1. Разбить программу, полученную в результате выполнения 2-й лабораторной работы, на три составляющие: void Input(); void Otput(); void GreatWork(). Каждую из этих процедур реализовать по следующему шаблону – инициализация, за которой следует непрерывное выполнение инструкций внутри бесконечного цикла:

void func(){

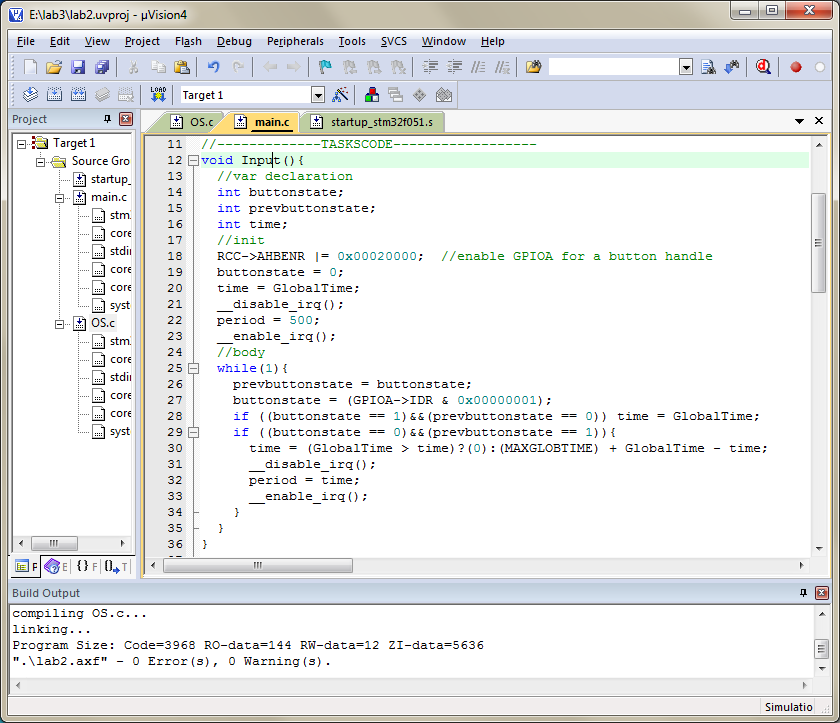
//инициализация

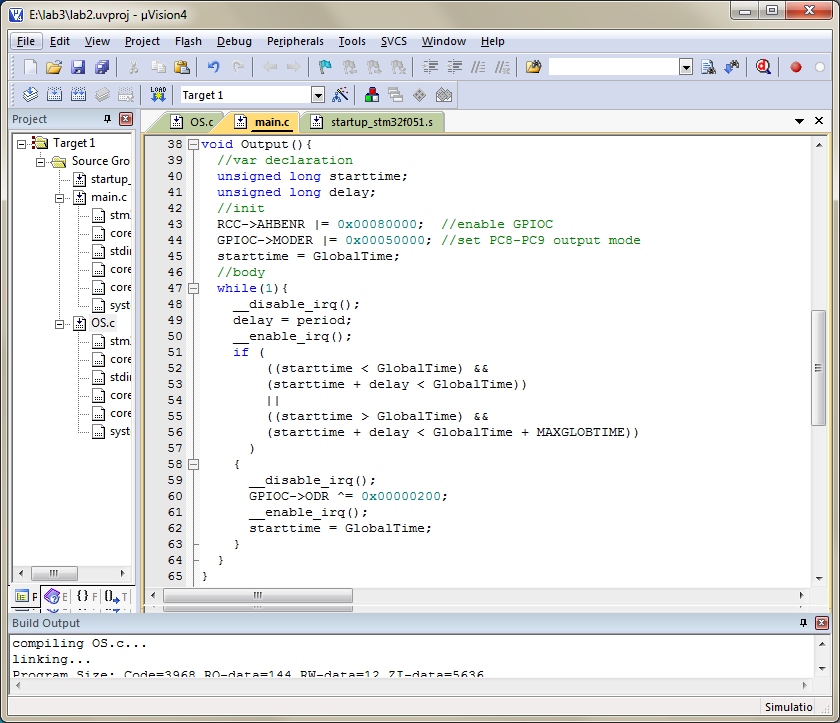
while(1){

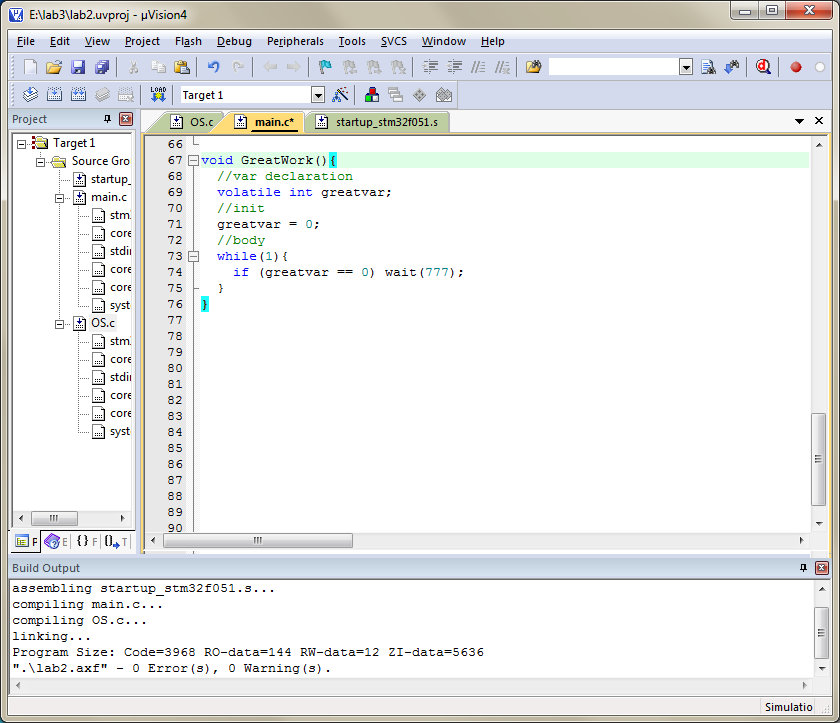
//код процедуры

}

}







Эти три задачи будут выполняться в псевдо-параллельном режиме. Каждая реализована так, как будто она одна в системе – в них нет функций переключения на другие задачи, только своя работа, бесконечно повторяющаяся в цикле.

Задача операционной системы (ОС) – периодически останавливать выполнение и переключать процессор на следующую задачу. Для этих целей мы будем использовать прерывание от таймера, в котором и будем вызывать ядро ОС.

Для того, чтобы процессор начал выполнять другу задачу, ему надо подменить контекст – заменить стек и регистры одной задачи на стек и регистры другой.

2. ??? Если просто скопировали код - нарисуйте алгоритм каждой функции



3. В файле main.c cоздать массив, в который поместить адреса на написанные процедуры, а также ключевые метки:

void \* const Tasks[] \_\_attribute\_\_((at(0x08001000))) = {

Input,

Output,

GreatWork,

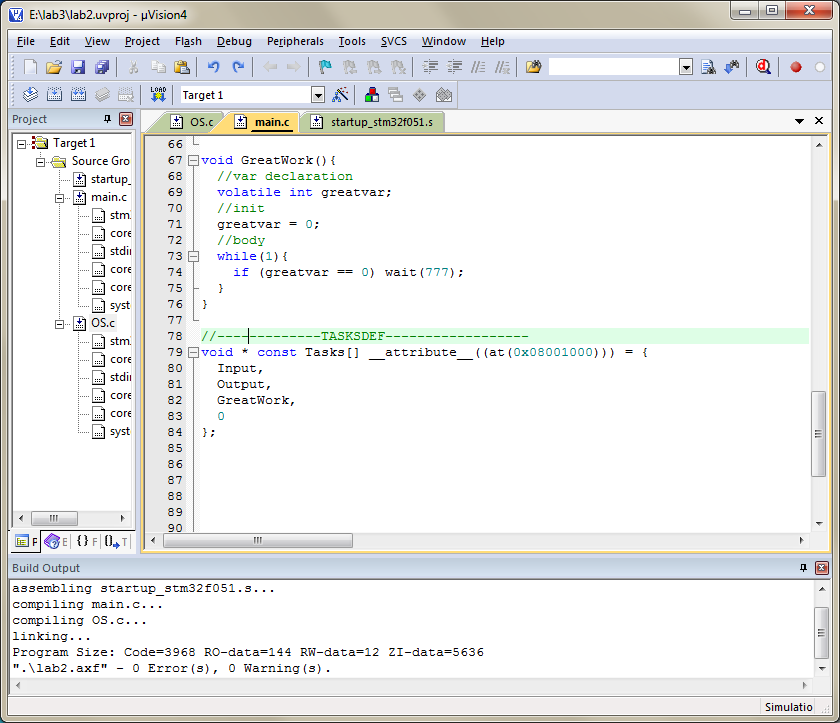
0

};

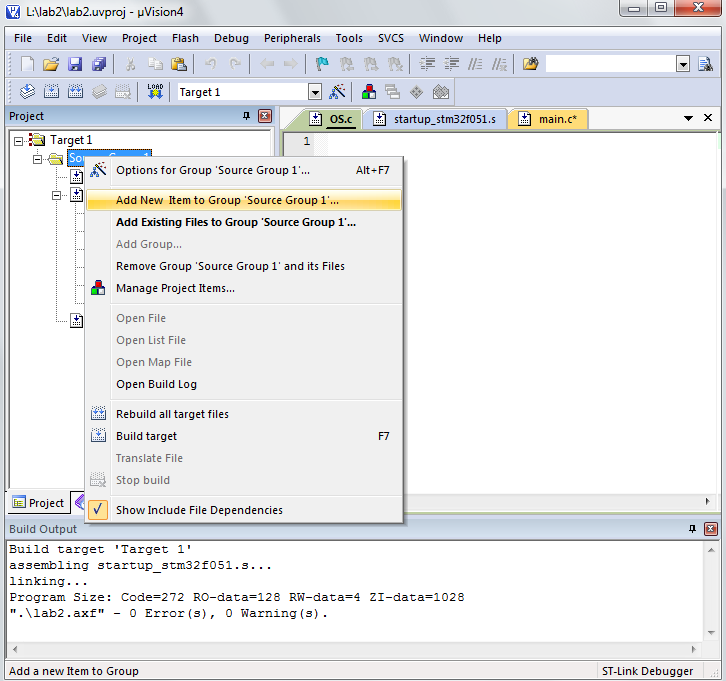
Этот список будет использоваться ОС для вызова заданий.

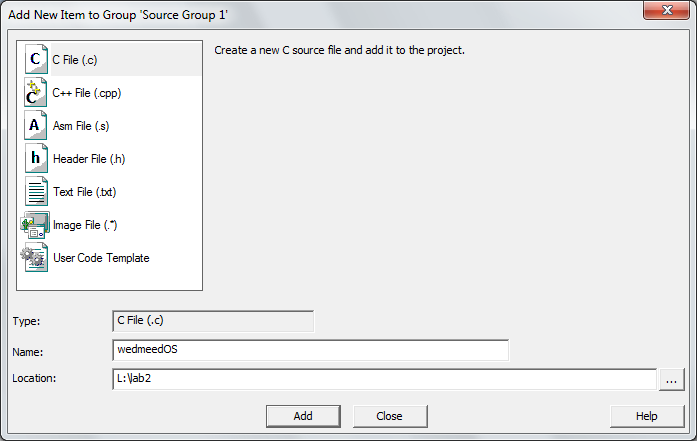
Приписка \_\_attribute\_\_((at(0x08001000))) необходима для того, чтобы структура была размещена в ПЗУ.

- ? для чего здесь ноль в конце?

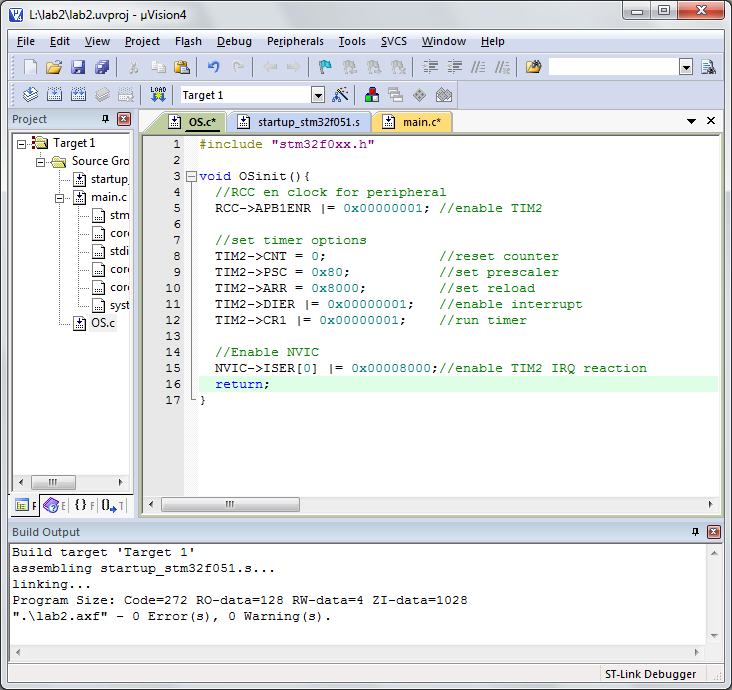


4. Создать новый файл %YourName%OS.c и подключить его к проекту.

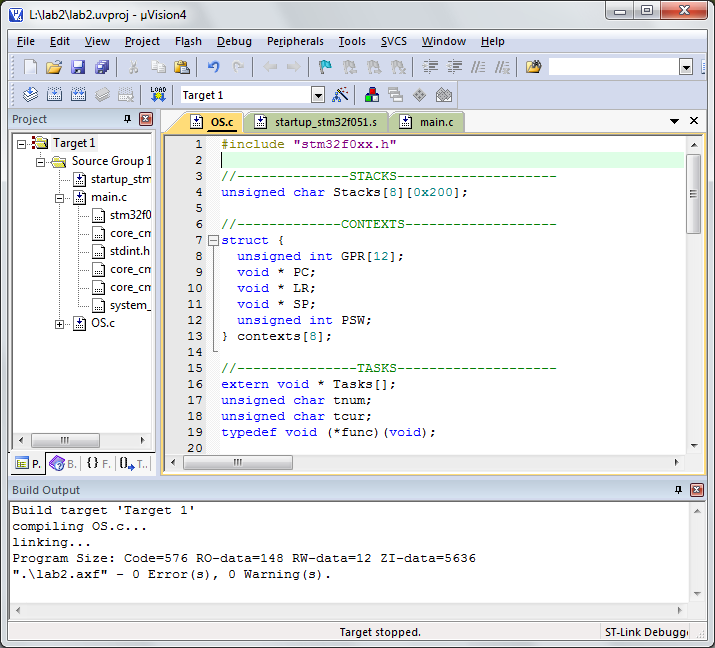




5. В %YourName%OS.c создать функцию void OSinit() и перенести в нее код из SystemInit, отвечающий за настройку таймера.



6. Импортировать в операционную систему список задач, выделить память под стек и под контекст задач

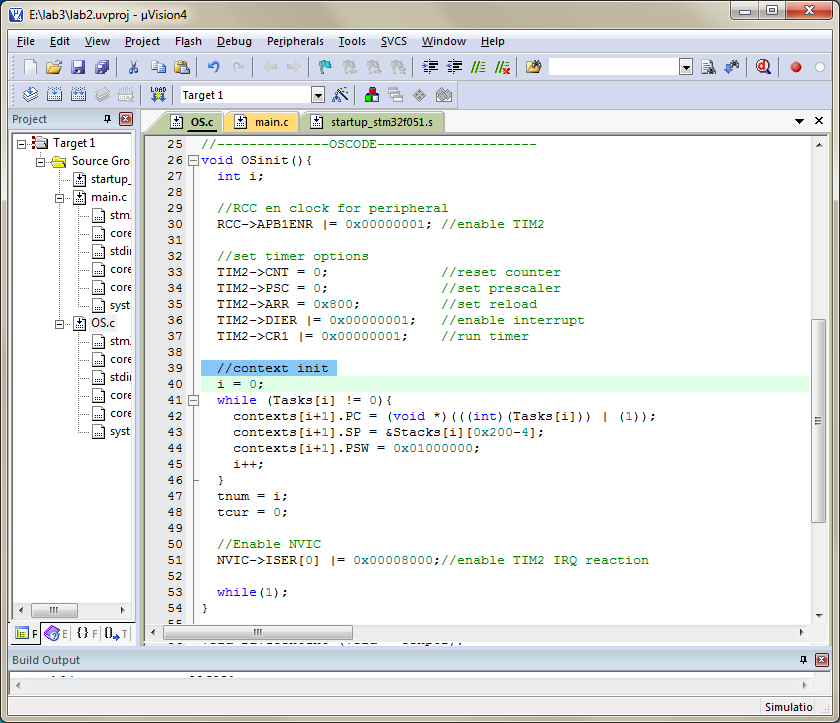


7. Добавить в OSinit() перед включением прерываний код, который для каждой задачи в списке Tasks:

- выделит память под стек;

- запишет контекст по умолчанию;

- настроит значения для PSW.

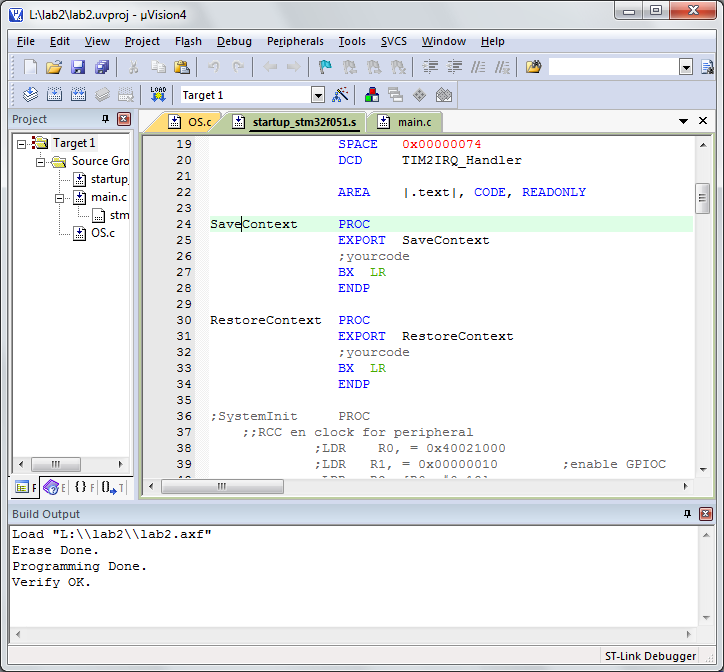


8. В startup\_\*.s реализовать функции сохранения и восстановления контекста по шаблонам

void SaveContext (\* void conptr);

void RestoreContext (\* void conptr);

в функции будет передаваться указатель на структуру с сохраненным контекстом. Функции должны работать с этой структурой, синхронизируя ее поля с регистрами процессора и стеком.



- Информация о том, куда идет сохранение регистров при прерывании, можно найти в arm\_architecture\_v7m\_reference\_manual\_errata\_markup\_1\_0.pdf, в разделе **Exception entry behavior**, а также экспериментально – с помощью отладчика.

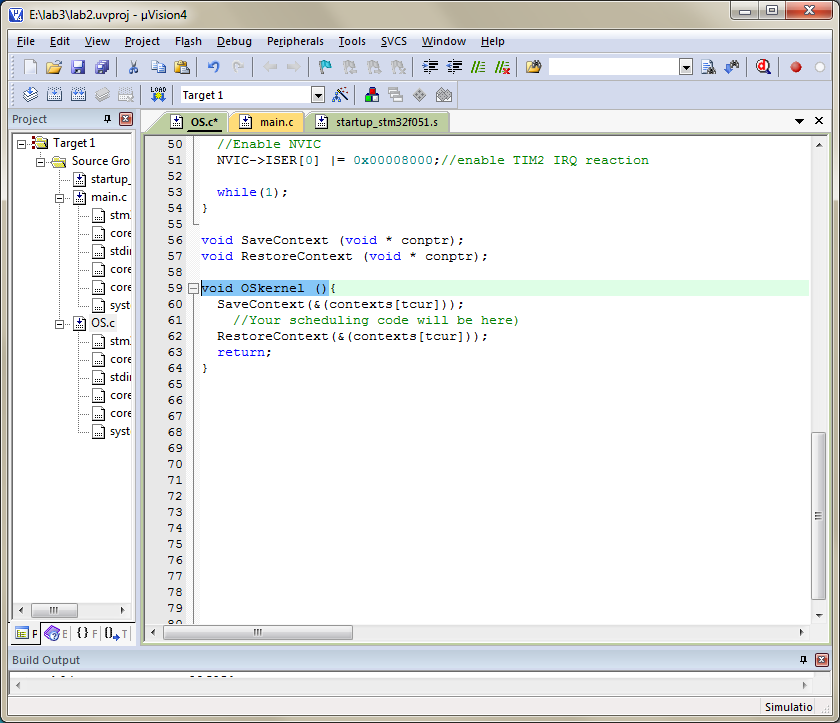
Состав контекста можно подглядеть в структуре для контекста из шага 6.

9. В %YourName%OS.c реализовать функцию void OSkernel (), которая будет

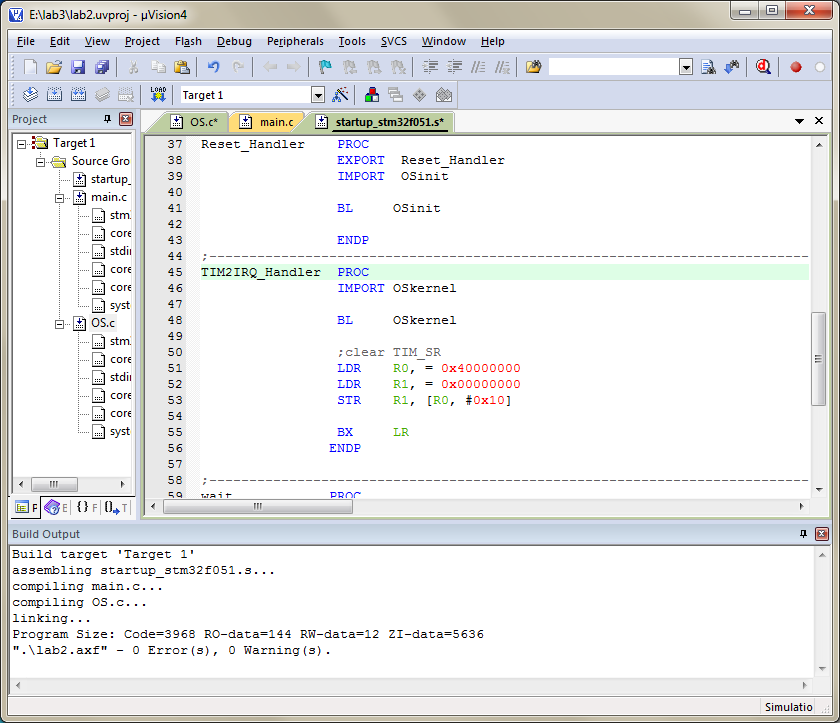
- вызывать функцию сохранения контекста для текущей задачи;

- реализовывать алгоритм планирования Round-Robin;

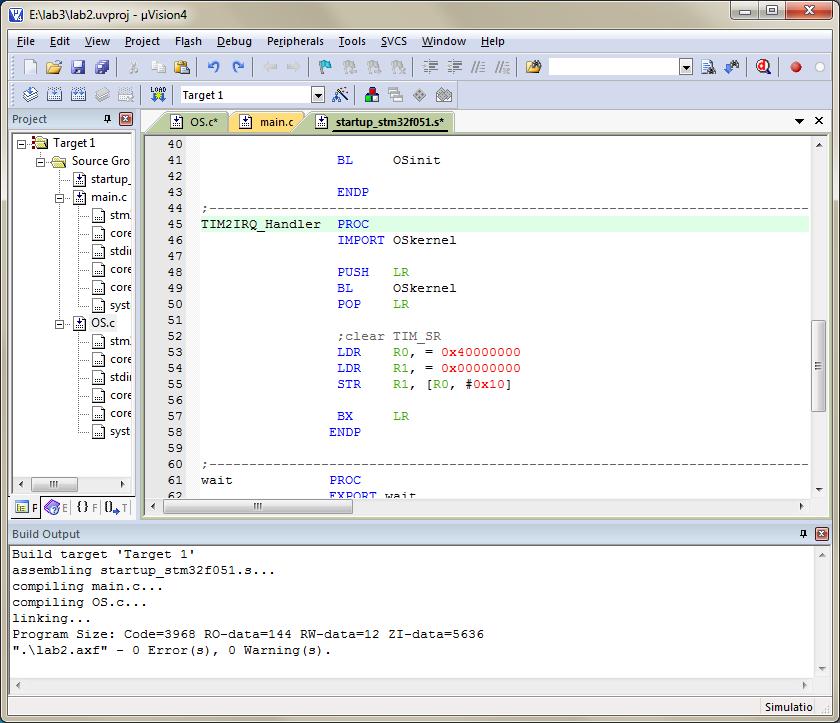
- вызывать функцию восстановления контекста для выбранной задачи.



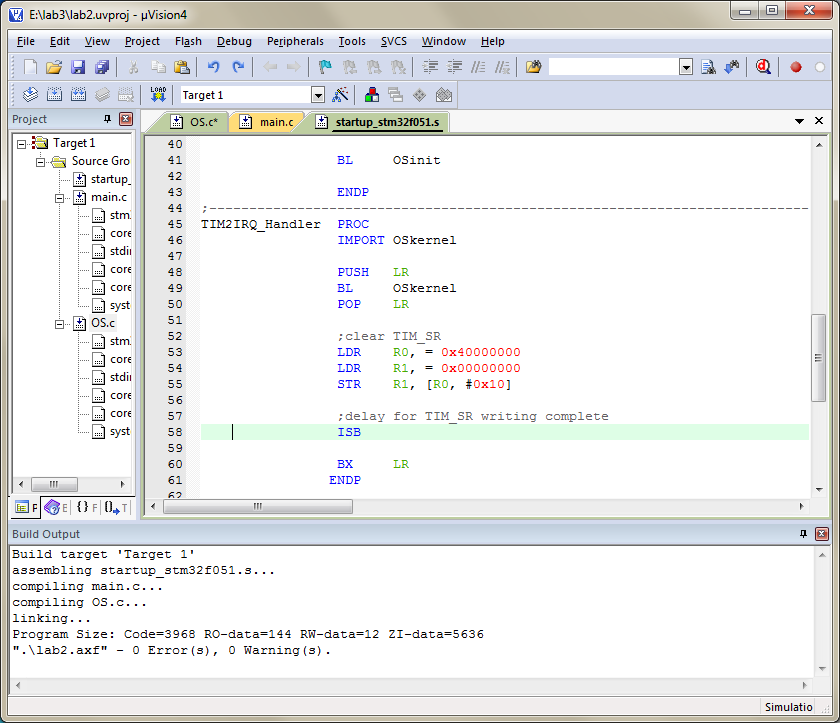
10. Убрать из обработчика прерывания весь лишний код (повторное сохранение регистров в стек и моргание светодиодом). Добавить в обработчик прерывания от таймера вызов OSkernel (), а в ResetHandler – вызов OSinit().



- для того, чтобы выйти из прерывания по "BX LR", регистр LR нужно сохранить (иначе BL OSkernel перезатрет его). Это можно сделать через стек:

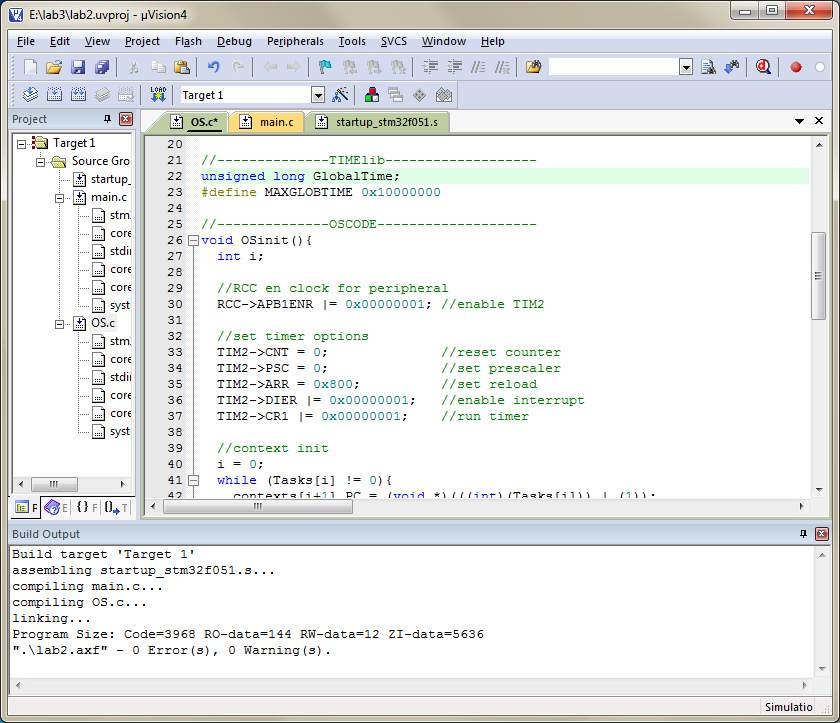


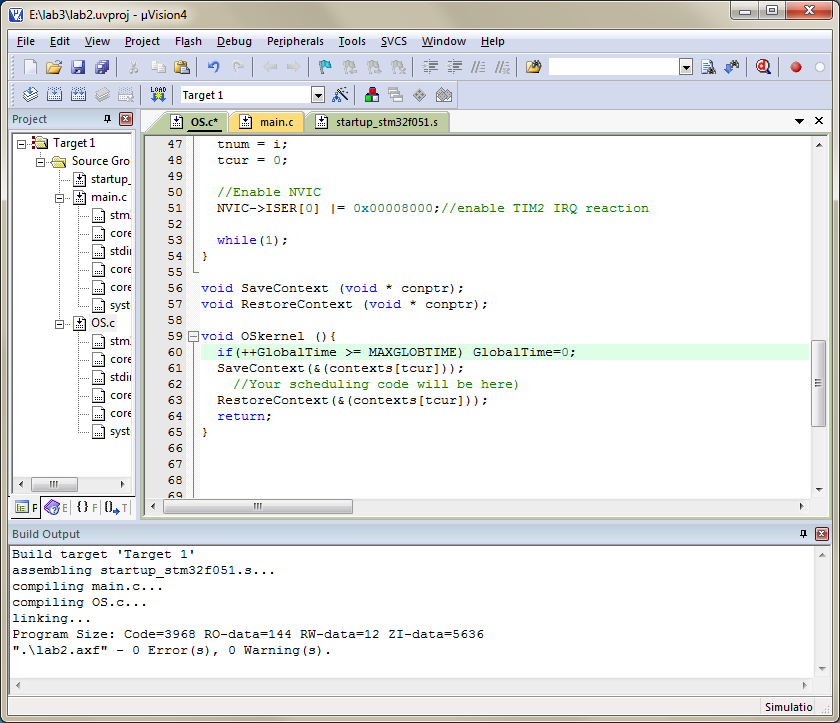
-если мы удалили POP пред BX LR в прерывании, то, из-за наличия в процессоре конвейера, может возникнуть такая ситуация, когда запись в регистр TIM\_SR еще не завершена, а процессор уже вышел из прерывания (запись в память – долгая операция, последние этапы которой выполняются в фоновом режиме конвейером в то время). Т.к. при выходе из прерывания его флаг не сброшен, то процессор опят попадет в это же прерывание. Для избежания такой ситуации перед выходом надо выполнить инструкцию ISB – синхронизатор конвейера.



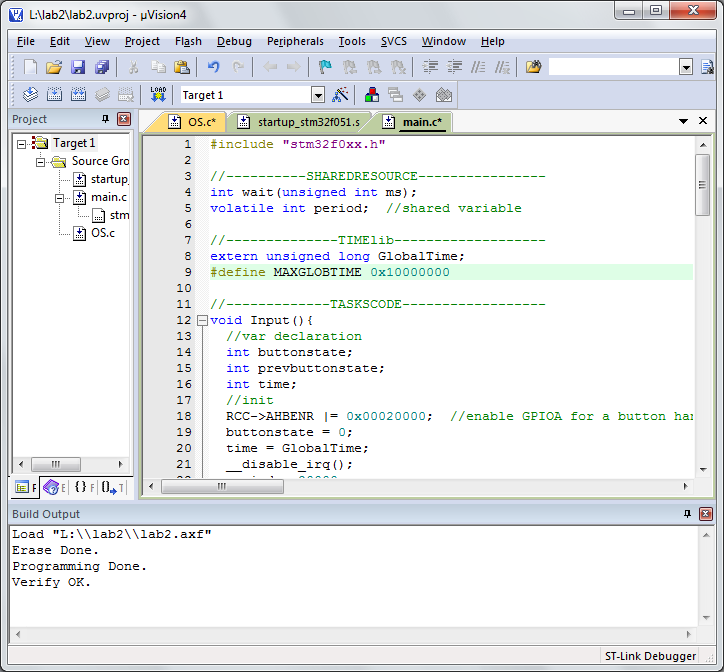
Сейчас, фактически, операционная система готова к работе. Чтобы переключение происходило достаточно часто, мы должны сильно увеличить частоту таймера. Но тогда им станет неудобно пользоваться в пользовательских задачах. Поэтому нужно встроить в ОС возможность получать текущее время.

11. Уменьшить в настройках таймера TIM2->PSC до 0. Добавить в файл операционной системы переменную GlobalTime, а в OSkernel () ее инкремент при каждом вызове.



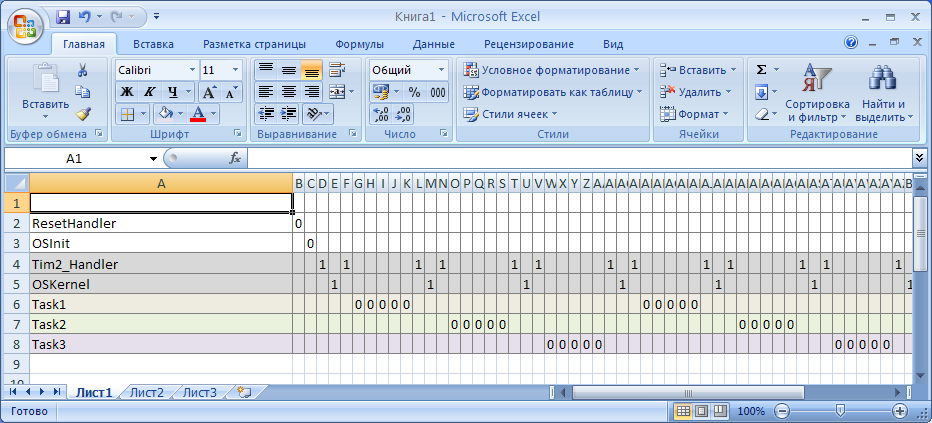


12. Импортировать определение GlobalTime в файл main.c. Обратите внимание на extern.



12. - Убедиться, что операционная система работает и в нее можно добавлять новые задачи

Диаграмма работы системы для 3х задач:



0 – нормальный режим, 1 - прерывание