Практические примеры задач на позиционирование моторов

Введение

На этом уроке Вы научитесь:

1. Пользоваться новым типом структур **Flat Sequence Structure** для организации кода программы
2. Добавлять комментарии в программе
3. Использовать сдвиговые регистры
4. Использовать управление положением мотора при решении робототехнических задач

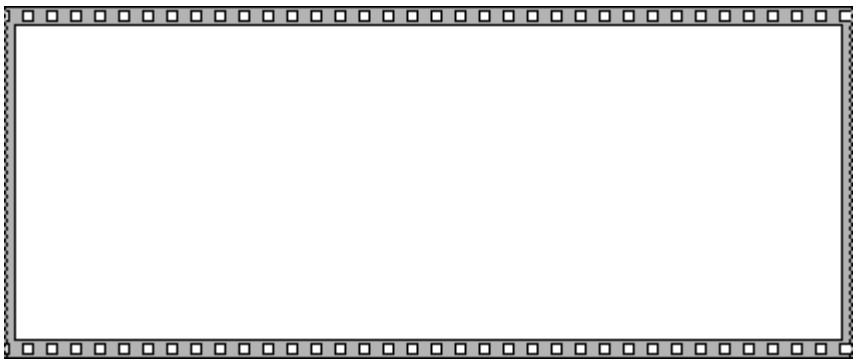
Шариковая пушка

Структура последовательности

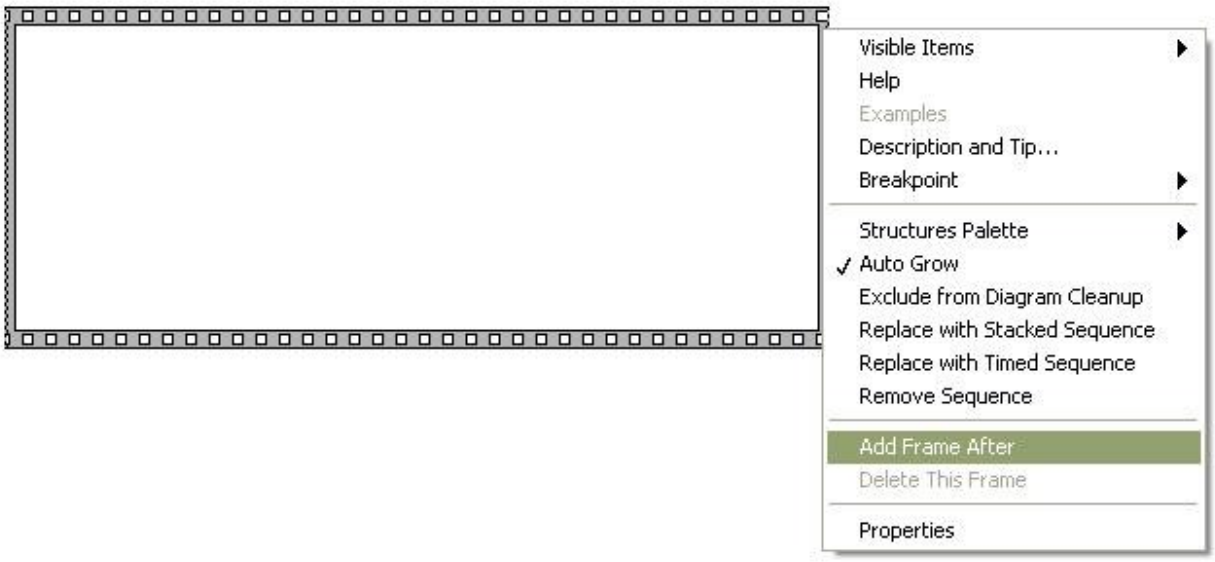
Нам понадобится робот с шариковой пушкой.

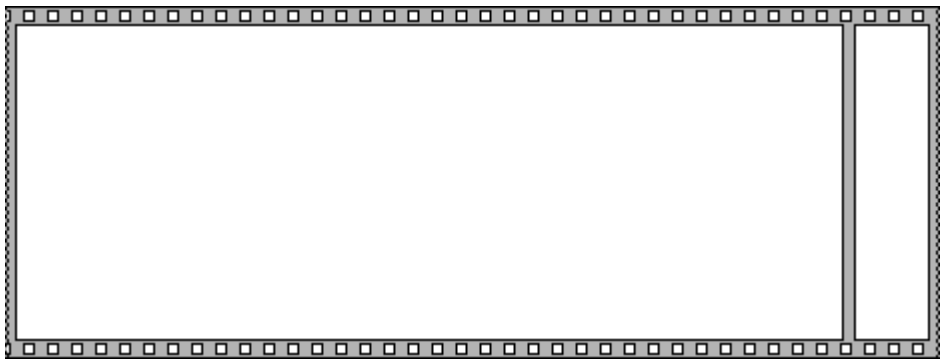
Пушка устроена таким образом что при повороте двигателя на 1 оборот, пушка выстреливает одним шариком. Нашей задачей будет написание программы которая по нажатию кнопки будет стрелять строго одним шариком.

Добавляем на панель редактирования диаграмм структуру **Flat Sequence Structure, NXT programming -> structures -> Flat Sequence Structure**

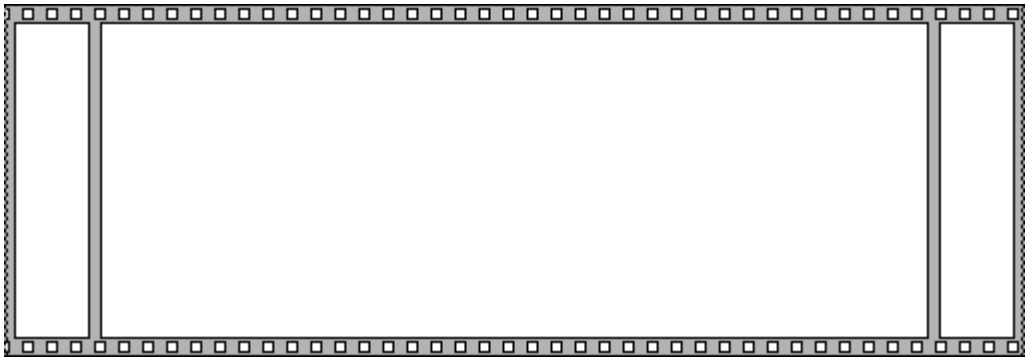


Кликаем правой кнопкой мышки на правой границе структуры и выбираем в меню **Add Frame After (Добавить кадр после текущего).**

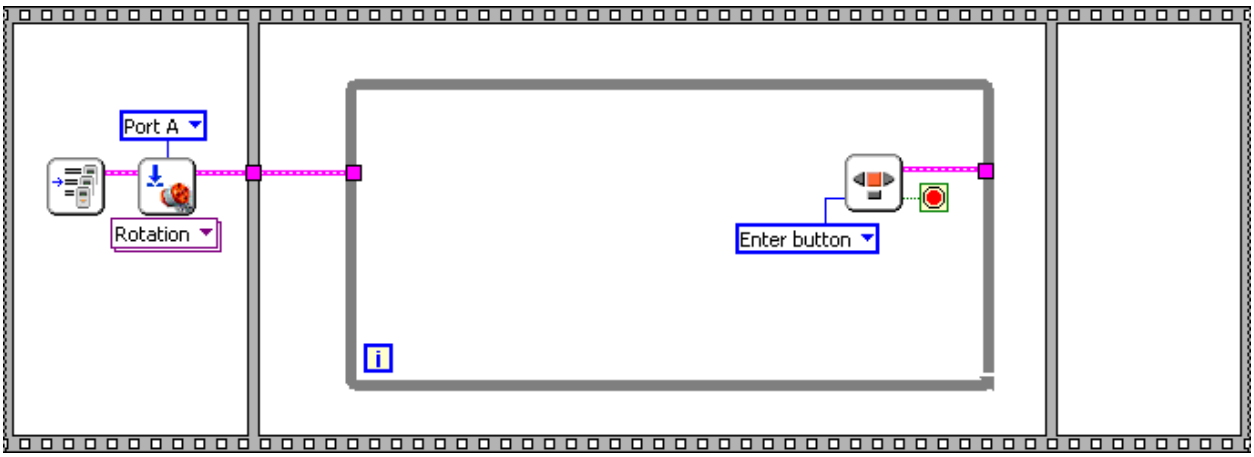




Тоже самое повторяем и с левой границей структуры, и выбираем **Add Frame Beffor (Добавить кадр до текущего).**



Теперь вставляем в окошки структуры шаблон программы который мы использовали ранее:

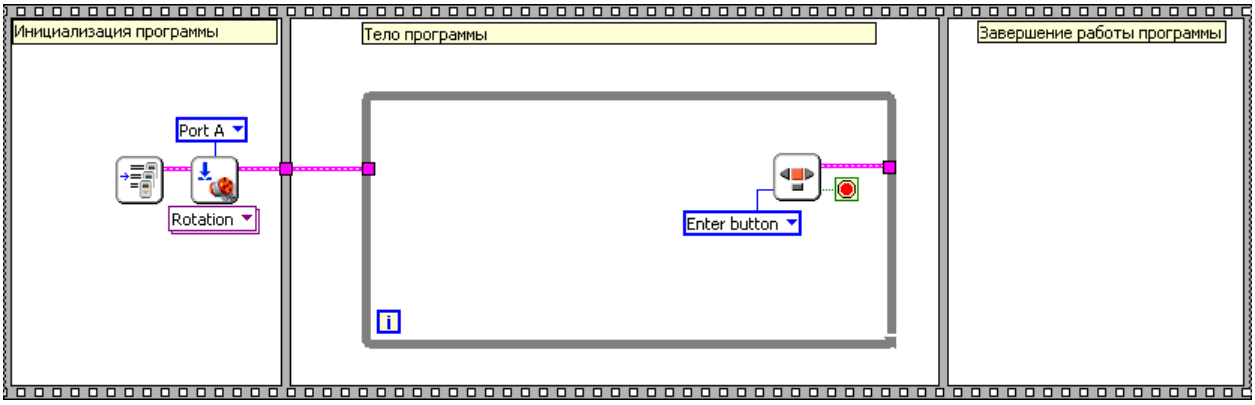


Таким образом наша программа разбита на три этапа (кадра):

- Инициализация программы;
- Тело программы
- Завершение программы

Добавим комментарии которые будут указывать в каком кадре что происходит.

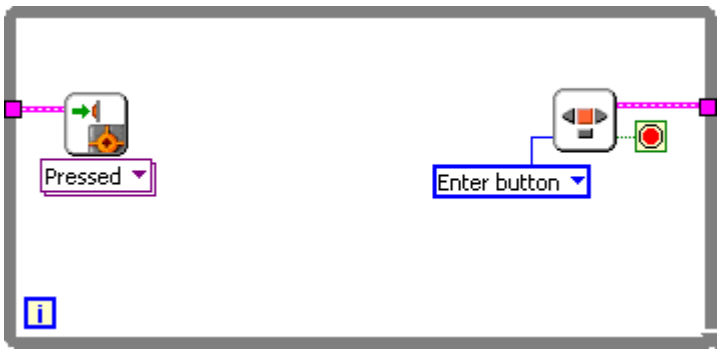
Кликаем двойным щелчком на поле первого кадра, появится желтое поле для ввода текста , это и есть один из наших будущих комментариев программы. Вводим «Инициализация программы» и перемещаем комментарий в верхнюю часть кадра. Также добавляем комментарии во второй кадр и в последний.



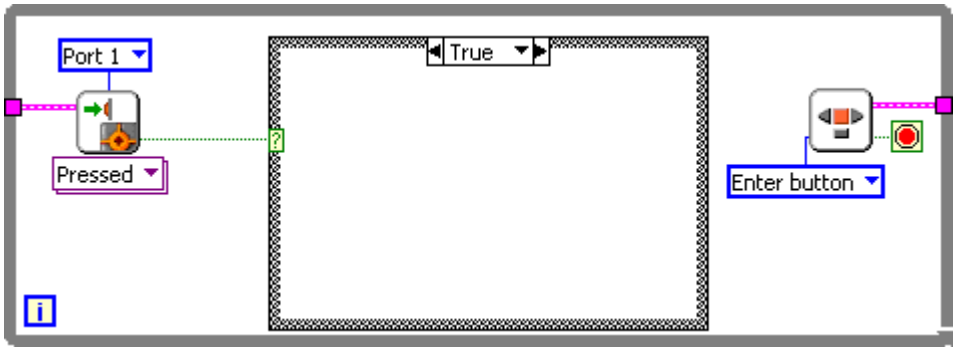
Далее мы будем работать в основном в среднем кадре «тело программы» поэтому в иллюстрациях будем рассматривать только тот кадр в котором мы что-то делаем, меняем добавляем, удаляем и т. д.

Ветвление по нажатию кнопки

По заданию робот должен стрелять шариком по нажатию кнопки, поэтому нам понадобится считывать состояние кнопки. Добавляем функцию чтения датчика **NXT I/O -> Read Sensor.**



По умолчанию выбран датчик касания то есть кнопка, это нас устраивает. Указываем порт к которому подключена кнопка, добавляем структуру ветвления **Case structure** и подсоединяем к терминалу параметра селектора выход состояния кнопки.

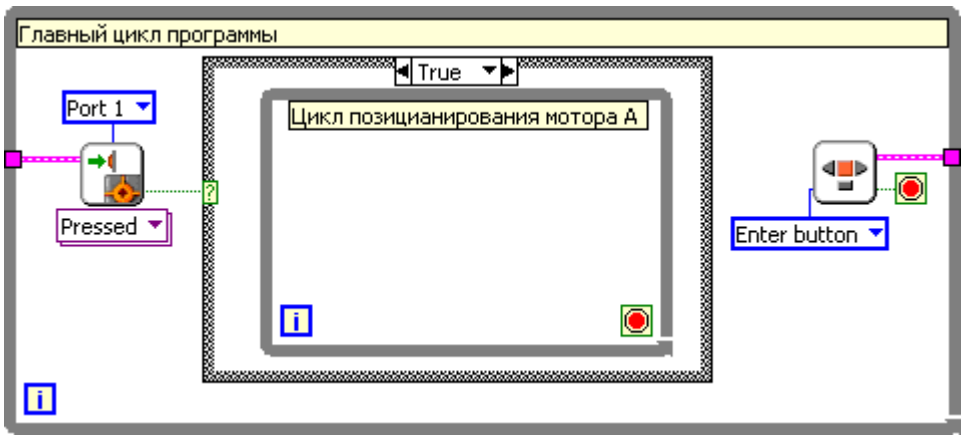


Разбираемся с ветвлением, если кнопка нажата, то тогда нам нужно выстрелить одним шариком, то есть провернуть двигатель ровно на 1 оборот или 360 градусов.

Проверяем что текущая выбранная ветка ветвления соответствует значению True и добавляем в нее цикл.

Цикл позиционирования с выходом по количеству итераций

Теперь у нас есть 2 цикла один внешний – главный цикл программы, а второй внутренний – цикл позиционирования мотора А, добавим комментарии с названиями циклов.

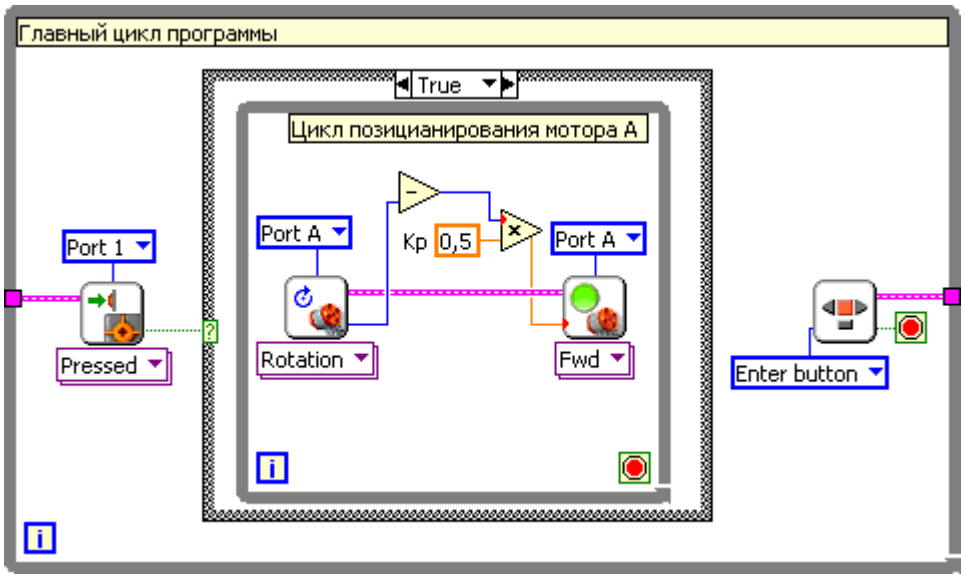


Внутри цикла позиционирования мотора А мы напишем программу аналогичную программе которую мы делали на прошлом уроке.

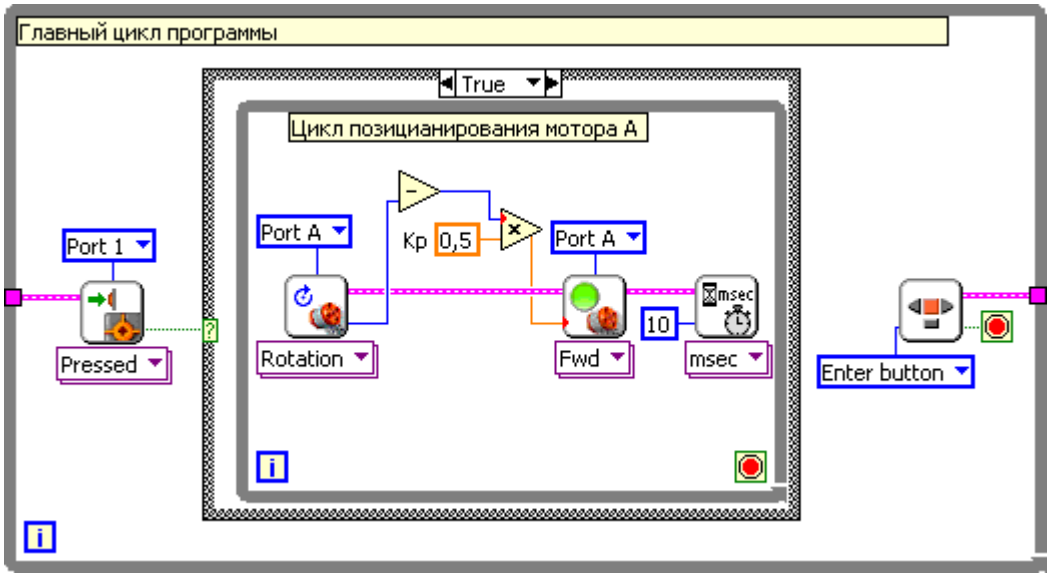
Внутрь цикла позиционирования добавляем следующие функции:

- чтение датчика **NXT I/O -> Read Sensor-> Read Rotation**
- вычитание
- умножение
- управление мотором **NXT I/O -> Motor control**

Значение датчика угла поворота вычитаем из задания (пока задание не подключаем), далее умножаем на коэффициент усиления Kp и подаем на вход задания мощности мотору.



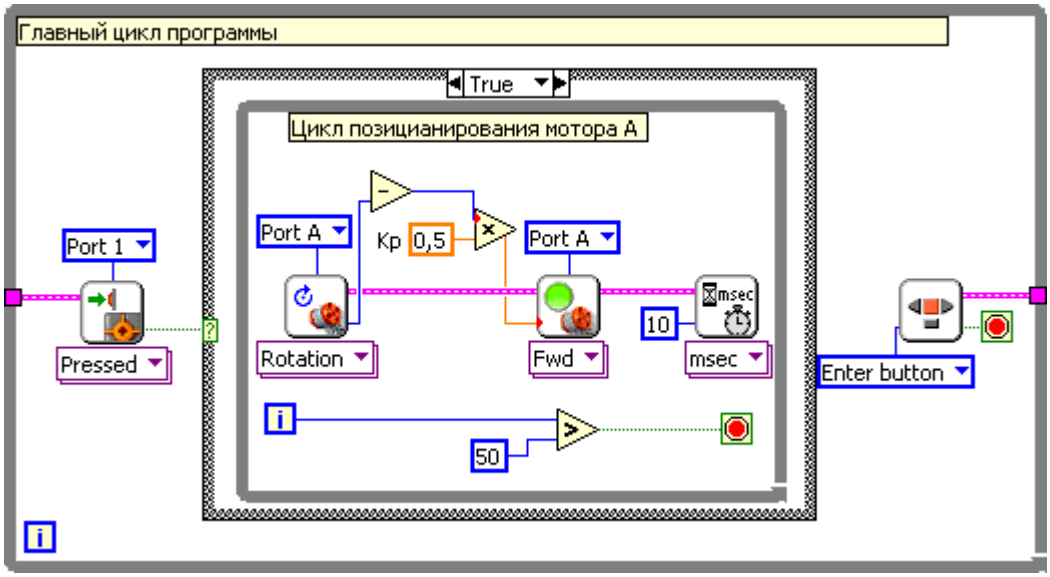
Добавляем временную задержку в 10 милесекунд **NXT I/O -> Wait for** (в меню нужно выбрать time->msec)



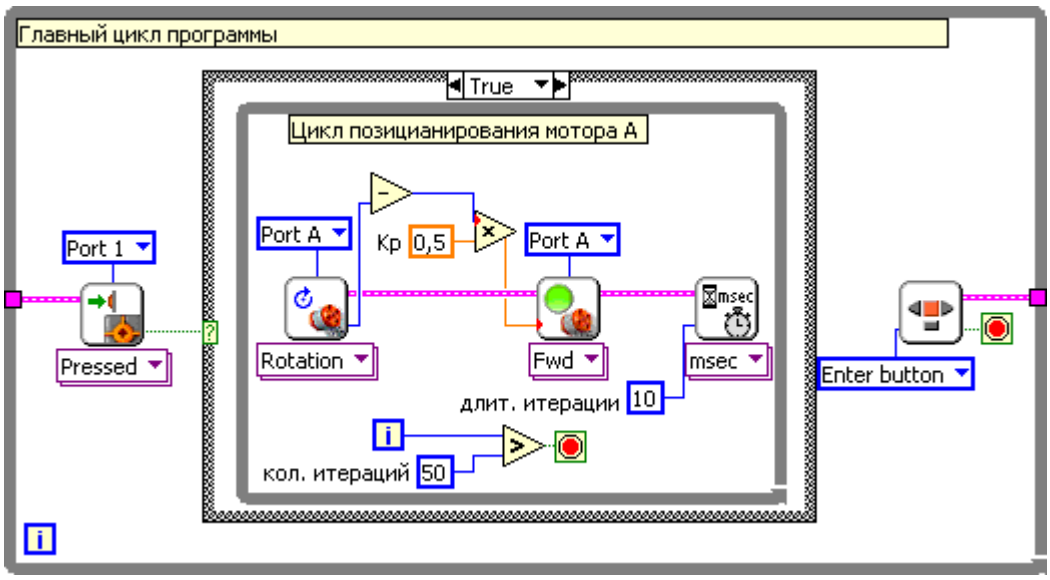
Нам нужно добавить условие по которому цикл будет завершаться, в прошлом уроке цикл позиционирования был главным циклом программы и завершал свою работу по нажатию оранжевой кнопки на панели NXT, сечас нам этот способ не подходит.

Мы сделаем завершение цикла по количеству итераций, для этого добавляем функцию сравнения **greather?**(больше?) **NXT programming -> comparison -> greather?**  **.** Выход функции логический, когда первое число больше второго, x>y, на выходе значение «True» (истина),а когда первое число меньше второго, x<y, на выходе значение «False» (ложь). Присоеденяем выход

функции к терминалу прерывания цикла позиционирования, на верхний вход подаем номер текущей итерации, а около ничжего входа создаем константу со значением 50.

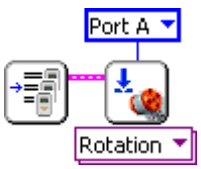


Не забываем у констант сделать поясняющие комментарии, длительность итерации и количество итераций:



Осталось разобраться с тем, какое задание мы выдаем для позиционирования мотора.

После запуска программы в кадре инициализации мы обнуляем датчик положения мотора А



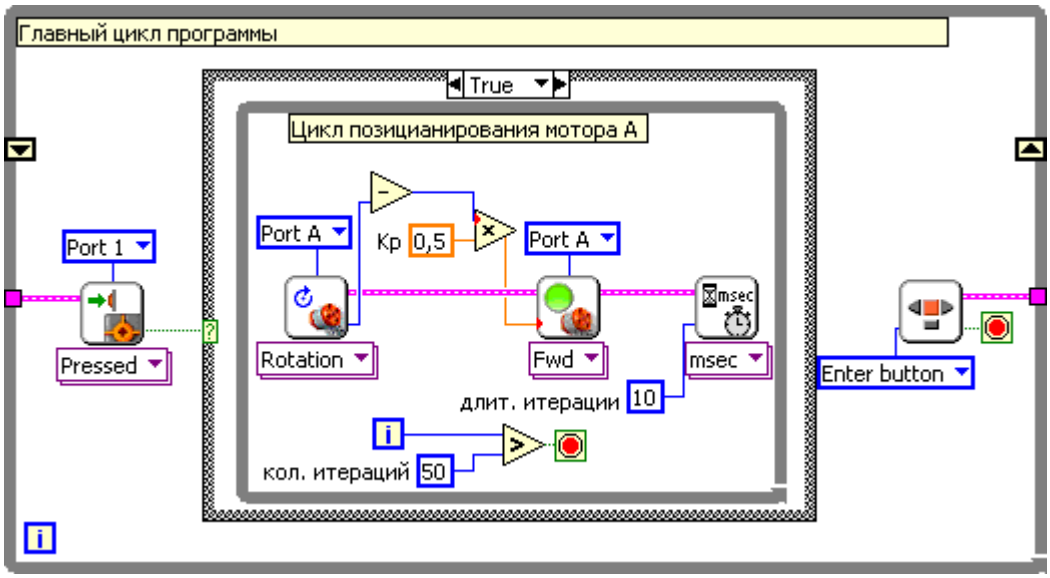
Следовательно для выстрела первого шарика положение задание будет 360, один оборот двигателя. Если мы захотим выстрелить второй раз, то есть два варианта это сделать. Первый, это поставить задание 0, тогда мотор из положения 360 вернется в положение 0, то есть при втором

выстреле, мотор будет крутиться в другую сторону. Второй вариант это дать задание 720 тогда мотор также сделает 1 оборот, но уже в ту сторону что и первый раз. Нам интересен 2 вариант.

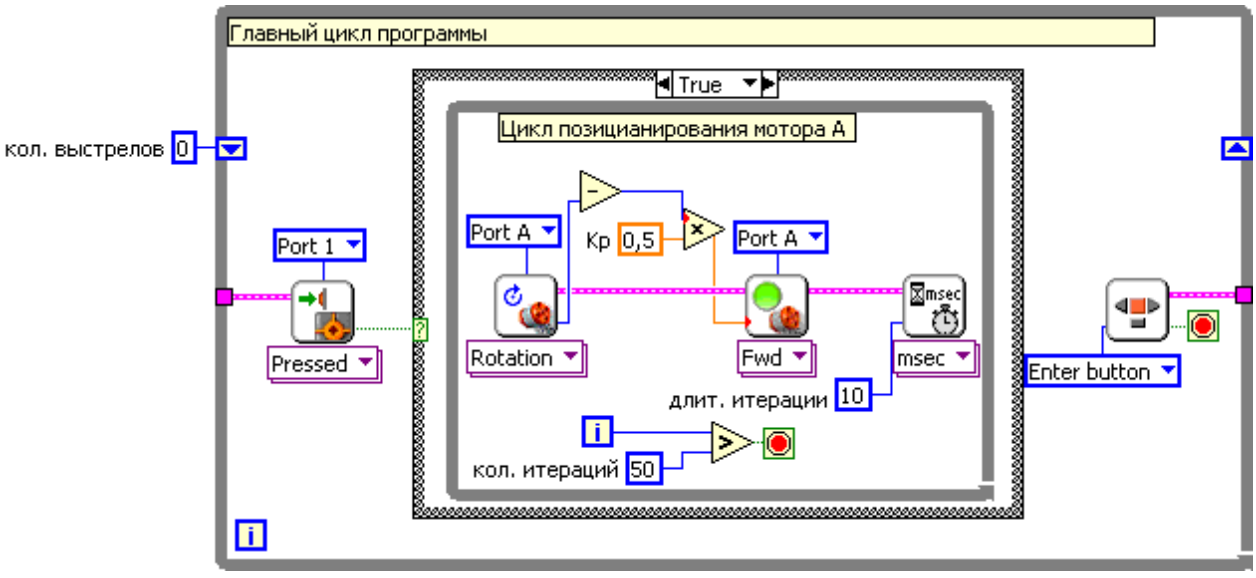
Сдвиговый регистр

Для расчета нового задания, нам нужно будет вести подсчет количества сделанных выстрелов. Для хранения этой информации воспользуемся сдвиговым регистром. Его задача сохранять какое-то значение из предыдущей итерации цикла для использования этого значения в текущей итерации.

Кликаем правой кнопкой мышки на левый край главного цикла программы и в меню выбираем **Add shift register** (добавить сдвиговый регистр).

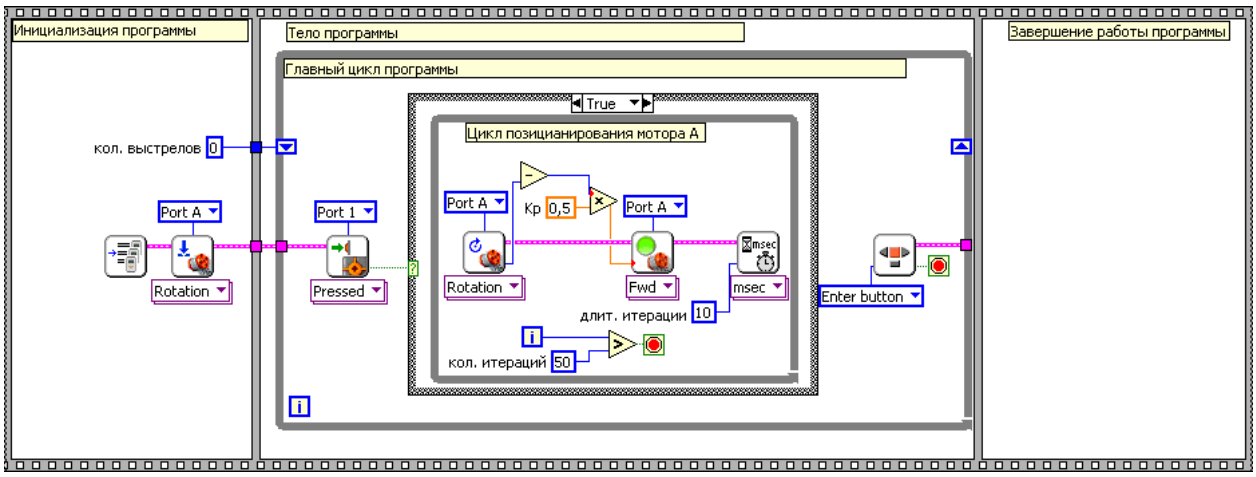


Через палитру функций добавляем константу и подключаем ее ко входному терминалу сдвигового регистра, комментируем эту константу как количество выстрелов, «кол. выстрелов».

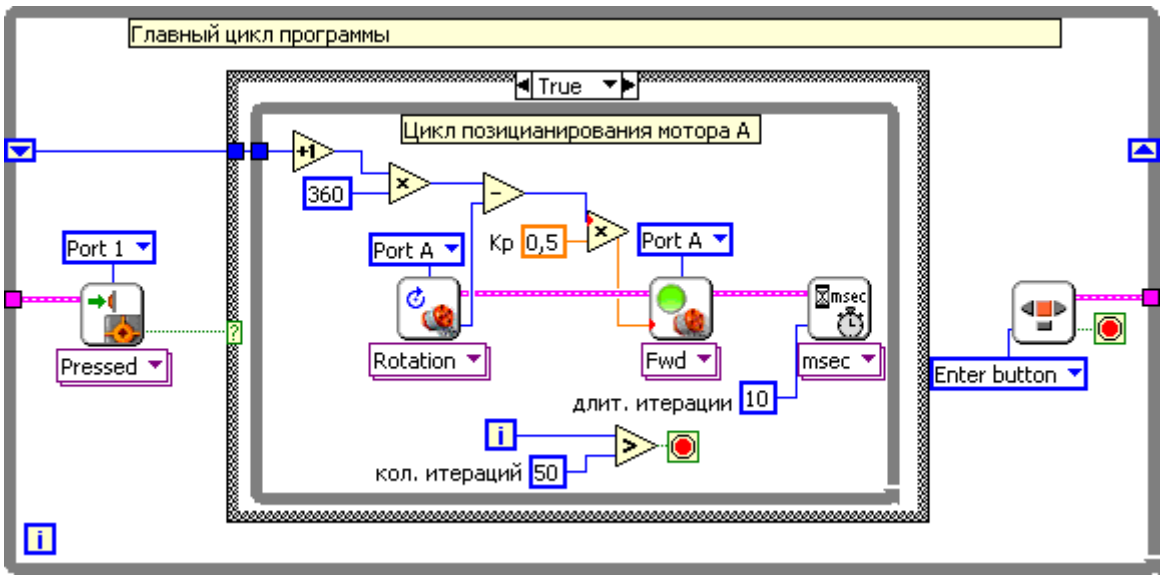


Эта константа нужна для инициализации сдвигового регистра при первой итерации.

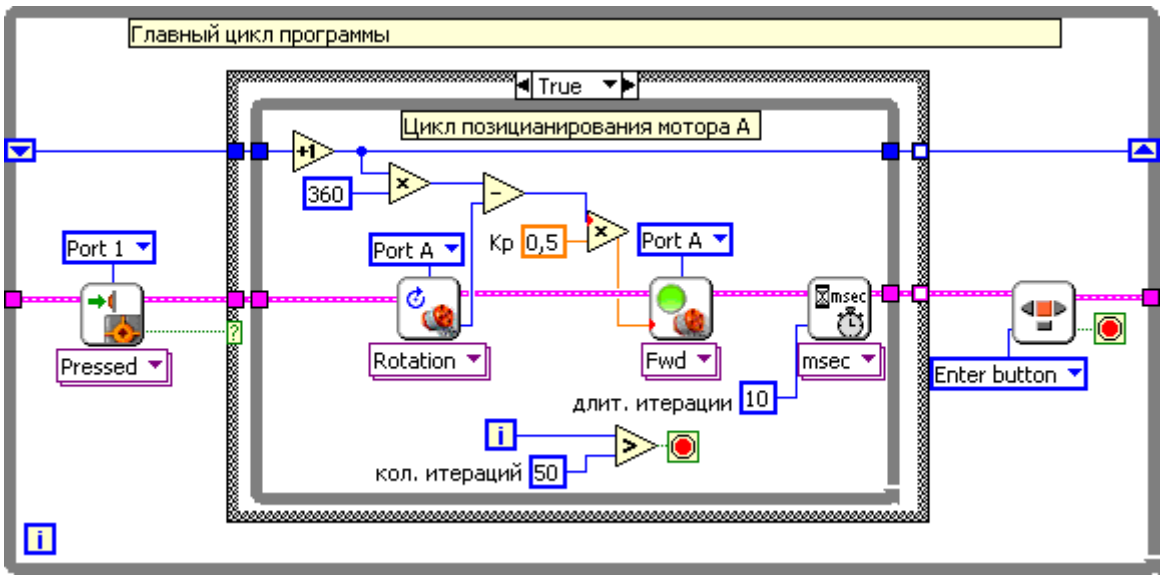
Для инициализации программы у нас есть отдельный кадр, переносим эту константу в первый кадр и через туннель присоединяем ее к терминалу инициализации регистра.



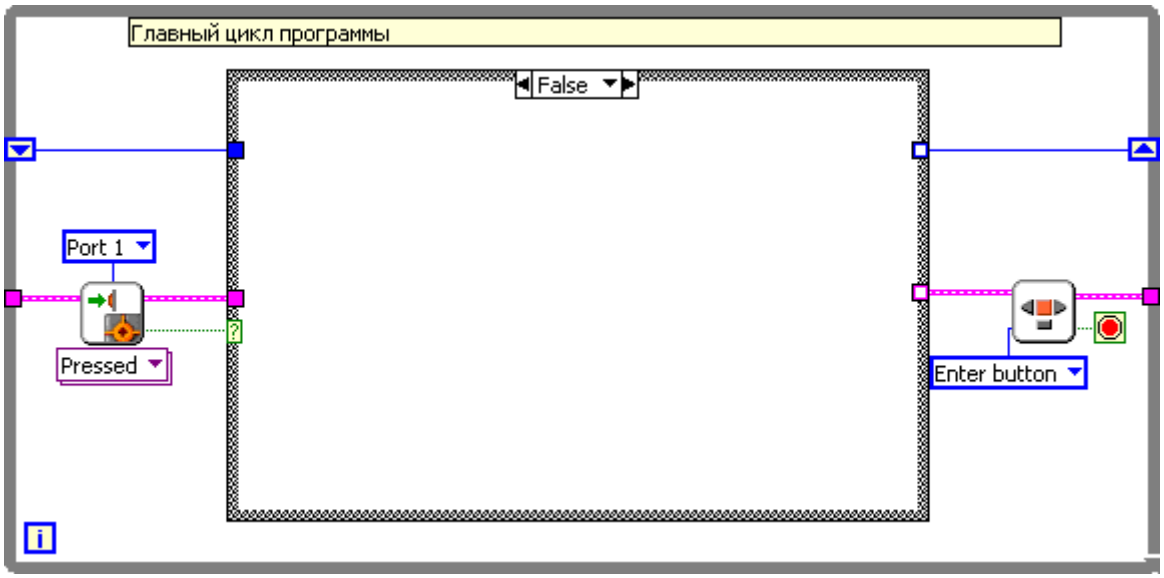
Заводим значение регистра в структуру ветвления прибавляем 1, при помощи функции increment **Functions ->Numeric -> increment**, а после умножаем на 360, чтобы получить задание положения для текущего выстрела.



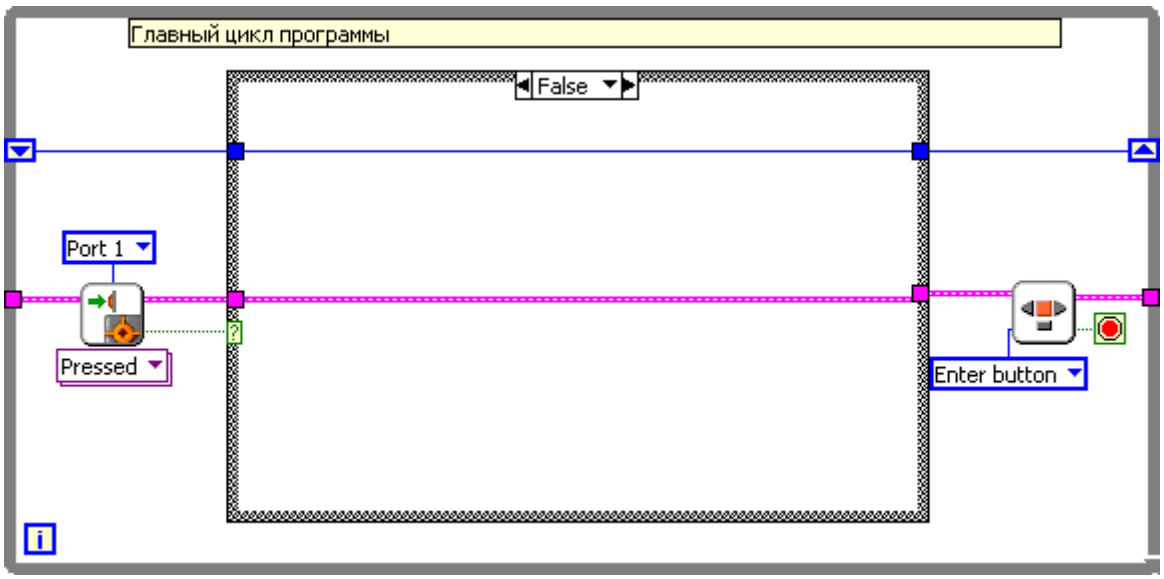
После того как мы увеличили значение регистра с количеством выстрелов на 1, нам нужно передать это значение в следующую итерацию главного цикла. Присоединяем к выходному терминалу регистра выход функции **Increment,** а также соединяем все входы и выходы кластера с информацией об NXT.



Мы закончили с обработкой нажатия кнопки, перейдем теперь к ветке которая выполняется при отжатой кнопке.



Здесь ничего не происходит, поэтому просто соединяем входные туннели с выходными.



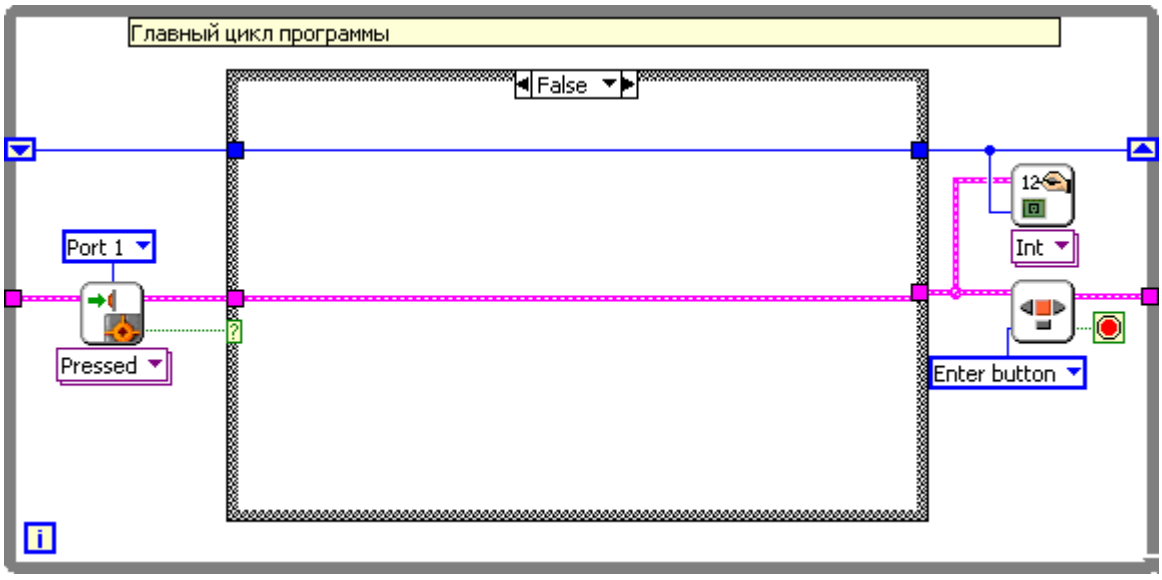
Сохраняем программу.

Загружаем в NXT и проверяем ее работу.

Заключение

В заключении этого урока давайте выведем количество совершенный выстрелов на дисплей NXT

Добавляем функцию Display control **Functions -> NXT I/O -> Display control,** в меню выбираем **Write -> integer**. Присоединяем эту функцию параллельно опросу состояния кнопки «Enter», после ветвления.



Загружаем программу в NXT и проверяем ее работу.

Дополнительное задание

Обойма шариковой пушки состоит из 5 патронов, сделайте так, чтобы после 5 выстрелов при нажатии кнопки-курка, робот не стрелял.