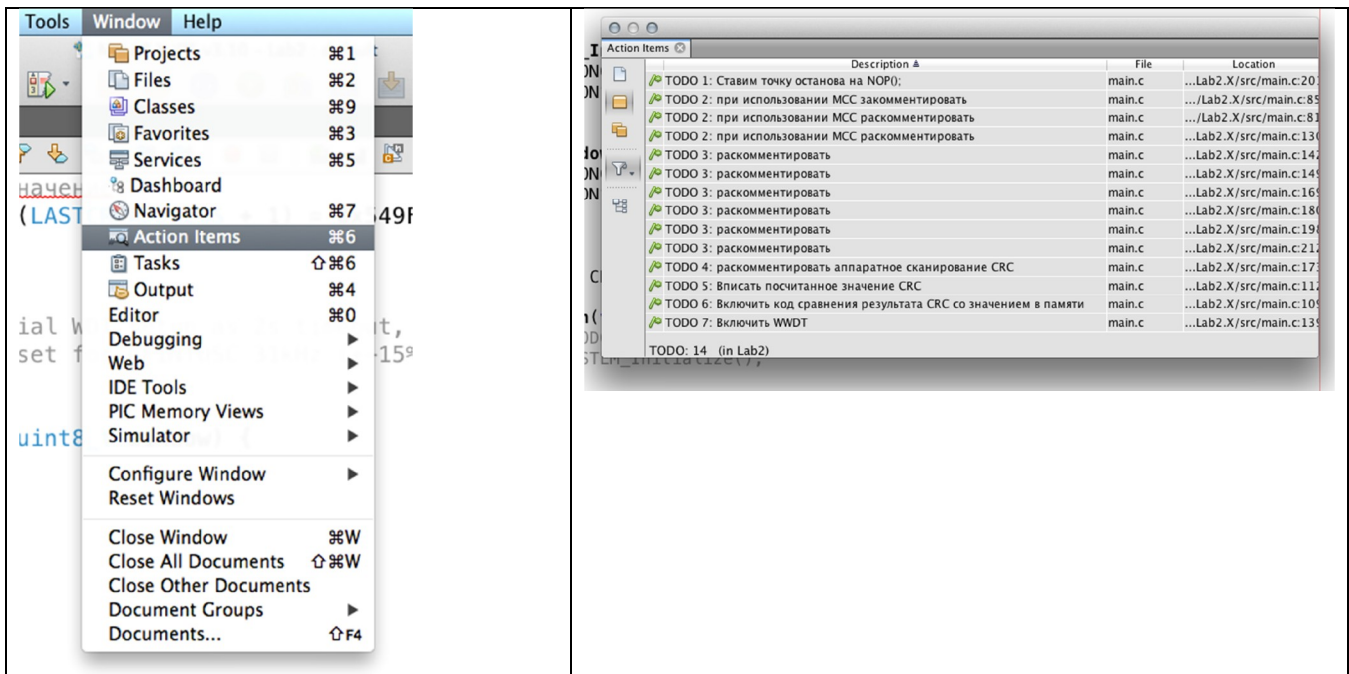
В Microchip разработали библиотеки низкоуровневых функций которые упрощают выполнение требований стандарта IEC 60730 для приборов с требованиями по безопасности «класса Б» (Class B). Такие библиотеки созданы и сертифицированы для каждого семейства микроконтроллеров (PIC16, PIC18, PIC24, dsPIC DSC и PIC32).

### ШАГ 1.

Открываем проект Lab2

Пока проект не использует MCC.

Открываем Window -> Action Items (Ctrl + 6). Тут по шагам показано где и что делать:



TODO 1. Ставим точку останова (BP).

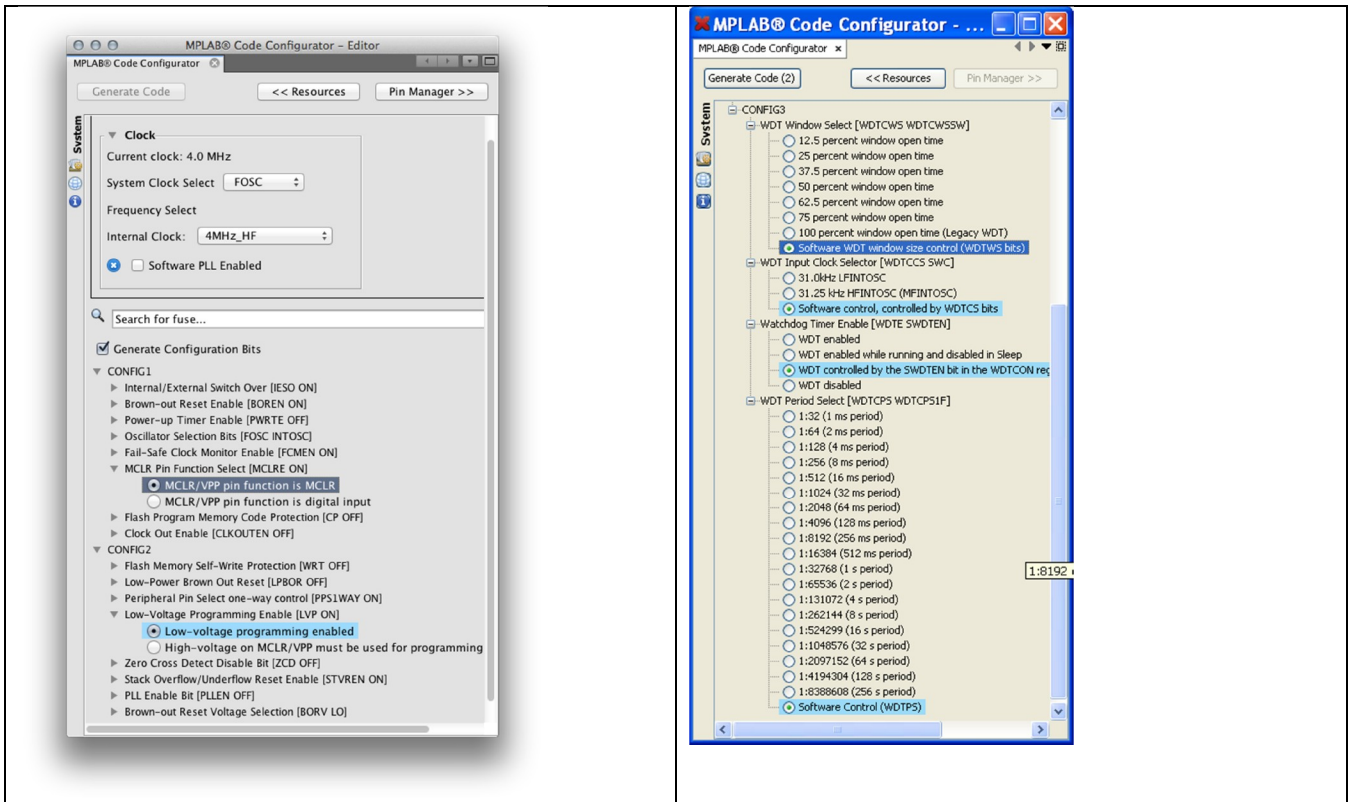
Компилируем, запускаем debug.



Видим, что программа работает (останавливается на точке останова).

## ШАГ 2. MCC

Запускаем MCC, конфигурируем тактовую частоту (4МГц), генерируем код.

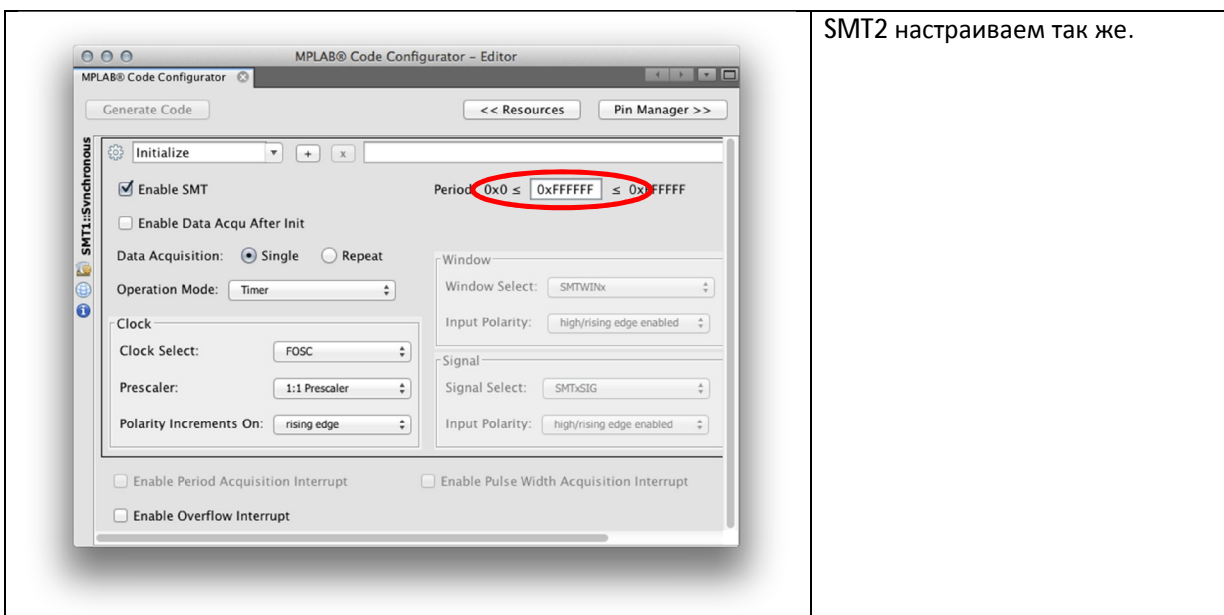


Убираем комментарии со строк отмеченных в TODO 2 (используем функции, сгенерированных в MCC – инициализация, слово конфигурации).

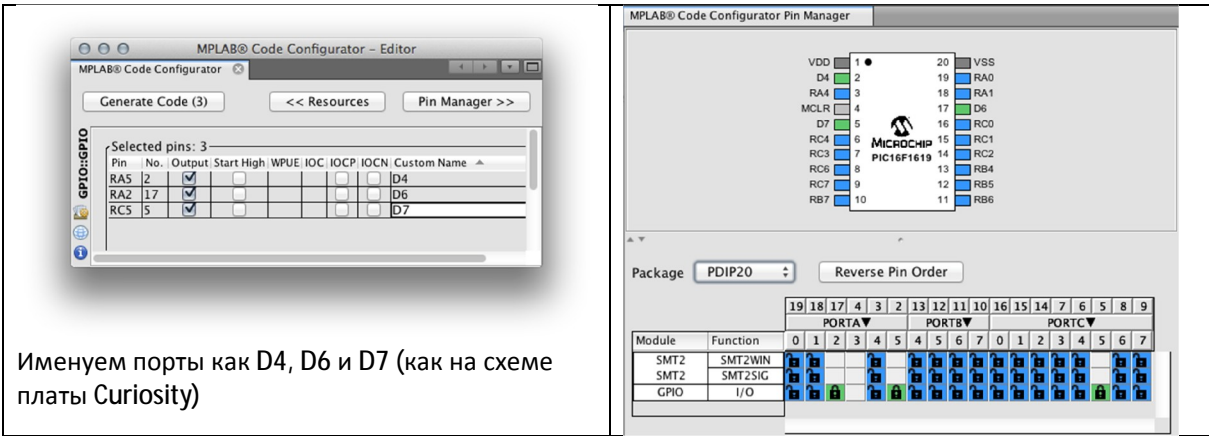
Запускаем отладку, видим, что код работает (останавливается на Точке останова).

## ШАГ 3. Конфигурируем GPIO (светодиоды), таймер SMT

Настраиваем SMT, GPIO



SMT2 настраиваем так же.

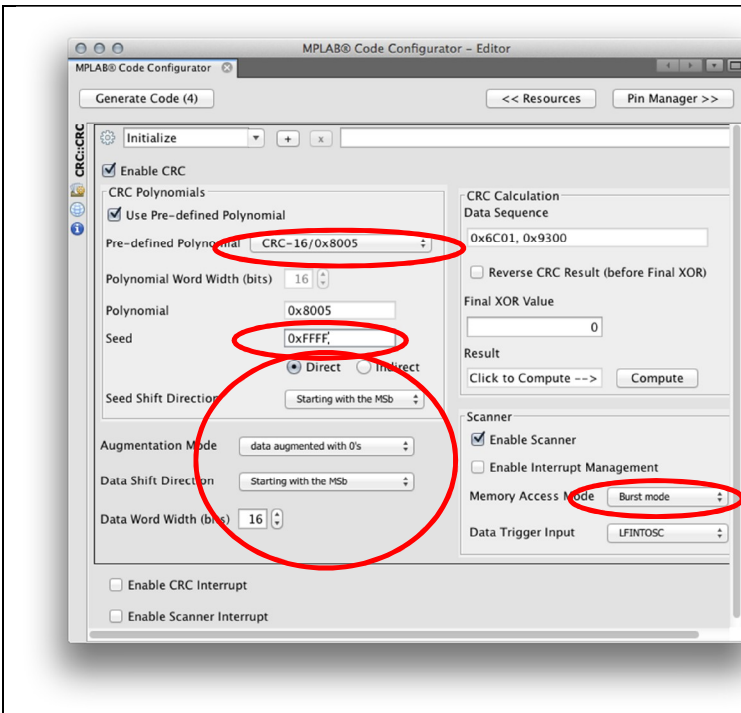


Убрать комментарии отмеченных в TODO 3 (использование функций SMT таймеров, индикация светодиодов).  
 Запускаем, смотрим время выполнения программного сканирования Flash и вычисления CRC.

Name	Type	Address	Value	Decimal
<input checked="" type="checkbox"/> SMT1TMR	SFR	0xD8C	0x39813C	3768636
<input checked="" type="checkbox"/> SMT2TMR	SFR	0xD9E	0x020170	131440
<input checked="" type="checkbox"/> CRCACC	SFR	0x793	0x496E	18798
<input checked="" type="checkbox"/> CRC_libraryResult	unsigned int	0x20	0x496E	18798
<Enter new watch>				

#### ШАГ 4. Аппаратное CRC

Настраиваем CRC.



Замечание.

Для MCC Version: 2.25.2 и PIC16F1619 есть «глюк» неправильной подстановки Seed в код.

После генерации кода откройте файл `\MASTERS\Lab2.X\mcc_generated_files\crc.c` и в строке 96 исправьте seed на `seed = 0xFFFF;`



Выполняем TODO 4 (запускаем аппаратное сканирование памяти и вычисление CRC)

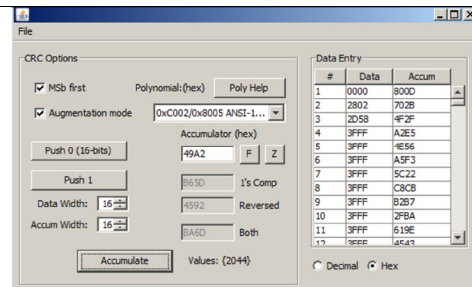
Запускаем отладчик, смотрим время выполнения аппаратного сканирования Flash и вычисления CRC. Сравниваем с время SMT1 (программное сканирование и CRC) и SMT2 (аппаратное сканирование и CRC). Разница почти в 30 раз.

Name	Type	Address	Value	Decimal
SMT1TMR	SFR	0xD8C	0x39813C	3768636
SMT2TMR	SFR	0xD9E	0x020170	131440
CRCACC	SFR	0x793	0x496E	18798
CRC_libraryResult	unsigned int	0x20	0x496E	18798

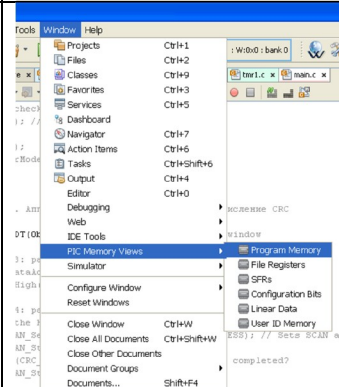
Выполняем TODO 6 (включаем проверку вычисления CRC с сохраненным значением в памяти).

### ШАГ 5. Вычисляем CRC утилитой CRC Calculator

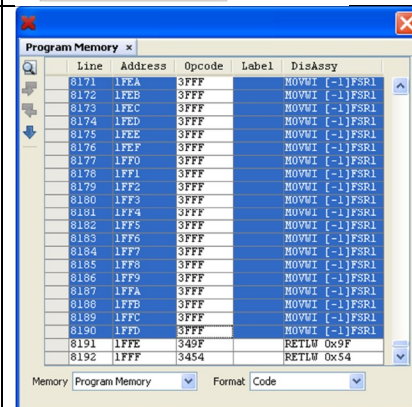
Запускаем CRC Calculator



В MPLAB X IDE открываем окно отображения памяти программ



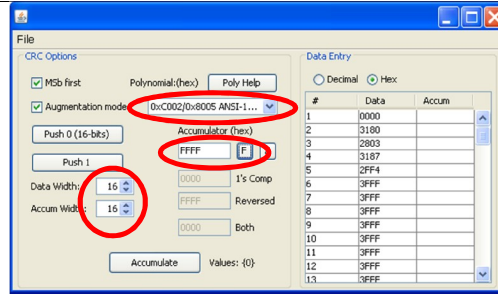
Копируем (Ctrl+C) с 0-го адреса до адреса 0x1FFD (не трогаем 2 последние ячейки – там будем хранить CRC).  
Сохраняем в текстовый файл (File -> new file и т.д).



Открываем CRC Calculator  
File -> Export File – экспортируем наш файл с дампом памяти.

Выбираем полином ANSI-16  
Ширина Аккумулятора и Данных = 16 бит  
Начальное значение аккумулятора FFFF

Вычисляем CRC (Accumulate)



Вписываем результат в TODO 5 (значение CRC памяти сохраняем в 2-х последних ячейках Flash памяти).

Запускаем отладчик. Результат вычисления CRC в утилите, программно и аппаратно должны совпадать.

### ШАГ 6. WWDT



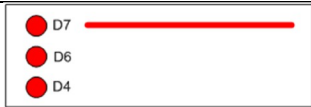

Можем закомментировать TODO 6 , выполняем задание TODO 7 (включаем WWDT).

Запускаем. Работает.

Изменяем тактовую частоту на 8МГц, или можно закомментировать программное или аппаратное CRC.  
Запускаем. Срабатывает WWDT (сброс сторожевого таймера произошел слишком рано)

Выводы. Аппаратное вычисление CRC в десятки раз быстрее, занимает меньше памяти.

Оконный сторожевой таймер срабатывает если его сброс происходит слишком рано или слишком поздно.  
Позволяет более точно контролировать ход выполнения кода.

Диагностика хода выполнения программы с включенным WWDT и проверкой результата CRC (см. TODO 6)	Свечение светодиодов
	D7 ~1сек (программное сканирование, CRC) D6 ~0.3сек (аппаратное сканирование, CRC) D4 2сек (while(1);)
	Правильные CRC (если проверяем) D7 ~1сек (программное сканирование, CRC) D6 ~0.3сек (аппаратное сканирование, CRC) D7 и D6 2сек
	Неправильная CRC (если проверяем) D7 (программное сканирование, CRC)
	WWDT сбрасывает МК на программном CRC D7 ~1сек (программное сканирование, CRC) D6 <0.3сек (аппаратное сканирование, CRC) WWDT сбрасывает МК на аппаратном CRC

## Лабораторная работа №3

Изучение математического акселератора на примере вычисления цифрового фильтра с Конечной Импульсной Характеристикой (КИХ, FIR).

КИХ фильтр описывается формулой:

$$y(n) = \sum_{k=0}^{N-1} (h_k(n) \times x(n - k))$$

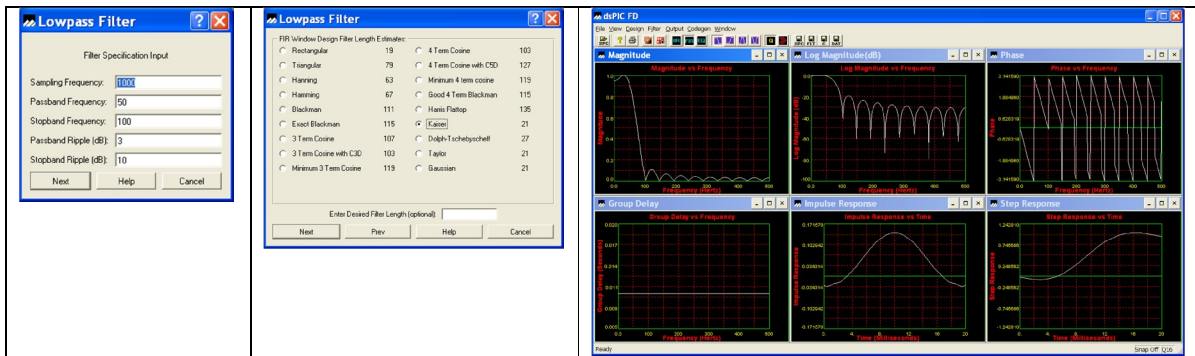
, где:

$y(n)$  – результат фильтрации,

$x(n)$  – входной отсчет,

$h$  – коэффициенты фильтра (импульсная характеристика фильтра для  $N$ -отсчетов).

Не вдаваясь в подробности, будем считать, что у нас есть коэффициенты фильтра, полученные например из dsPIC Filter Design.



### ШАГ 1.

Итого, для каждого входного отсчета нужно провести  $N$  умножений с накоплением.

Конфигурируем MathACC, SMT2

The figure shows two screenshots of the MPLAB Code Configurator. The first screenshot shows the 'MATHACC:Default' configuration with 'Enable Math Accelerator' checked, 'Mod' set to 'Add/Multiply', 'Operation' set to 'only Multiply', and 'Output' set to 'accumulated'. The second screenshot shows the 'SMT2::Synchronous' configuration with 'Enable SMT' checked and 'Period' set to '0x00000000' to '0xFFFFFFFF'.

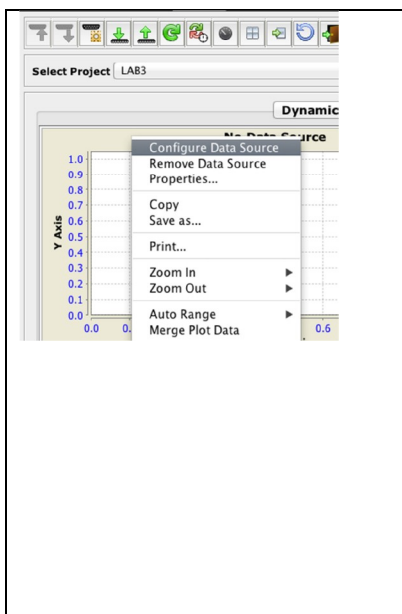
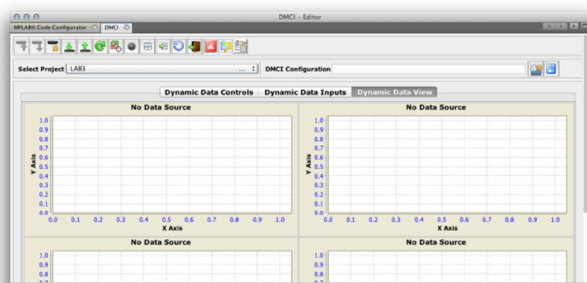
Только умножение, знаковые входные данные, результат с накоплением

Таймер SMT может помочь в определении времени вычислений

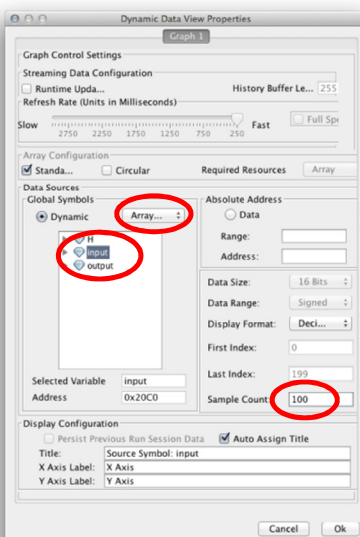
Ставим точку останова после фильтра (TODO 1).

## Шаг 2. Настраиваем DMCI

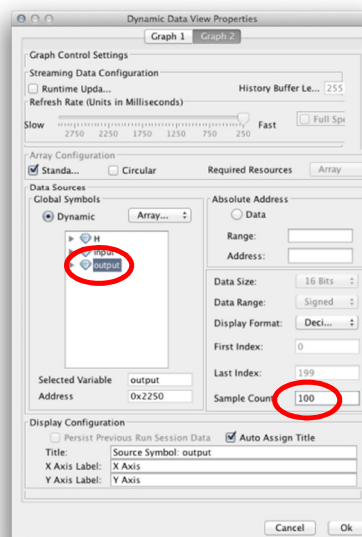
DMCI как и MCC устанавливается через меню Плагинов.



Правый клик на окне

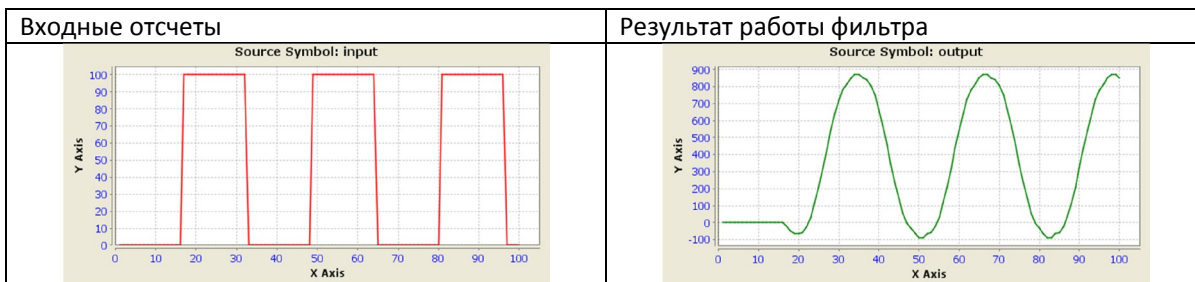


Входной буфер



Выходной буфер

Должны получить следующий результат:



Замечания.

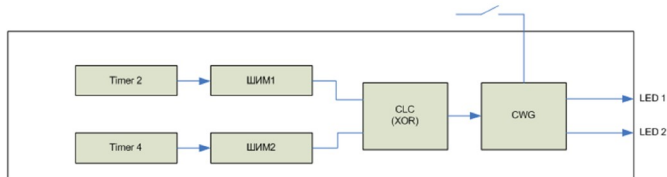
Данный пример сделан в ознакомительных целях. По правильному, нужно как минимум нормировать коэффициенты фильтра.

Пример использует большой буфер, так как в DMCI мы хотим увидеть «большую» картинку. В реальной программе может быть целесообразно хранить только последние N входных отсчетов (N – число отсчетов импульсной характеристики фильтра).

## J Домашнее задание J

### Лабораторная работа №1 (домашнее задание 1)

Изучение модуля формирования комплементарных сигналов (Complementary Waveform Generator, CWG). Делаем то же самое что в Лаб.1, но с 2-я каналами в противофазе с использованием CWG.



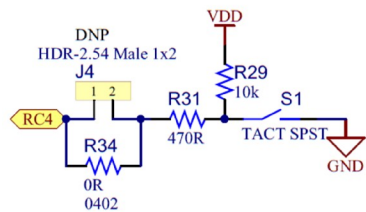
Добавляем CWG в проект.

Один выход прямой, второй инверсный

Автовыключение и Авторестарт (см.ниже)

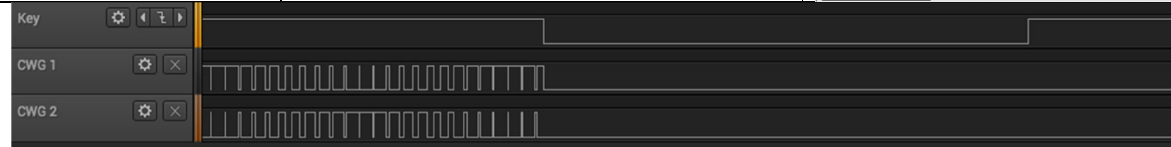
Внешний сигнал может аппаратно отключать выходы CWG и переводить их в predetermined состояние.

Мы можем задействовать кнопку S1.

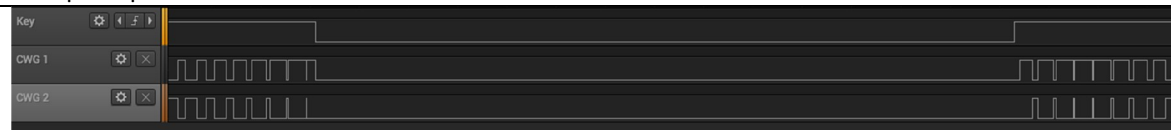


MPLAB® Code Configurator Pin Manager

Module	Function	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
TMR2	T2																
PWM3	PWM3OUT																
PWM4	PWM4OUT																
TMR4	T4																
CLC1	CLCIN0																
CLC1	CLCIN1																
CLC1	CLCIN2																
CLC1	CLCIN3																
CLC1	CLCOUT																
CWG	CWGIN																
CWG	CWID																
CWG	CWGLIC																
CWG	CWG1B																
CWG	CWG1A																



Авторестарт выключен



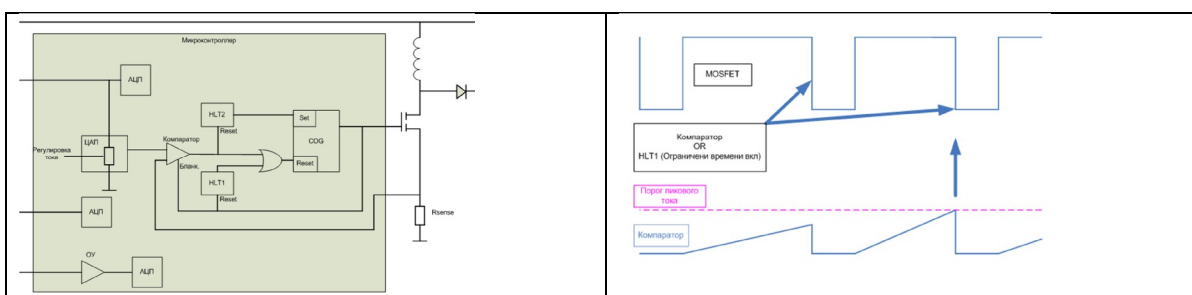
Авторестарт включен

## Лабораторная работа №1 (домашнее задание 2)

Изучение таймера ограничения (Hardware Limit Timer, HLT).

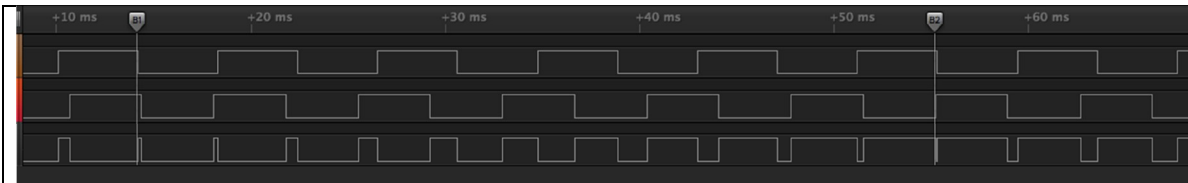
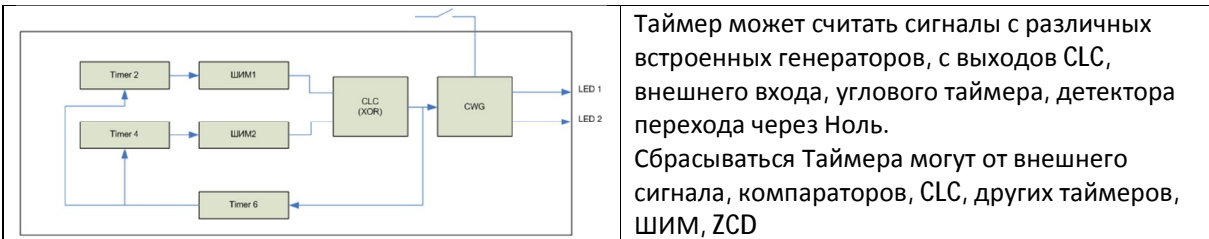
Функция Таймера ограничений встроена в Timer2, Timer4, Timer6 и т.д. или в некоторых МК присутствует как отдельный таймер. Позволяет ограничить время работы периферии, если в ожидаемое время не пришло какое-либо событие.

Пример. На СІР периферии микроконтроллера стоим импульсный источник питания.



Силовой ключ открывается по таймеру. При срабатывании компаратора – закрывается. Но необходимо контролировать время открытого состояния MOSFET и не допускать, чтобы оно было больше заданного времени (например, если нет сигнала с компаратора). Эту функцию может обеспечить HLT таймер.

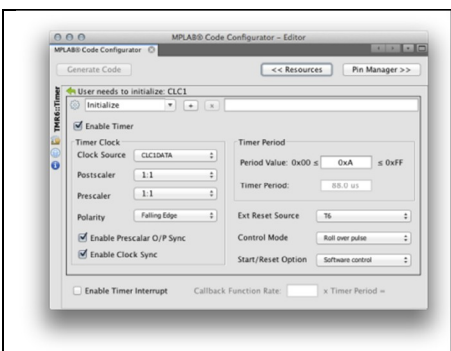
В нашем примере (Лабораторная работа 1) дополнительный таймер (Timer 6) будет считать импульсы CLC и сбрасывать используемые для формирования ШИМ таймера (Timer 2 и Timer 4).



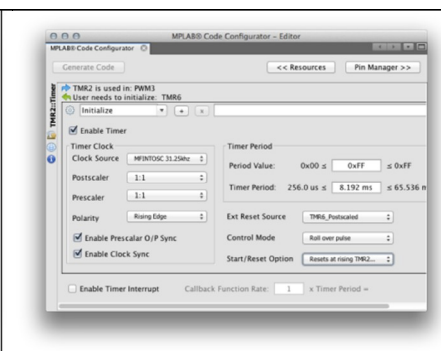
Исходная форма сигналов без ограничения.



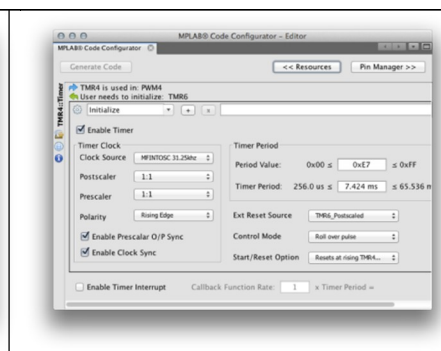
Таймер 6 считает 10 импульсов и сбрасывает Таймер 2 и Таймер 4



TMR6 считает импульсы CLC



TMR2 сбрасывается от TMR6



TMR4 сбрасывается от TMR6

## Лабораторная работа №2 (домашнее задание)

Посчитайте CRC данных из буфера в ОЗУ.

Примеры смотрите в сгенерированном MCC файле crc.h

Проверить результат можно с помощью утилиты CRC Calculator.

## Лабораторная работа №3 (домашнее задание)

В лабораторной работе №3 используются сгенерированные с помощью MCC функции использования математического акселератора.

Они не всегда оптимальны. Например, мы вычисляем функцию вида  $y += H * x$  с автоматическим накоплением результата. Используем функцию:

```
uint32_t MATHACC_MultiplicationResultGet(uint16_t b, uint16_t c)
```

Т.е. на каждом вызове используемая функция возвращает результат умножения. Так как у нас используется автонакопление, то нет необходимости в возврате результата, следует лишь по завершению цикла (перемножение входного отсчета на все коэффициенты фильтра) взять результат накопления.

Поэтому можно переписать функцию так, чтобы она не возвращала результат. Это даст выигрыш во времени выполнения.

Функция MATHACC\_MultiplicationResultGet берет входные данные, заносит их в регистры MathAccelerator, запускает вычислитель и ждет окончания результата. Здесь тоже можно несколько оптимизировать – проверить занятость ускорителя сразу при входе в функцию и если Мат.Ускоритель свободен, то занести новые данные и запустить вычисление.

Сравните:

Оригинальный код	Оптимизированный код под FIR
<pre>uint32_t MATHACC_MultiplicationResultGet(uint16_t b, uint16_t c) {     uint32_t result = 0;      PID1K1H = (uint8_t) ((c &amp; 0xFF00) &gt;&gt; 8);     PID1K1L = (uint8_t) (c &amp; 0x00FF);     PID1SETH = (uint8_t) ((b &amp; 0xFF00) &gt;&gt; 8);     PID1SETL = (uint8_t) (b &amp; 0x00FF);     PID1INH = 0;     PID1INL = 0; // starts module operation      while (PID1CONbits.PID1BUSY == 1); // wait for the module to     complete      result = PID1OUTLL;     result = (result   ((uint32_t) PID1OUTLH &lt;&lt; 8)) &amp; 0x0000FFFF;     result = (result   ((uint32_t) PID1OUTH &lt;&lt; 16)) &amp; 0x00FFFFFF;     result = (result   ((uint32_t) PID1OUTH &lt;&lt; 24)) &amp; 0xFFFFFFFF;      return result; }</pre>	<pre>void MATHACC_Multiplication(uint16_t b, uint16_t c) {      while (PID1CONbits.PID1BUSY == 1); // wait for the module to     complete      PID1K1H = HIGH_b(c);     PID1K1L = LOW_b(c);     PID1SETH = HIGH_b(b);     PID1SETL = LOW_b(b);     PID1INH = 0;     PID1INL = 0; // starts module operation }  #define LOW_b(x) ((char*)&amp;x)[0] #define HIGH_b(x) ((char*)&amp;x)[1]</pre>