**Полезные соотношения:**

Общепринятая функциональная зависимость для построения графика в полярной системе координат: .

Функциональная зависимость для построения графика в полярной системе координат, удобная для понимания и восприятия обучающимися: .

Проецирование на окружность функциональной зависимости: , .

----------------------------------------------------------------------------------------------------

**Подготовка исходных данных для построения графиков функциональных зависимостей**

Исходными данными для построения графиков функциональных зависимостей являются два параметра:

- диапазон построения ([*Xn*; *Xk*]);

- шаг построения (*h*).

Диапазон построения, в свою очередь, распадается на два параметра:

- начало диапазона построения (*Xn*);

- конец диапазона построения (*Xk*).

Итого для ввода исходной информации, необходимой для построения графиков функциональных зависимостей в пакете прикладных программ *National Instruments LabView,* необходимо предусматривать на графическом пользовательском интерфейсе три вещественных числовых контроллера. Эта структура входной информации является универсальной, потому уместно хранить её в качестве шаблона виртуального прибора, необходимого для отображения графиков функциональных зависимостей (Рисунок 1).

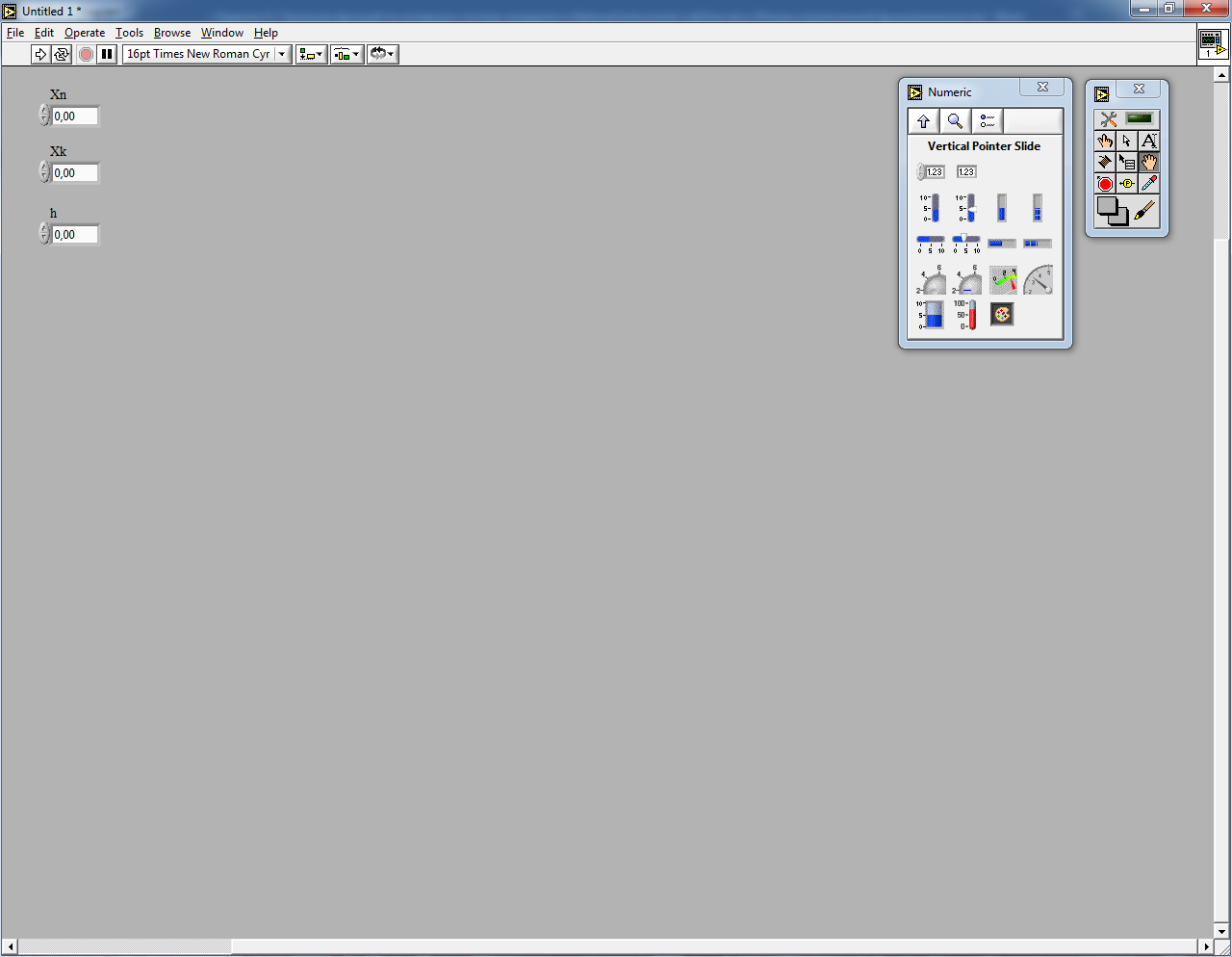


Рисунок 1 – Размещение на передней панели вещественных контроллеров, предназначенных для ввода начала диапазона построения (*Xn*), конца диапазона построения (*Xk*), шага построения (*h*)

Помимо входной информации в шаблоне виртуального прибора разумно сразу предусмотреть и выходную информацию. С точки зрения графического пользовательского интерфейса всё просто – в качестве выходной информации достаточно иметь в наличии область для построения графиков функциональных зависимостей (*XY Graph*). Этот элемент располагается в разделе графиков (*Graph*) в перечне интерфейсных элементов управления (Рисунок 2).

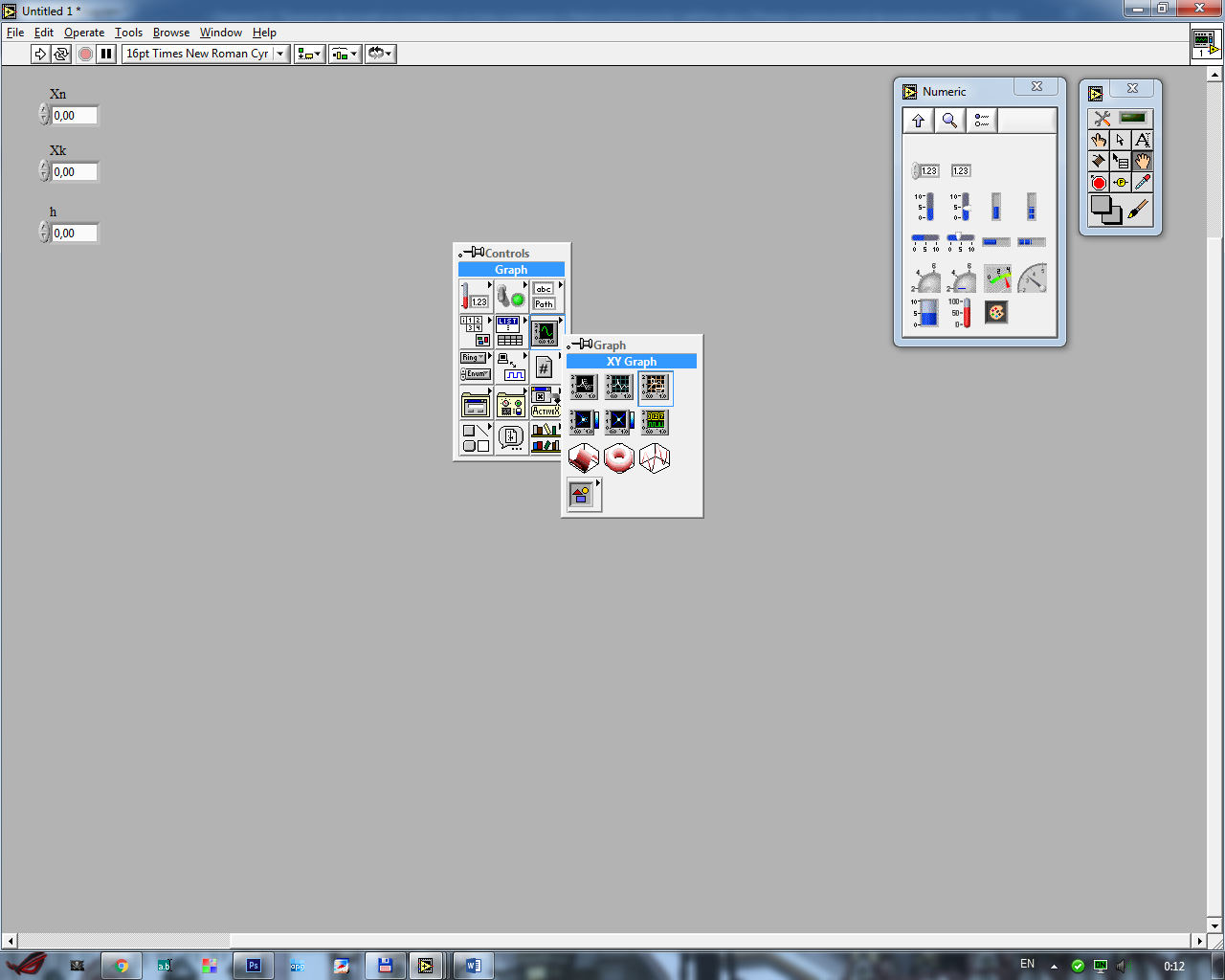


Рисунок 2 – Маркировка места расположения области для построения графиков функциональных зависимостей (*XY Graph*) из раздела графических интерфейсных элементов (*Graph*)

Вместе с размещённой областью для построения графиков функциональных зависимостей передняя панель виртуального прибора «График функциональной зависимости» начинает выглядеть громоздко, но взамен занятому пространству достигается высокая информативность графического пользовательского интерфейса (Рисунок 3).

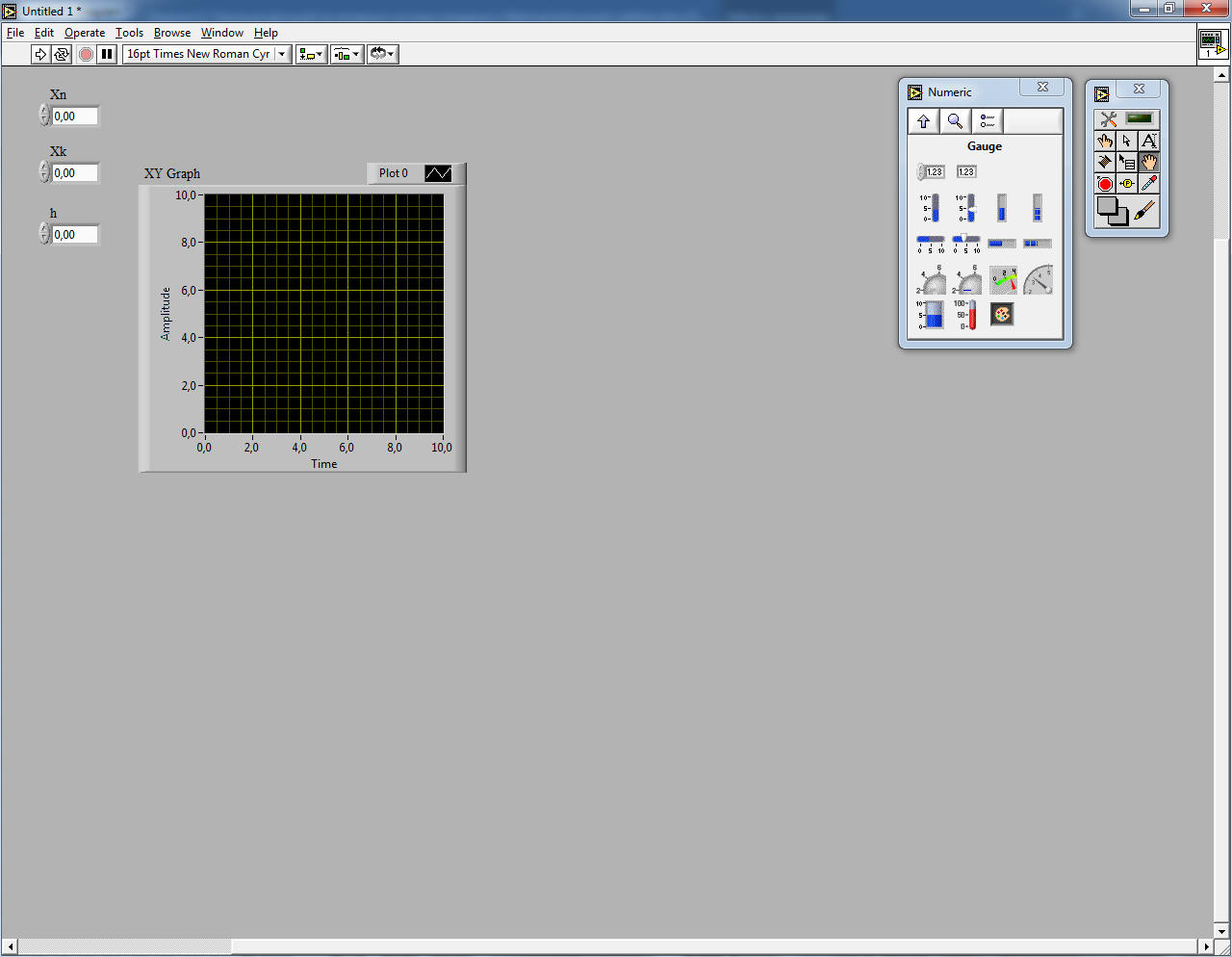


Рисунок 3 – Размещение области для построения графиков функциональных зависимостей (*XY Graph*) на передней панели виртуального прибора

Заключительным этапом настройки шаблона виртуального прибора, необходимого для построения графиков функциональных зависимостей, является фиксация значений диапазона и шага, заданных по умолчанию (Рисунок 4). Как правило, по умолчанию в качестве начала диапазона выбирается абсцисса, равная «–10», в качестве конца диапазона – абсцисса, равная «+10», шаг выставляется равным «0.01». После внесения этих значений в соответствующие числовые контроллеры для каждого из них в контекстном меню, вызываемом правой кнопкой мыши, выбирается следующее «*Data Operations > Make Current Value Default*» (Операции с данными > Установить значение заданным по умолчанию).

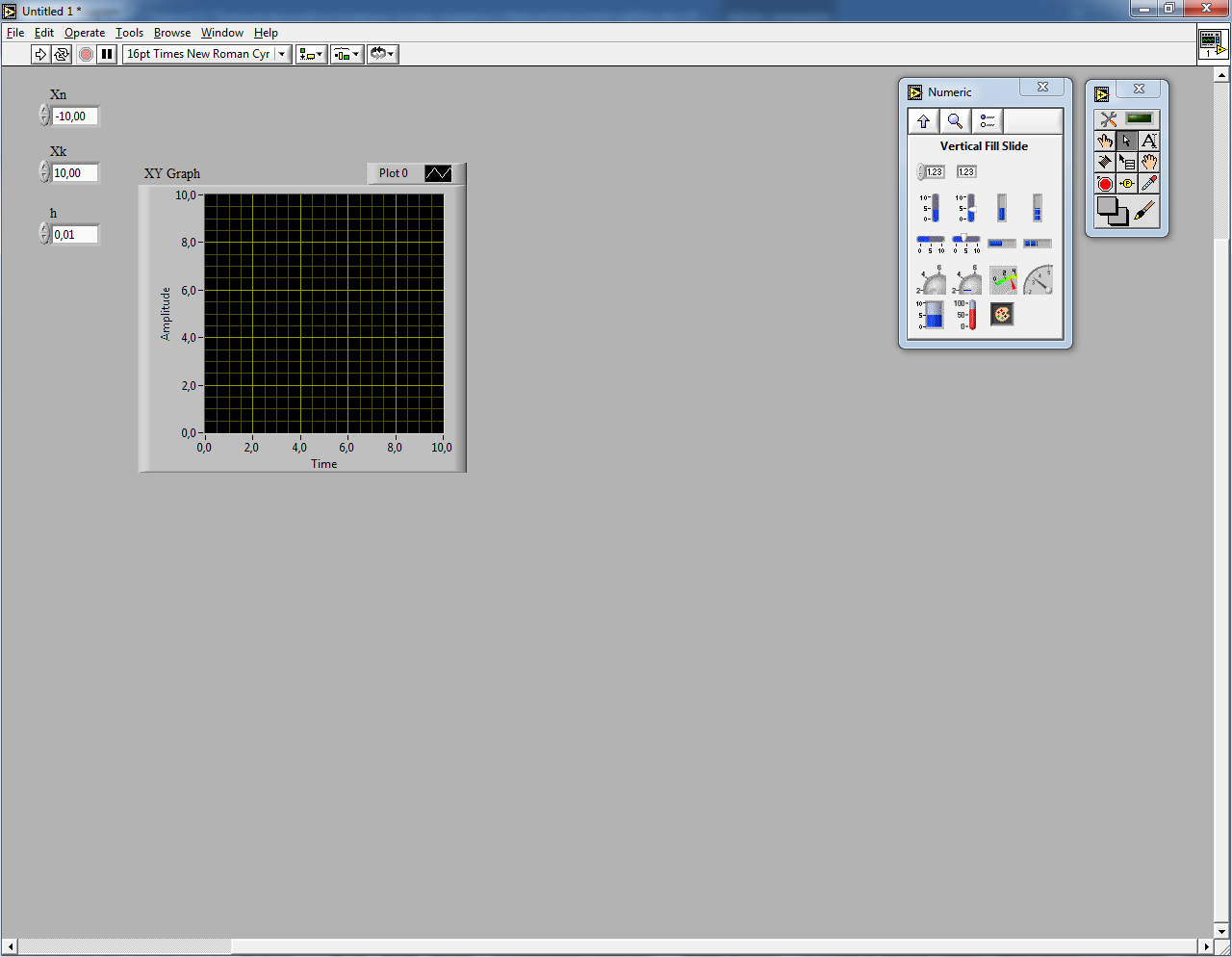


Рисунок 4 – Настройка стандартных значений диапазона и шага

На этом работу по настройке передней панели виртуального прибора можно считать завершённой, потому на следующем этапе необходимо перейти к настройке блок-диаграммы – к написанию кода виртуального прибора. Там уже в соответствующем порядке расположены отображения элементов передней панели (Рисунок 5).

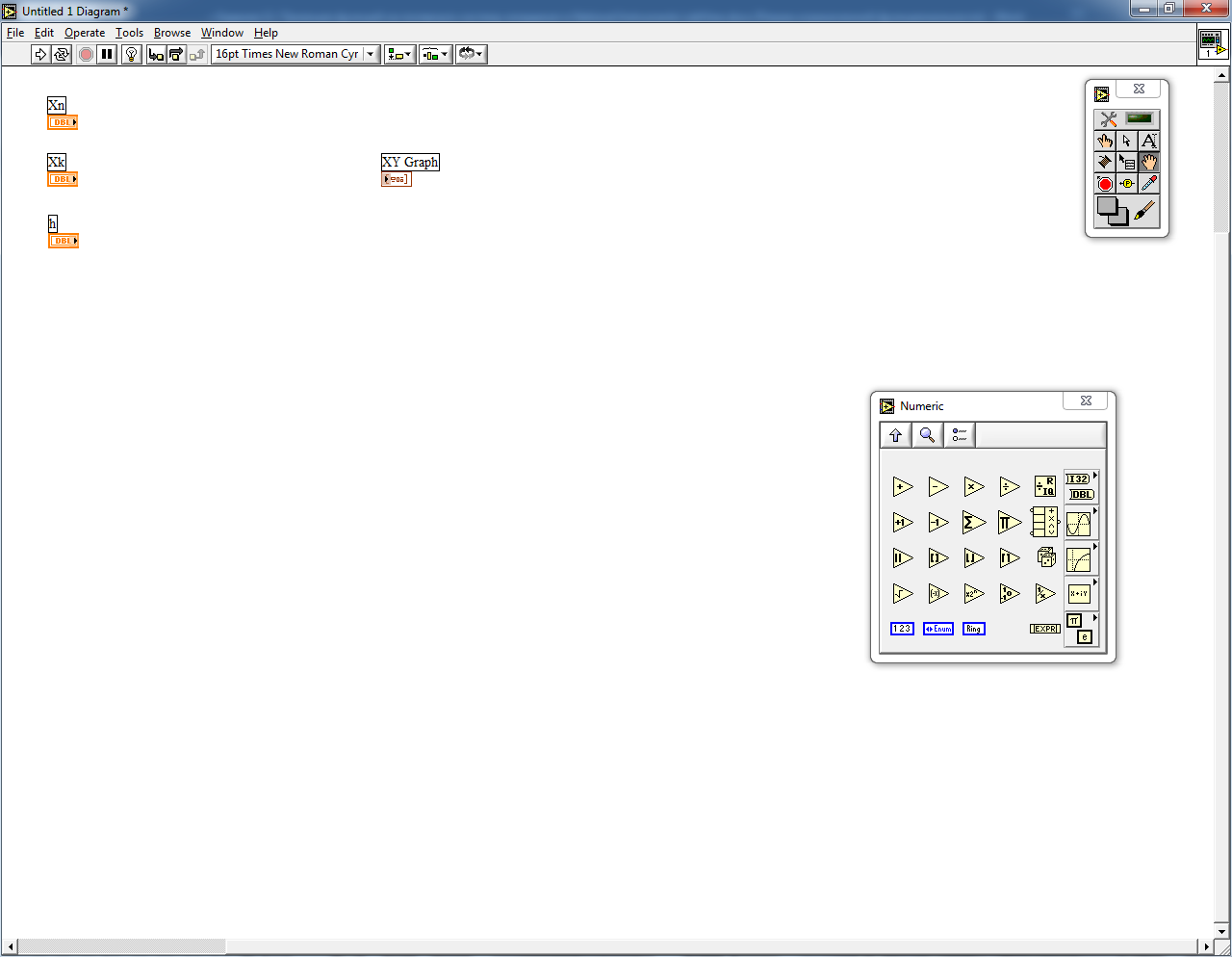


Рисунок 5 – Переход к написанию кода на блок-диаграмме. Исходное состояние входной и выходной информации, необходимой для построения графиков функциональных зависимостей

Данные для построения графиков функциональных зависимостей должны поставляться в виде массивов точек. Математически объект типа «точка» состоит из «сцепки» координат по абсциссе и ординате, определяющих положение данного объекта на плоскости (*x*; *y*). На языке пакета прикладных программ *National Instruments LabView* «точка» – это кластер («сцепка»), состоящий из двух координат. Потому для создания точки необходимо организовать упомянутую «сцепку». Делается «сцепка» посредством размещения на блок-диаграмме элемента сцепления (*Bundle*), расположение которого в перечне функций, размещаемых на блок-диаграмме, показано на Рисунке 6. На входы элемента сцепления последовательно сверху вниз подаются значения абсцисс и ординат.

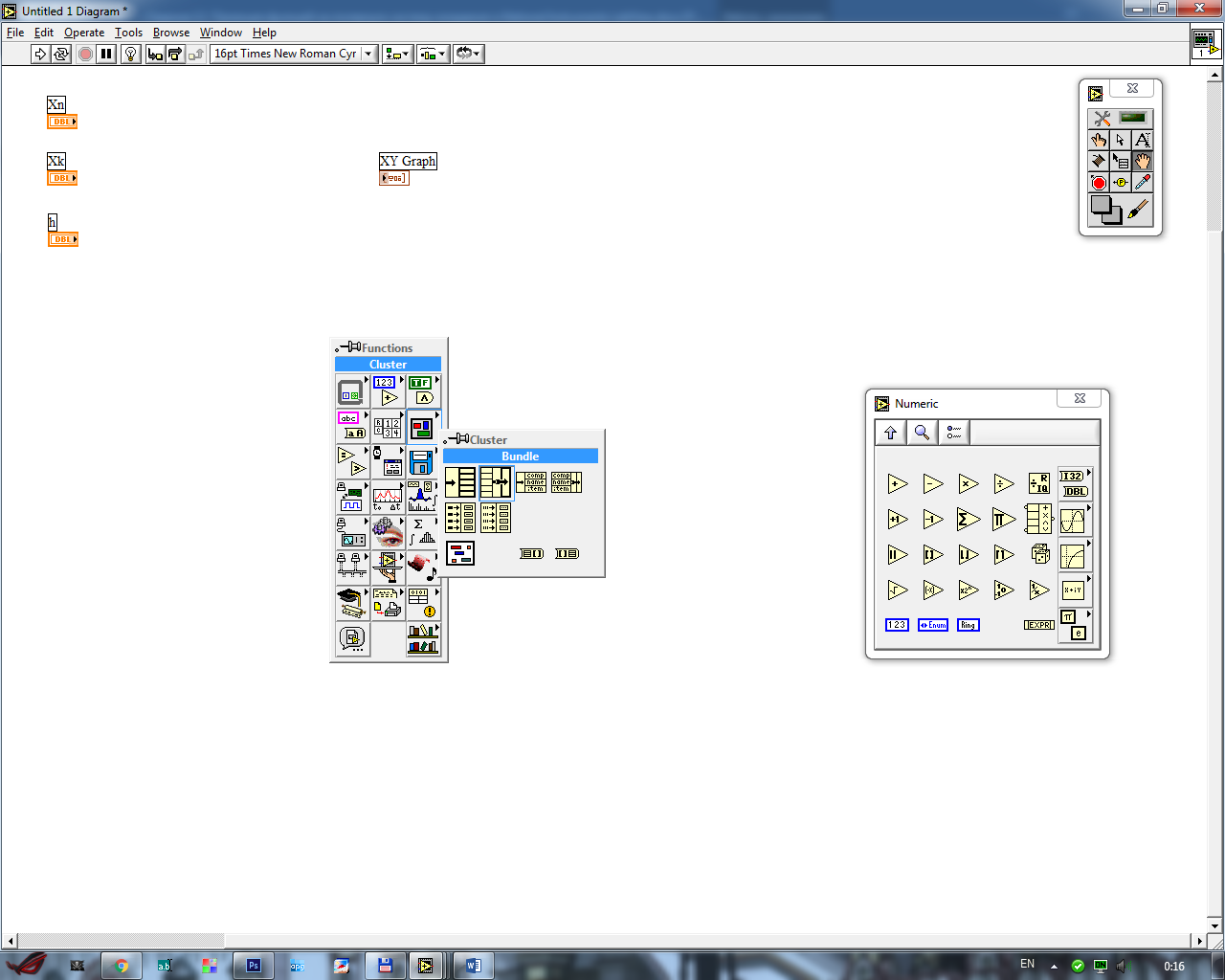


Рисунок 6 – Маркировка элемента сцепления (*Bundle*) компонентов в кластер в разделе кластеров (*Cluster*) перечня основных функций блок-диаграммы

К настоящему моменту в шаблоне отсутствуют какие-либо значения абсцисс и ординат, потому элемент с пустыми значениями на входе размещается на блок-диаграмме и подключается своим выходом ко входу области для построения графиков функциональных зависимостей. Пакет прикладных программ *National Instruments LabView* отметит данную связь как ошибочную, но на данном этапе это абсолютно нормальная ситуация, поскольку создаётся шаблон-заготовка (Рисунок 7).

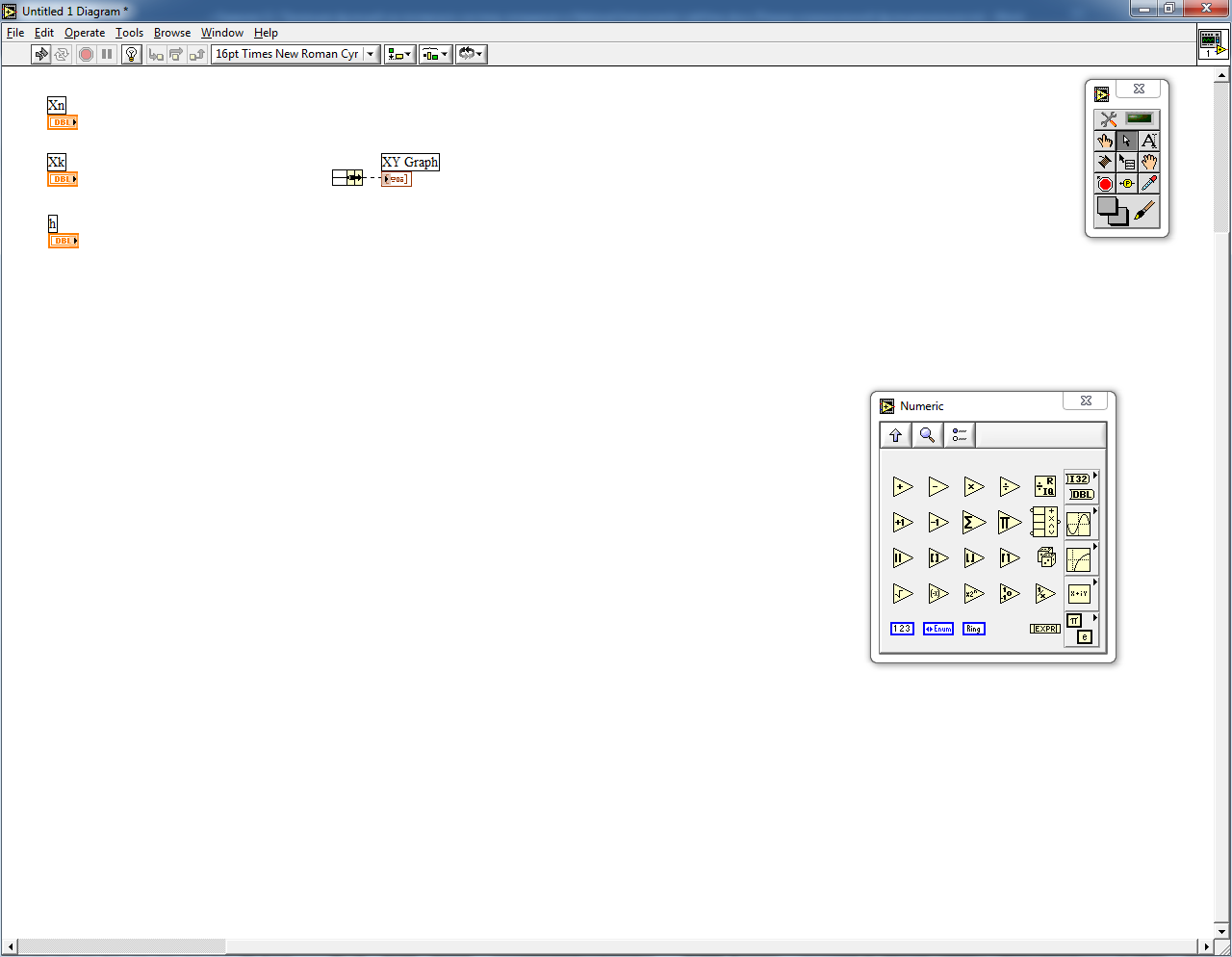


Рисунок 7 – Подключение пустой «сцепки» к области для построения графиков функциональных зависимостей

Поскольку получение значений абсцисс и ординат точек является циклическим процессом, требующим отдельного рассмотрения, для начала разумно в деталях познакомиться с работой элемента «область для построения графиков функциональных завистей» на статичном наборе данных. Наибольшую наглядность способны обеспечить константы, хранимые в целочисленных массивах. Место расположения константы, содержащей массив неопределённого типа, отмечено в перечне функций, размещаемых на блок-диаграмме, на Рисунке 8.

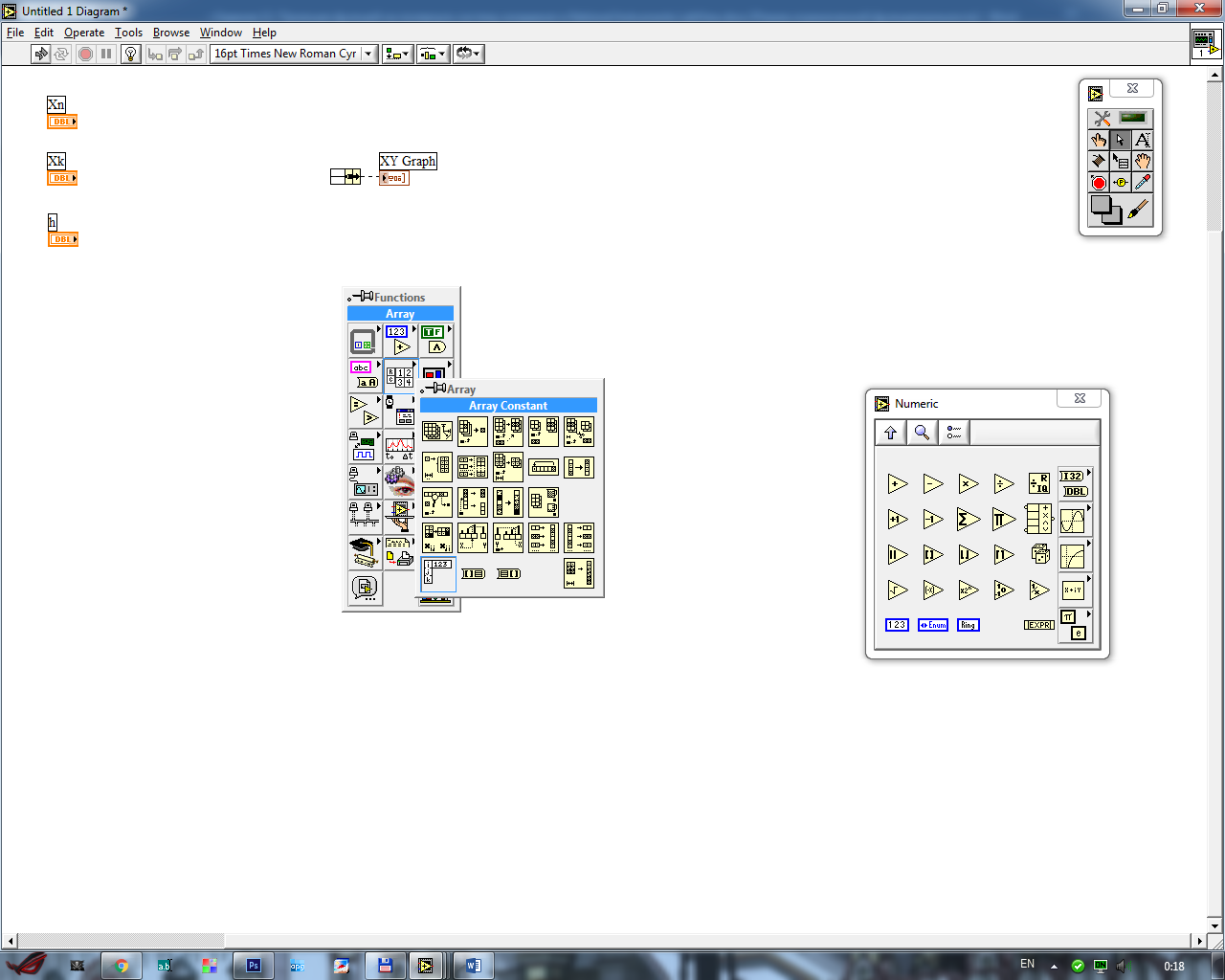


Рисунок 8 – Маркировка элемента «массив констант» (*Array Constant*) в разделе массивов (*Array*) перечня функций, размещаемых на блок-диаграмме

Размещение на блок-диаграмме элемента, для которого ещё не указан тип хранимых в нём данных, представлено на Рисунке 9.

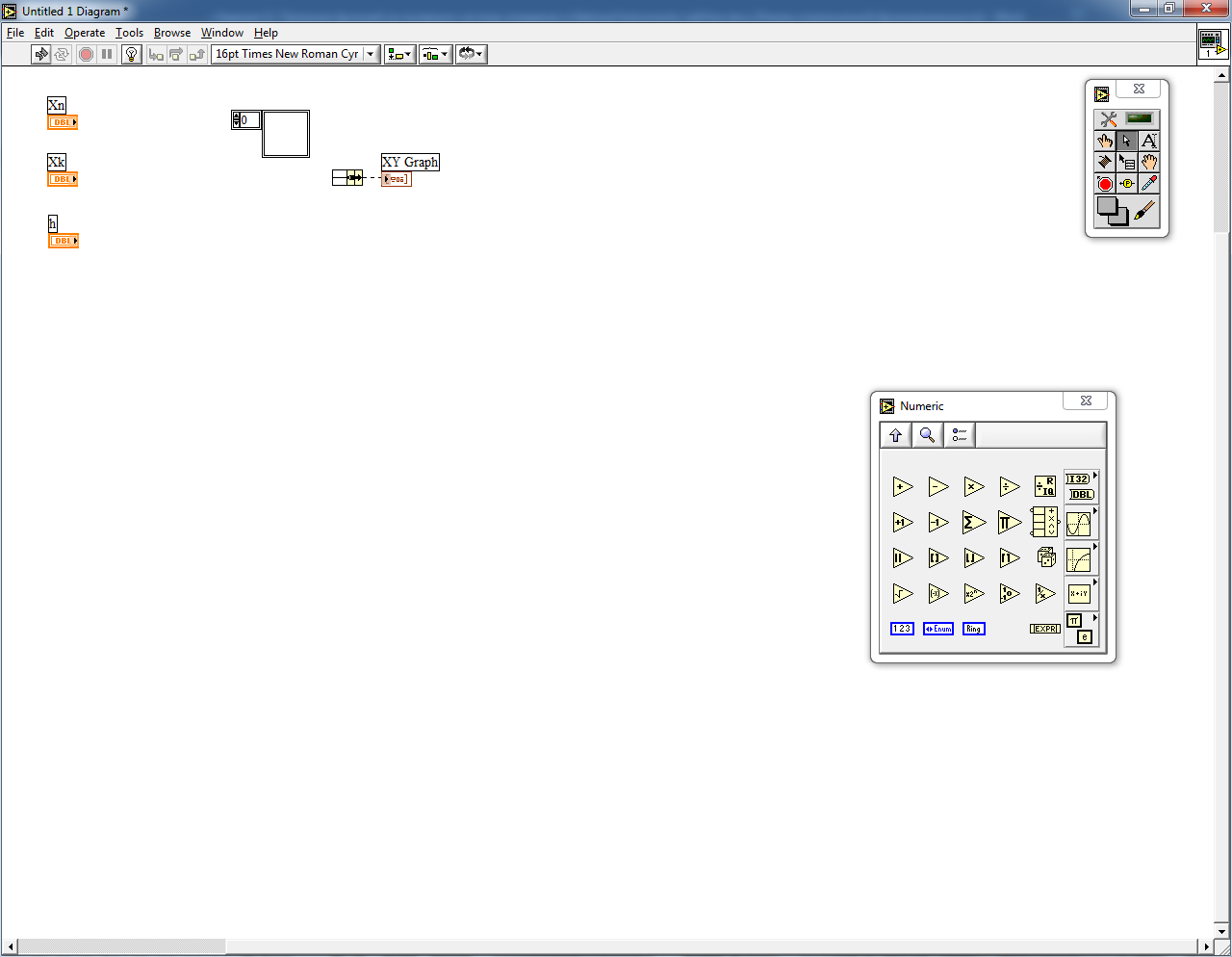


Рисунок 9 – Размещение не типизированного массива констант на блок-диаграмме

Для назначения числового типа данному элементу (в рассматриваемом случае целого числового типа) необходимо в разделе численных функций выбрать численную константу, которую разместить в области массива (Рисунок 10).

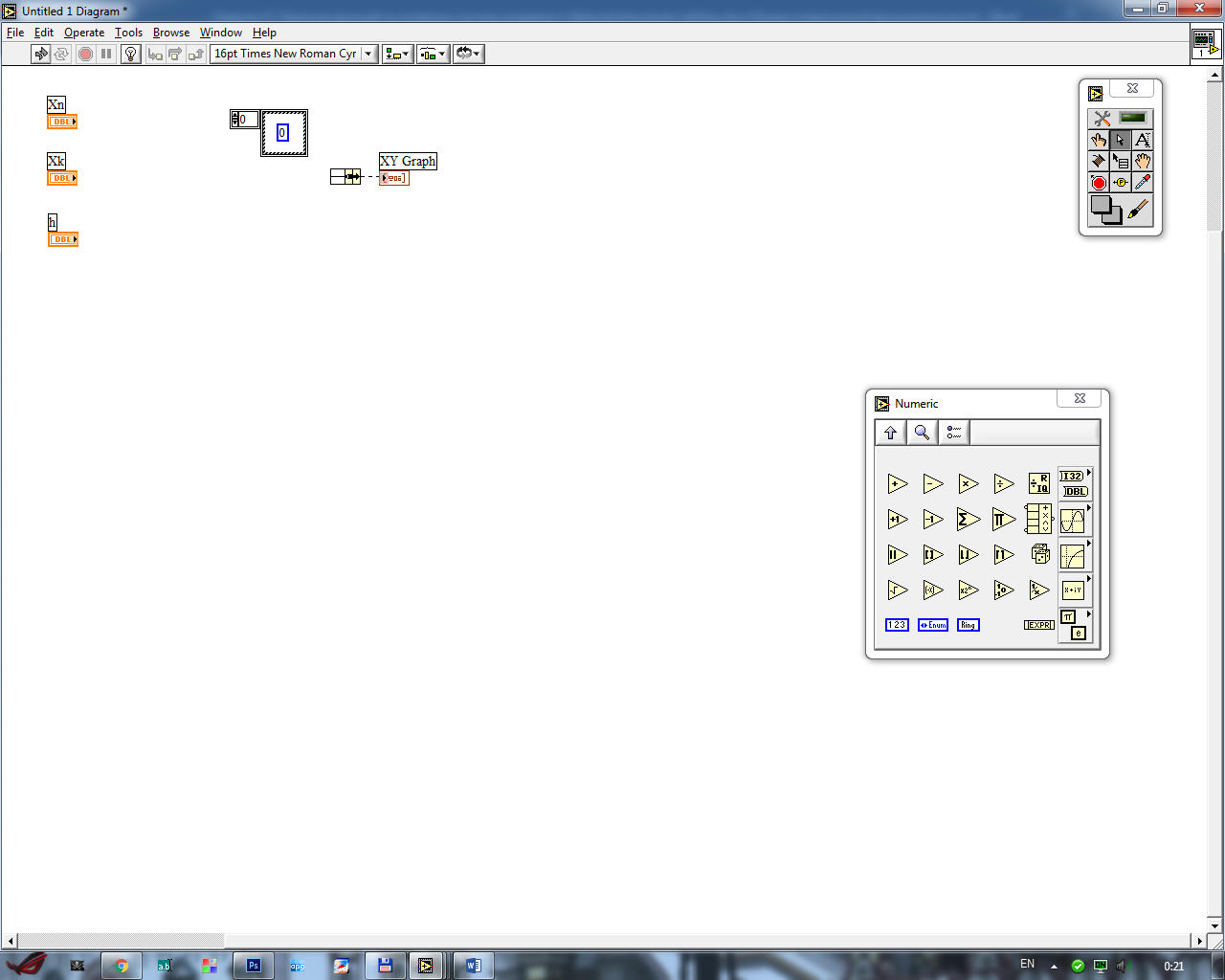


Рисунок 10 – Типизация массива констант посредством размещения в нём целочисленной константы

После выполнения такой операции структура массива констант несколько преобразится (Рисунок 11), в частности, окрасится в цвет указанного типа данных (в рассматриваемом случае – в синий цвет, соответствующий целым числам).

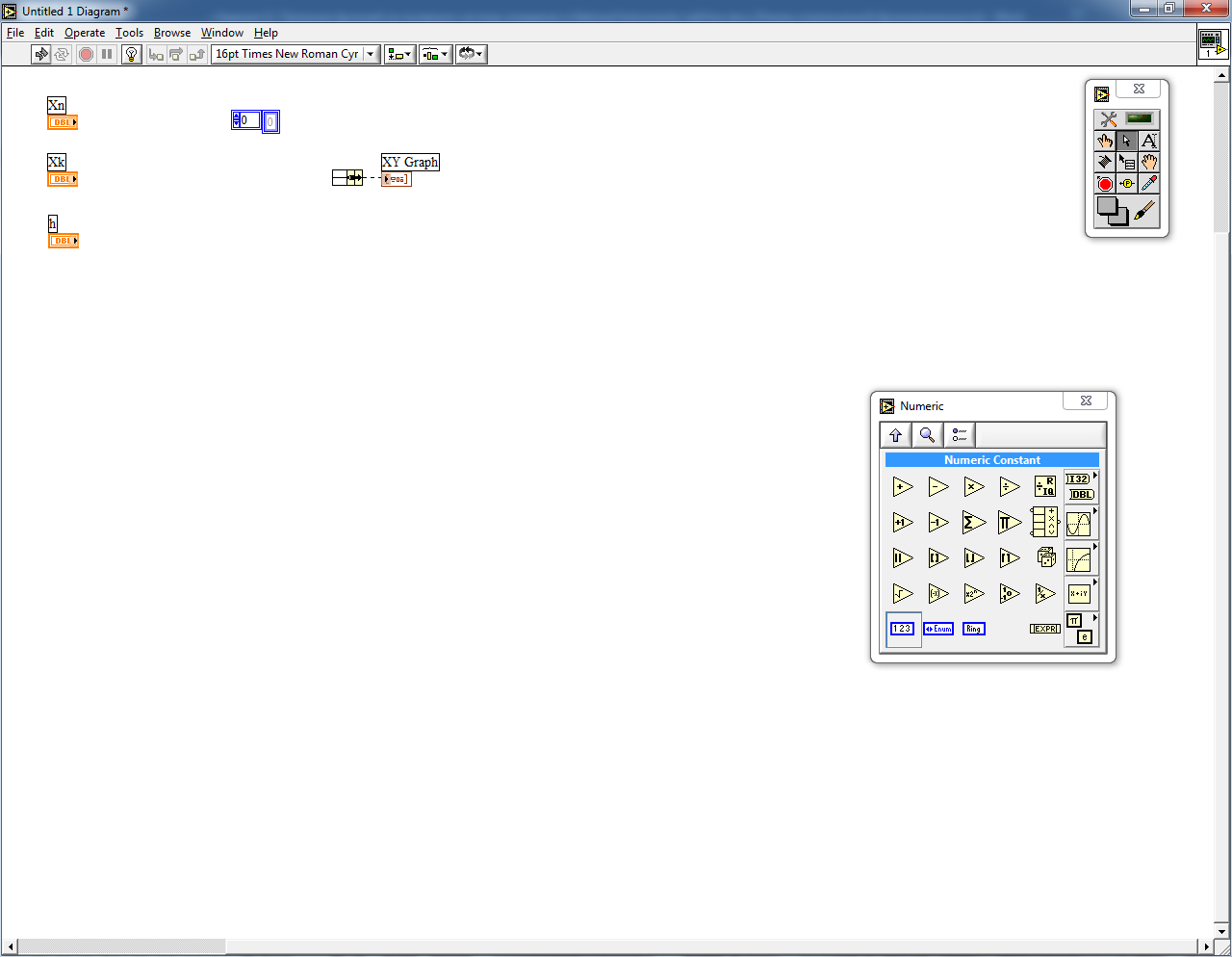


Рисунок 11 – Результат приведения массива констант к целому числовому типу

Для наглядности увеличим количество отображаемых элементов массива. Для этого на панели инструментов необходимо выбрать элемент позиционирования / изменения размерности / выбора и вытянуть интересующий элемент вниз, «ухватившись» за правый нижний угол (Рисунок 12).

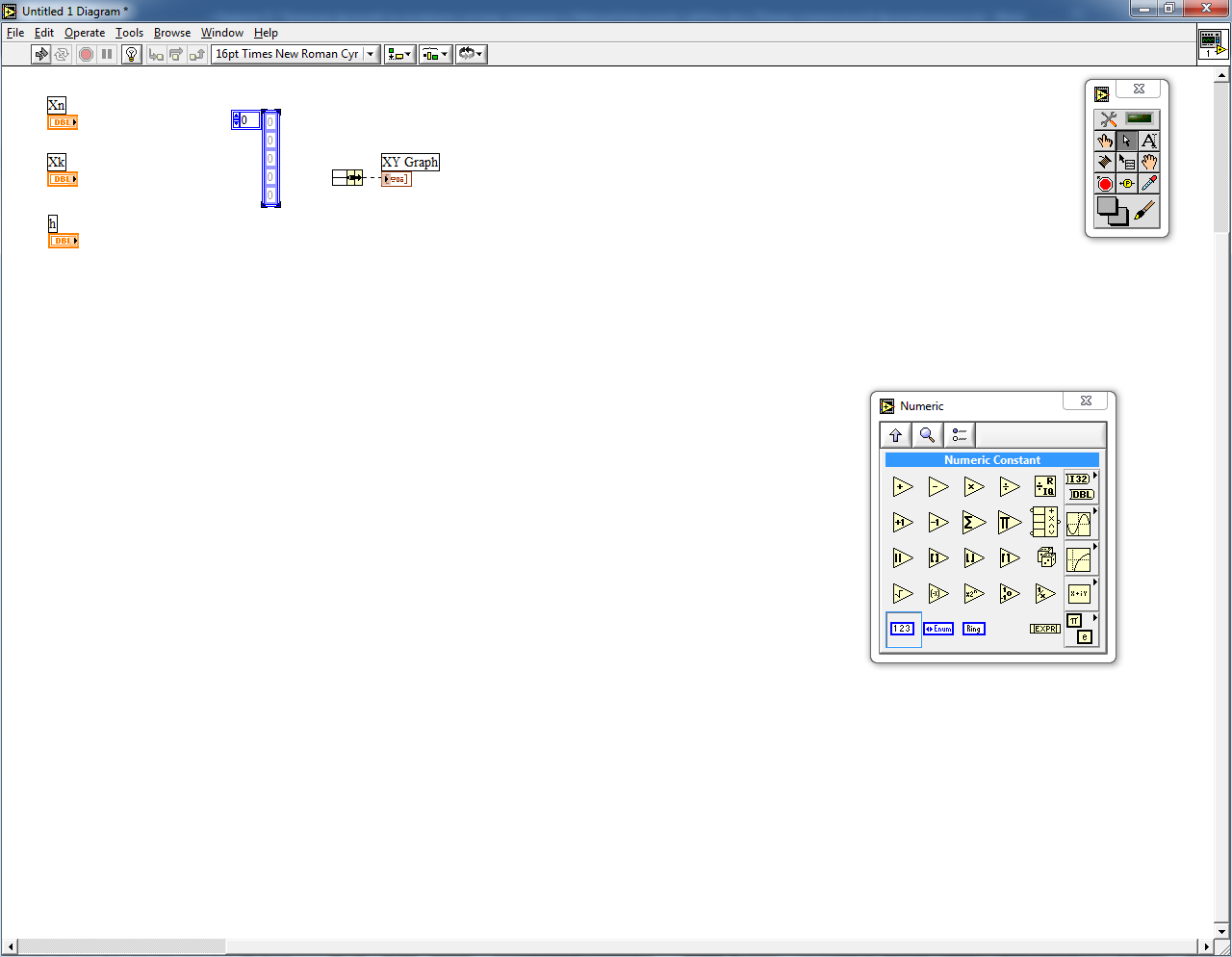


Рисунок 12 – Результат визуального расширения массива констант инструментом позиционирования / изменения размерности / выбора (*Position/Size/Select*)

Далее заполним массив целочисленными элементами от «0» до «4». Подразумевается, что это будут абсциссы будущих точек (Рисунок 13).

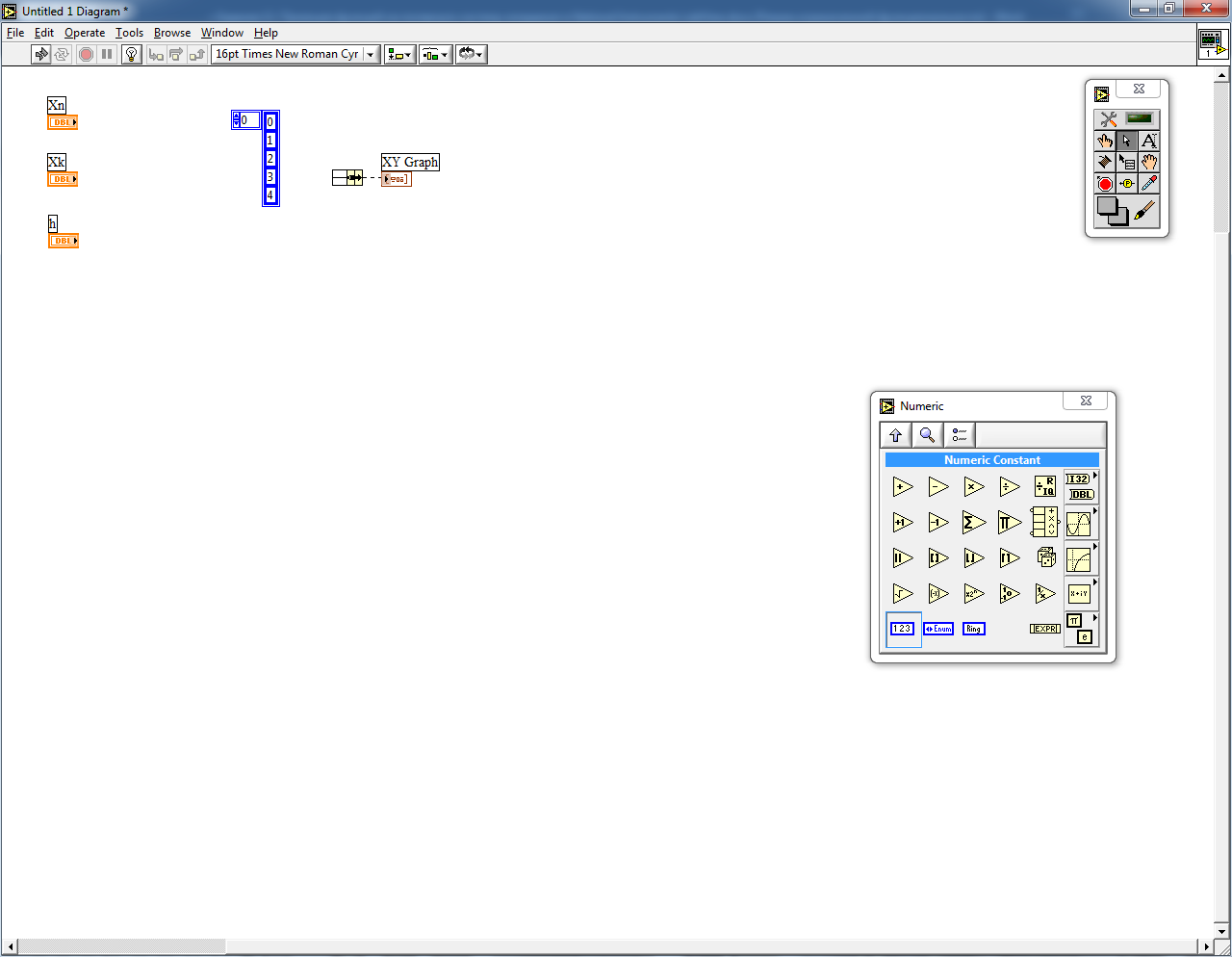


Рисунок 13 – Заполнение вручную элементов целочисленного массива

Скопируем и вставим на блок-диаграмму дубликат созданного массива точек – это действие значительно ускорит подготовку аналогичной структуры под ординаты будущих точек (Рисунок 14).

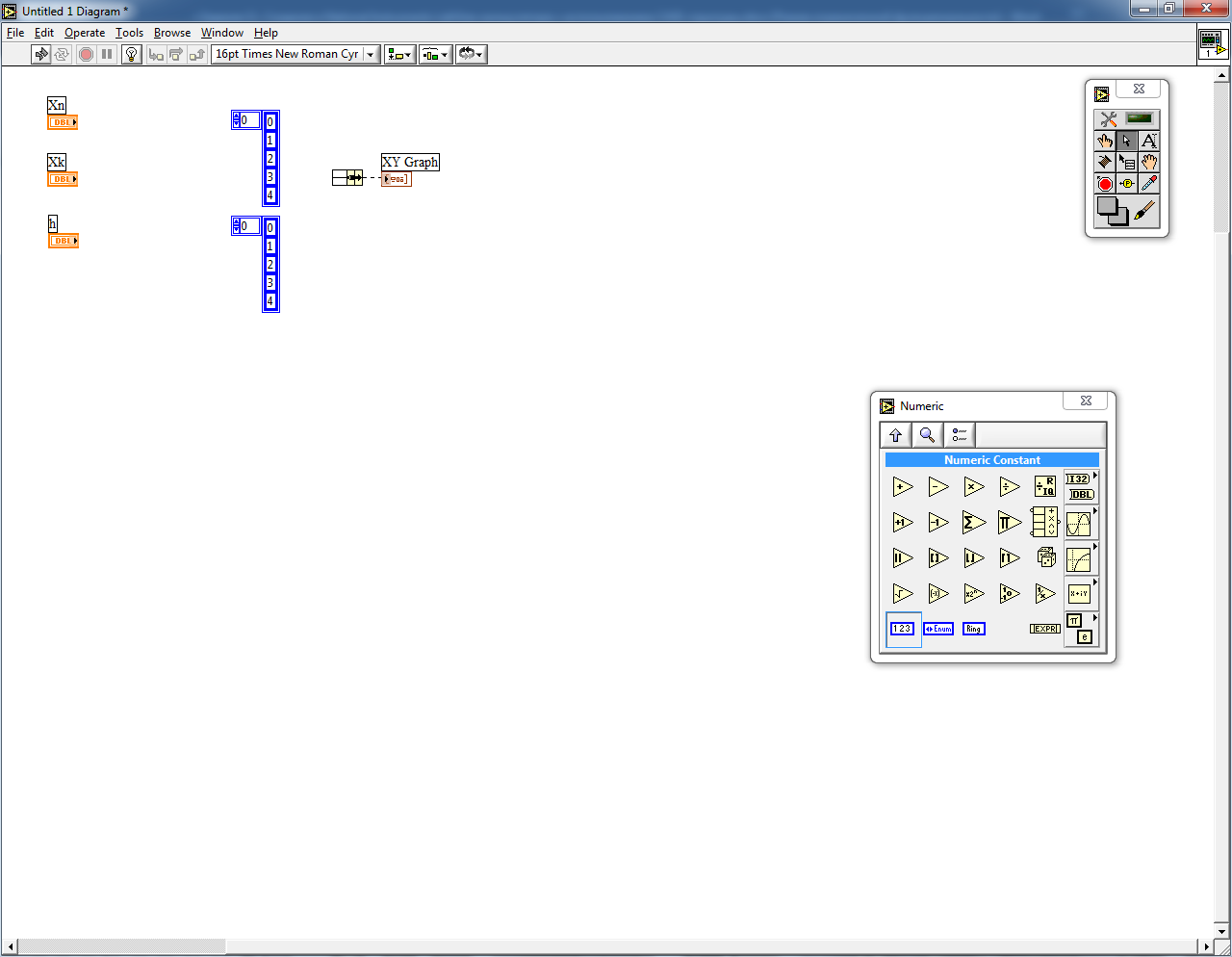


Рисунок 14 – Копирование целочисленного массива

Несколько расширим размер ячеек массива с использованием всё того же инструмента позиционирования / изменения размерности / выбора для корректного отображения двузначных чисел и введём вручную квадраты значений, записанных в массиве абсцисс – это и будут значения ординат (Рисунок 15). Итого рассматривается построение фрагмента параболы.

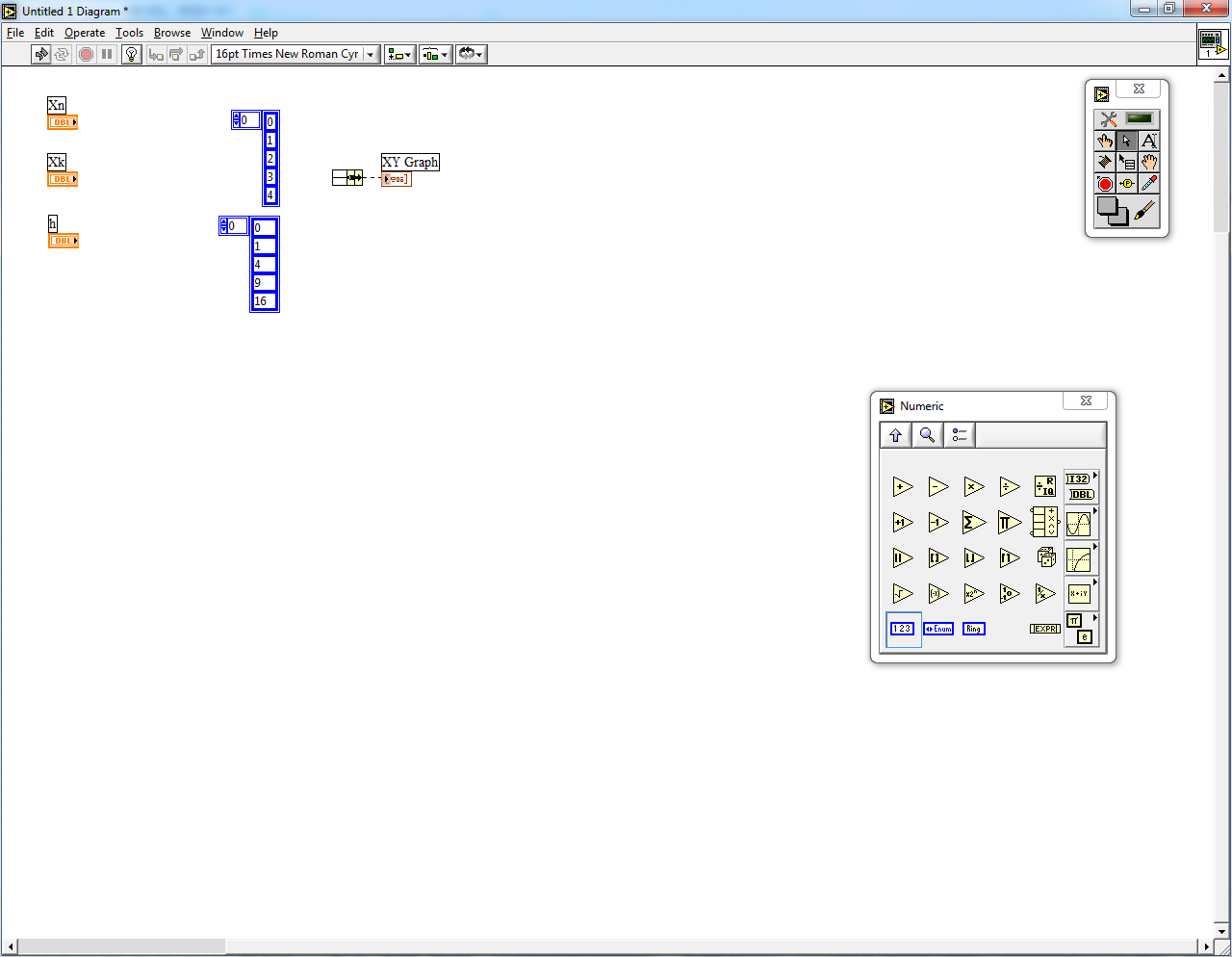


Рисунок 15 – Заполнение вручную продублированного целочисленного массива значениями основного целочисленного массива, возведёнными в квадрат

Далее массив абсцисс подключается к первому входу элемента сцепления, а массив ординат – ко второму. На этом кластер «точка» укомплектован и на выходе элемента сцепления имеется массив точек, достаточный для построения графика функциональной зависимости в соответствующей этому области (Рисунок 16). В таком состоянии виртуальный прибор уже может быть запущен на исполнение. Проверим его работоспособность.

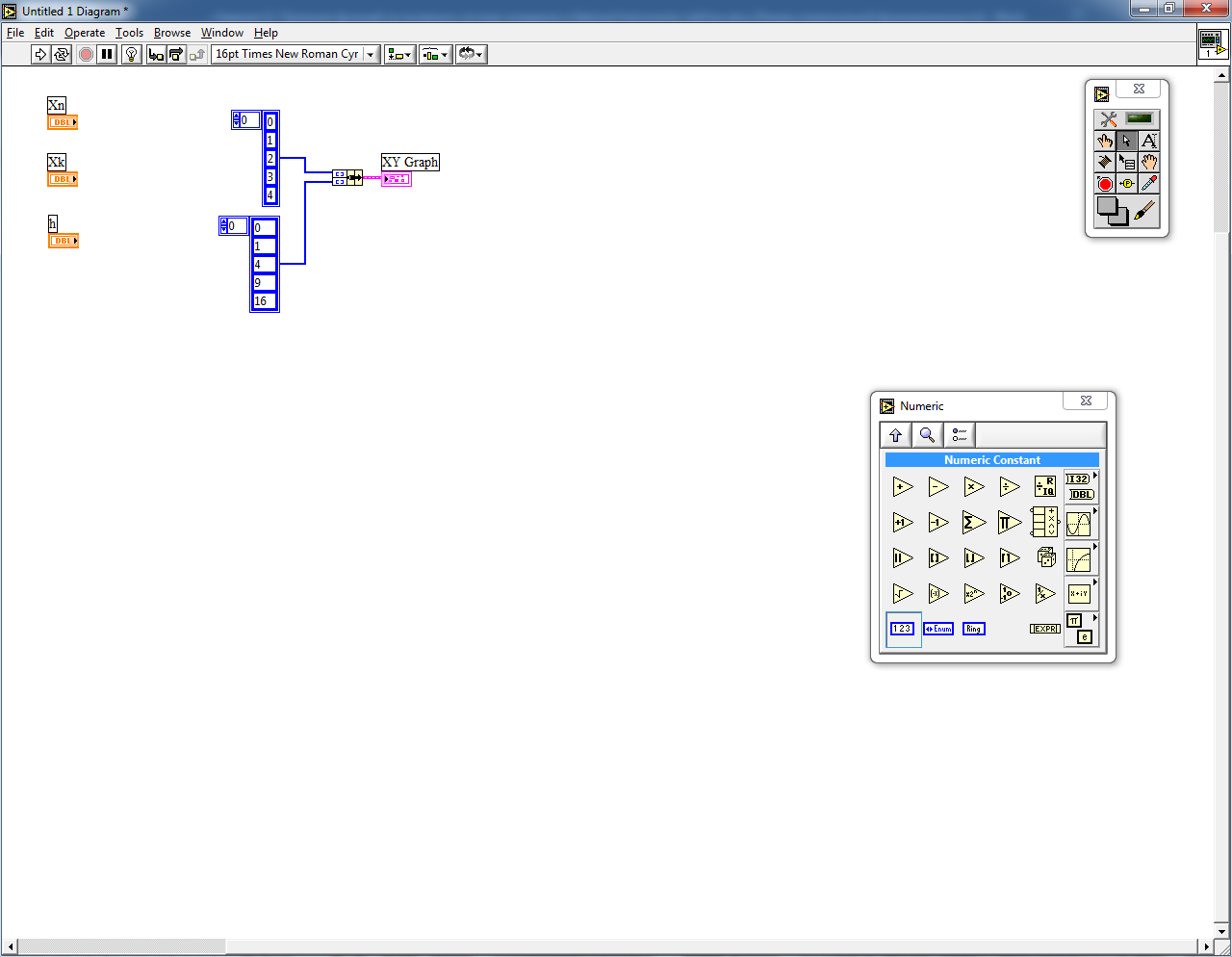


Рисунок 16 – Подключение целочисленных массивов к элементу сцепления значений в кластер

В результате запуска на однократное исполнение виртуальный прибор отвечает отображением в области графика функциональных зависимостей точек, соединённых между собой линиями – этот вид отображения доступен по умолчанию (Рисунок 17).

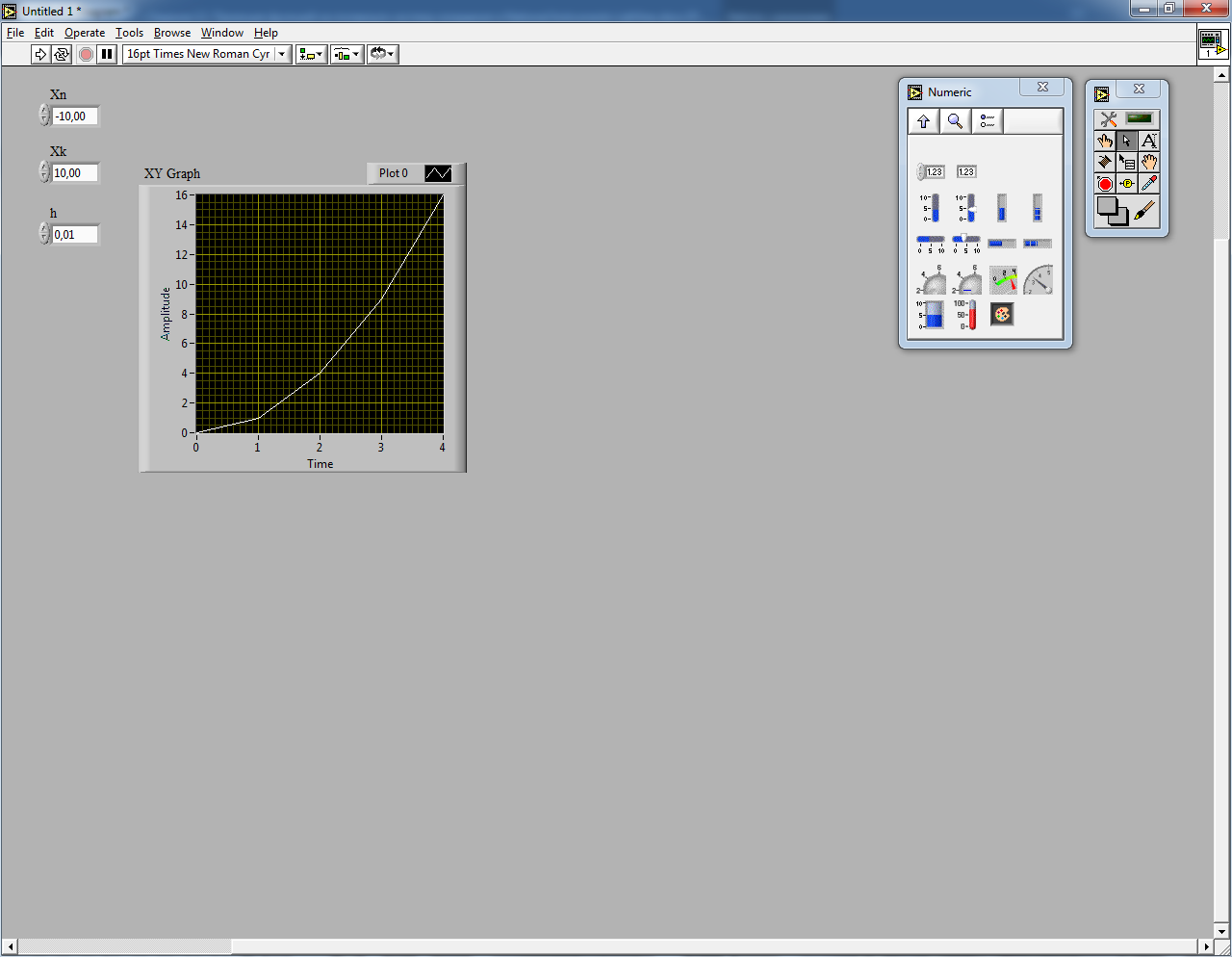


Рисунок 17 – Результат запуска созданного виртуального прибора

Начинающему пользователю может показаться не очевидным наличие точек на данном графике, потому дополнительно выполним изменения в настройках области для построения графиков функциональных зависимостей. Разные фрагменты этой области содержат своё собственное контекстное меню, вызываемое правой кнопкой мыши. Рассмотрим сначала настройку трассировки, расположенной в правой верхней части интерфейсного элемента. Первая же строчка контекстного меню (*Common Plots*) позволяет настроить одно из типовых отображений точек графика: линии, точки, столбы и другие (Рисунок 18). Переключим режим отображения на точки.

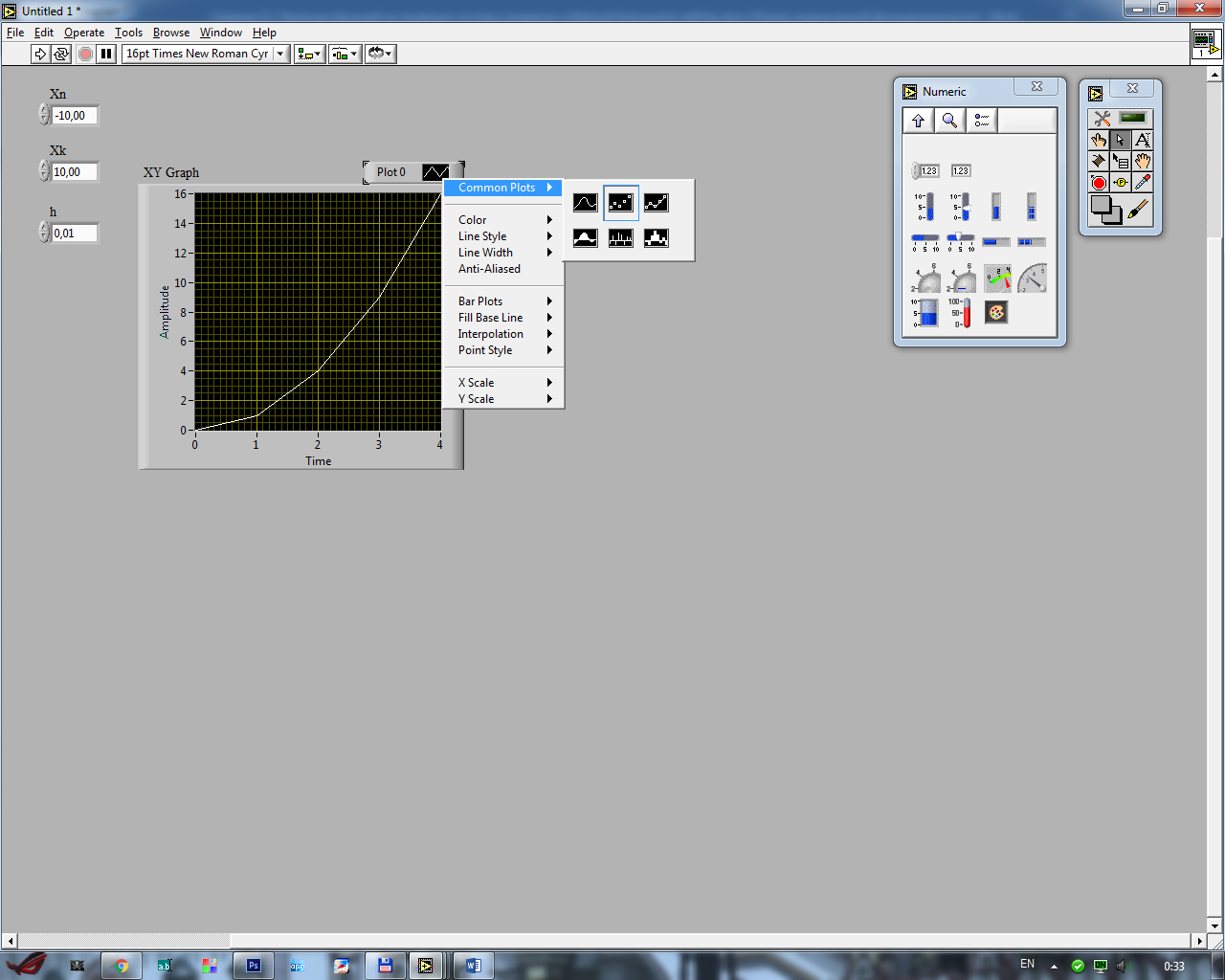


Рисунок 18 – Переход к настройке отображения точек / линий в области для построения графиков функциональных зависимостей

В результате изменения режима область для построения графиков функциональных зависимостей переформатируется (Рисунок 19).

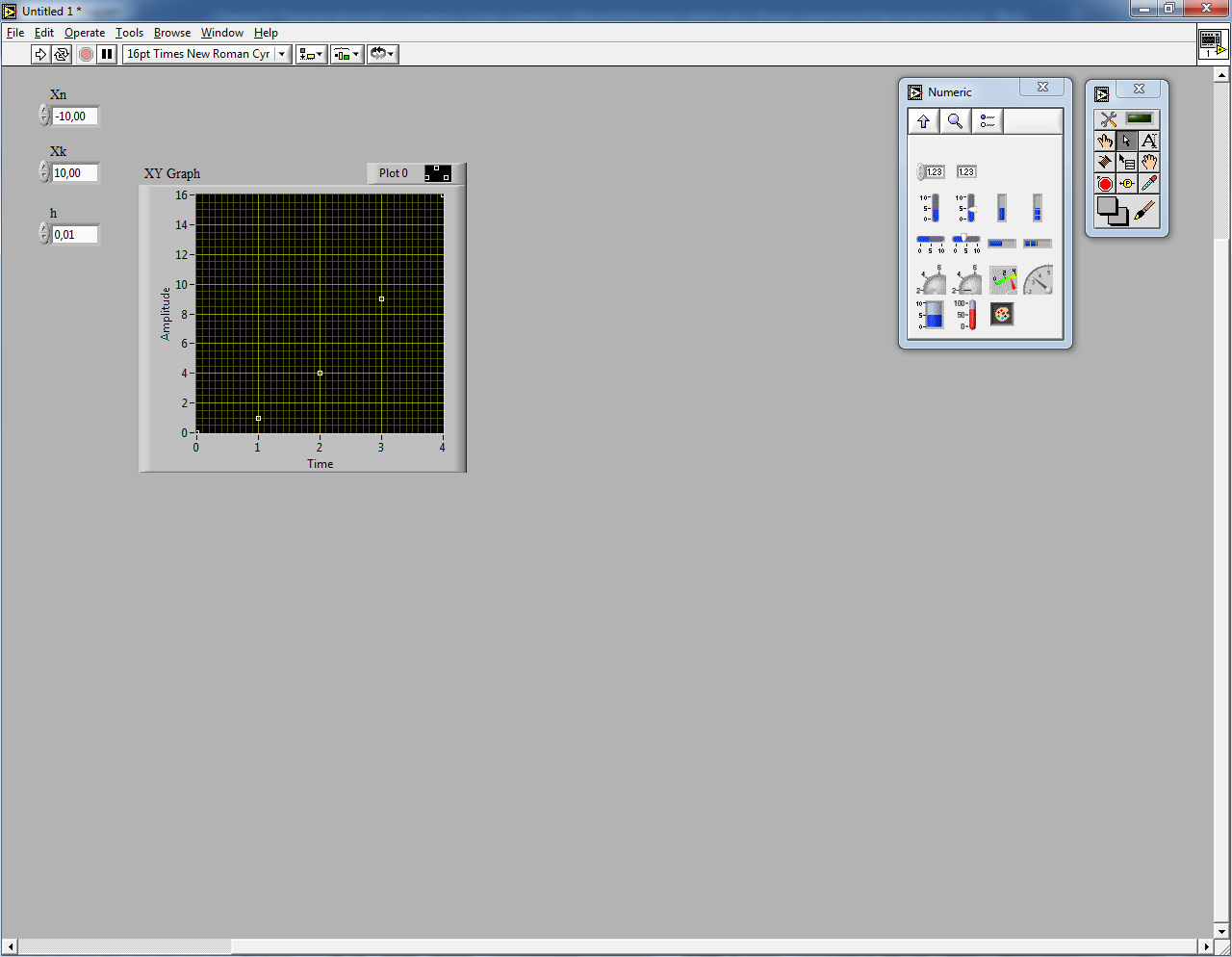


Рисунок 19 – Выбран вариант отображения только точек без соединяющих их линий

«Выколотые» точки могут быть не достаточно показательными и мелкими, потому необходимо также научиться настраивать и корректное представление точек в области построения графиков функциональных зависимостей. В том же контекстном меню раздел стилизации точек («*Point Style*») позволяет переопределить их отображение (Рисунок 20).

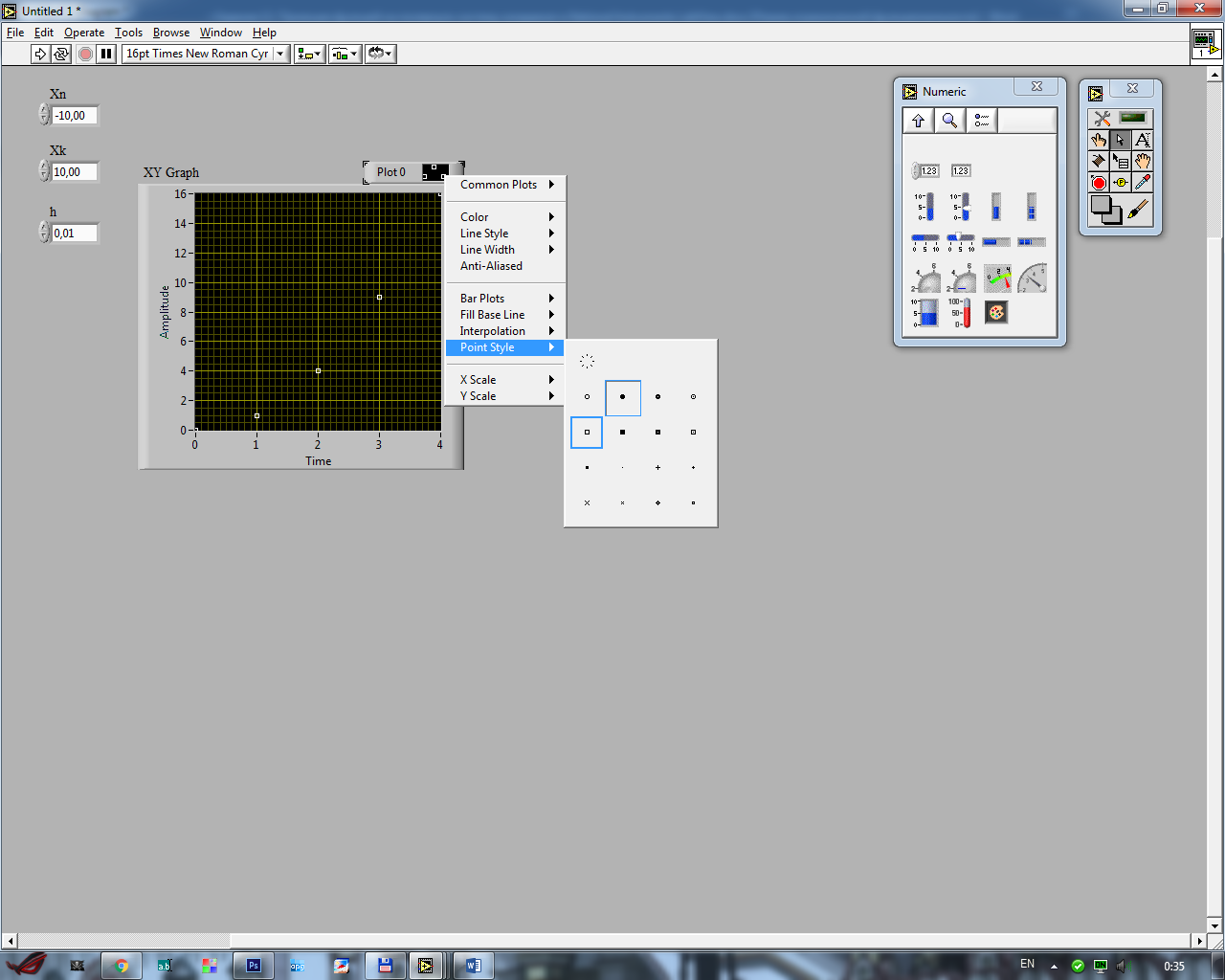


Рисунок 20 – Переход к настройке отображения вида точек в области построения графиков функциональных зависимостей

Результат изменения вида точек представлен на Рисунке 21.

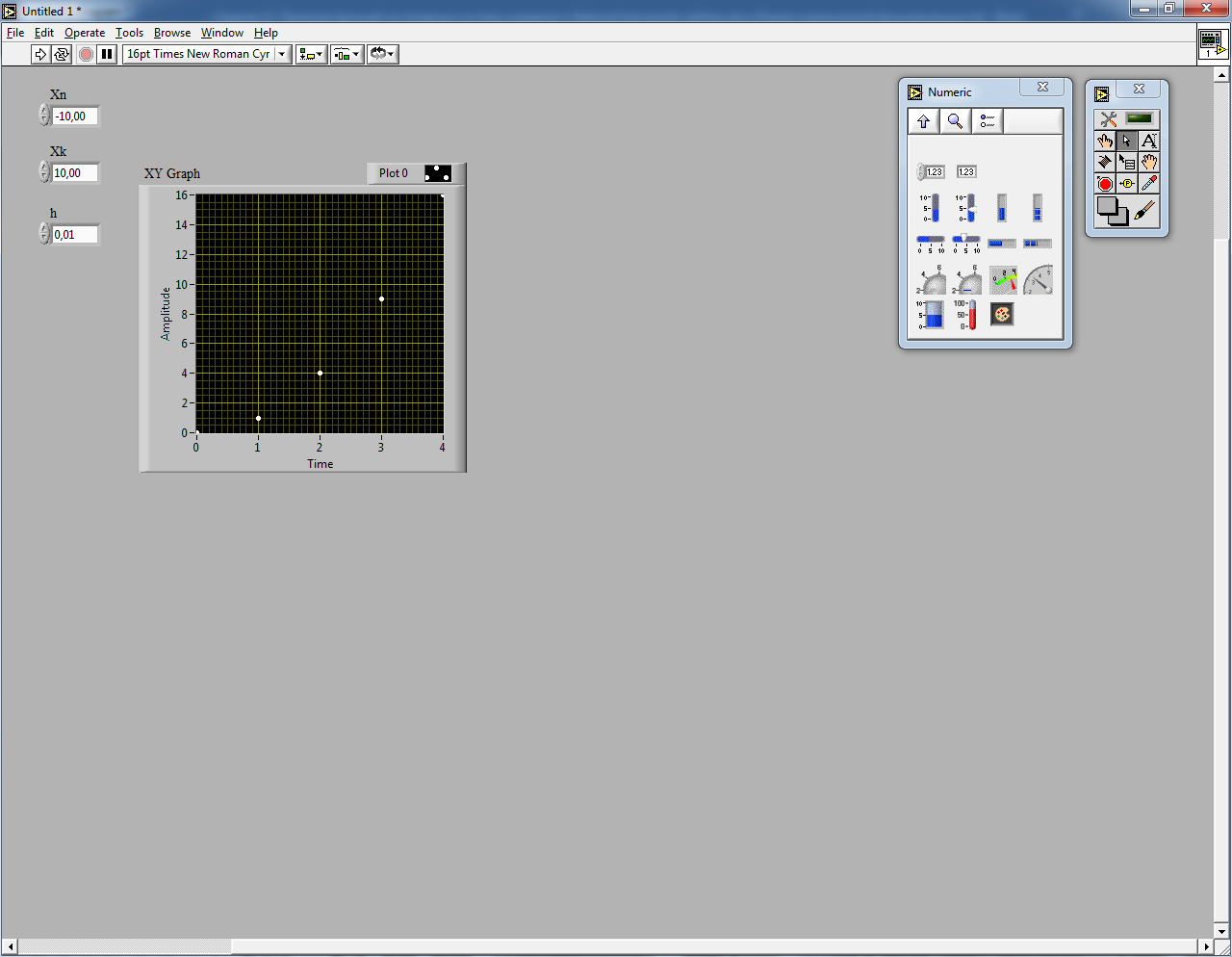


Рисунок 21 – Выбран вариант отображения круглых, заполненных точек

Размер точки напрямую зависит от размера линии, потому в том же контекстном меню, обратившись к изменению толщины линии («*Line Width*»), можно изменить размер точек (Рисунок 22).

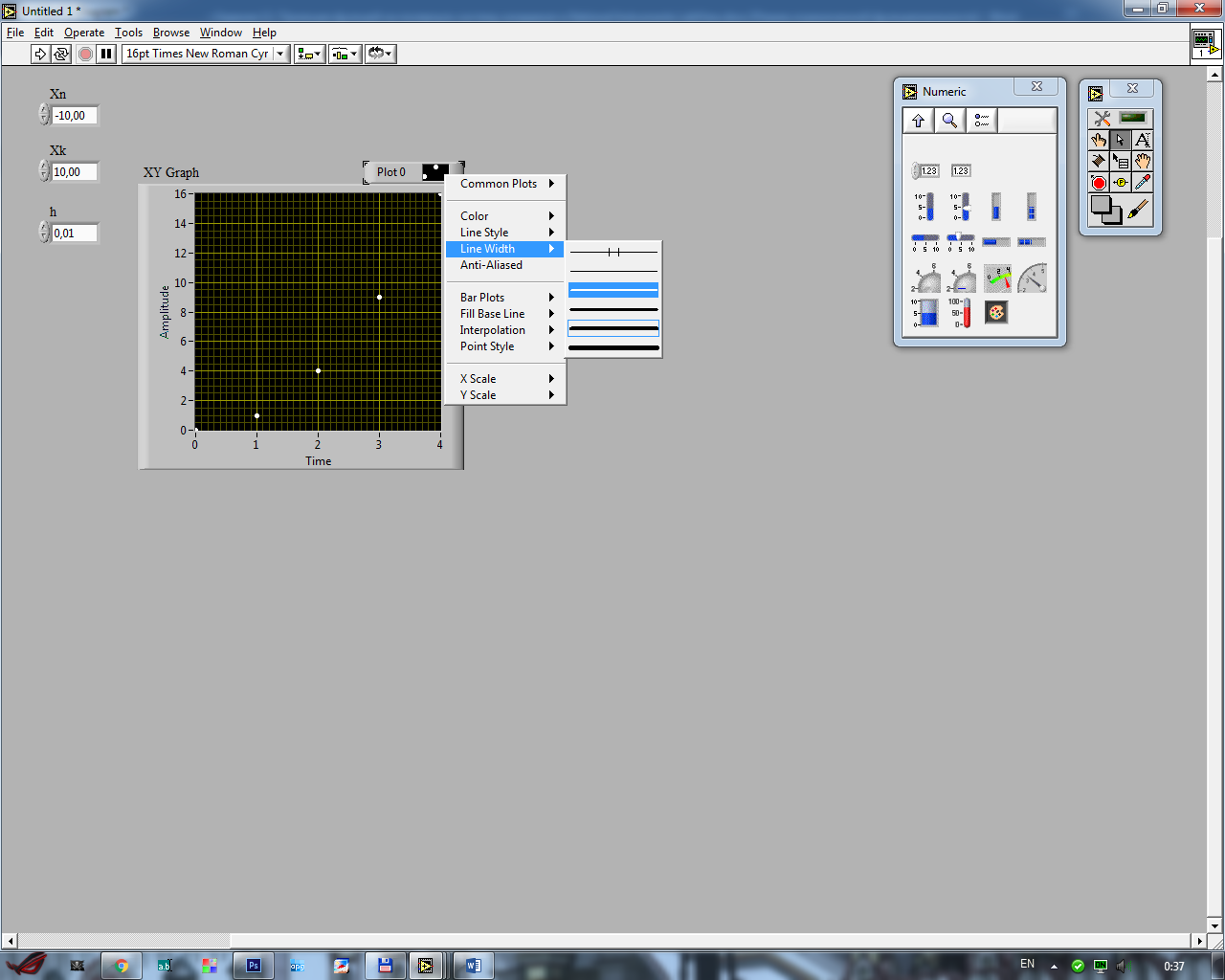


Рисунок 22 – Переход к настройке толщины линий и, соответственно, точек в области для построения графиков функциональных зависимостей

Результат изменения размера точек представлен на Рисунке 23.

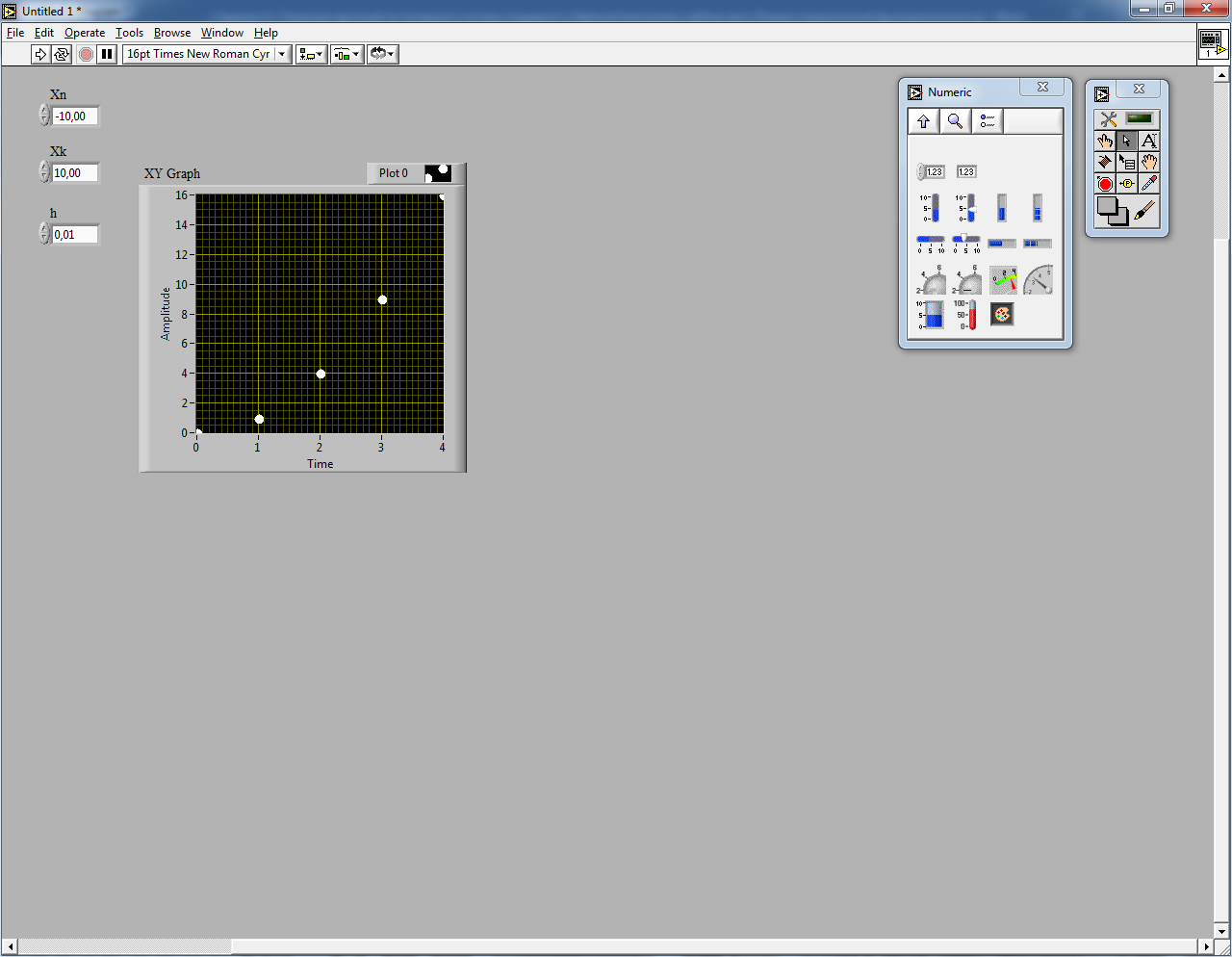


Рисунок 23 – Результат увеличения размера точек в области для построения графиков функциональных зависимостей

Линии и точки могут нормально существовать в рамках одной трассировки, но отображаться они будут одним и тем же цветом. Для отображения точек и линий различными цветами рекомендуется одни и те же данные подавать на различные трассировки. Рассмотрим такой пример, вернёмся к блок-диаграмме, продублируем элемент сцепления и информацию, подаваемую на его входы (Рисунок 24).

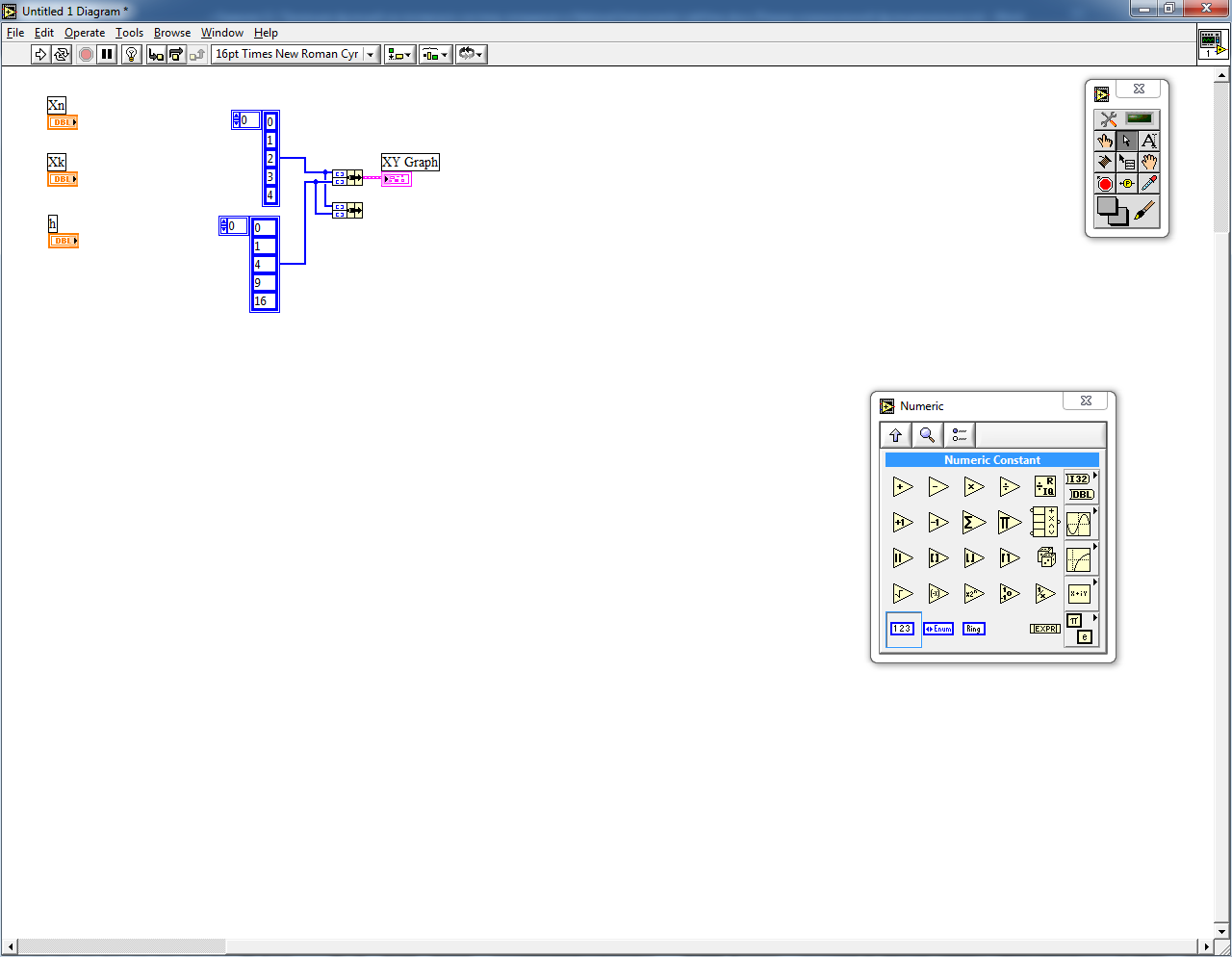


Рисунок 24 – Создание дубликата элемента сцепления значений в кластер и подача на его входы тех же значений

Как было ранее отмечено, область для построения графиков функциональных зависимостей принимает на вход массивы точек. Потому для работы с различными трассировками необходимо промежуточное звено между элементами «*Bundle*» и «*XY Graph*». Таковым звеном является конструктор массива (*Build Array*). Расположение этого элемента в перечне функций, размещаемых на блок-диаграмме, представлено на Рисунке 25.

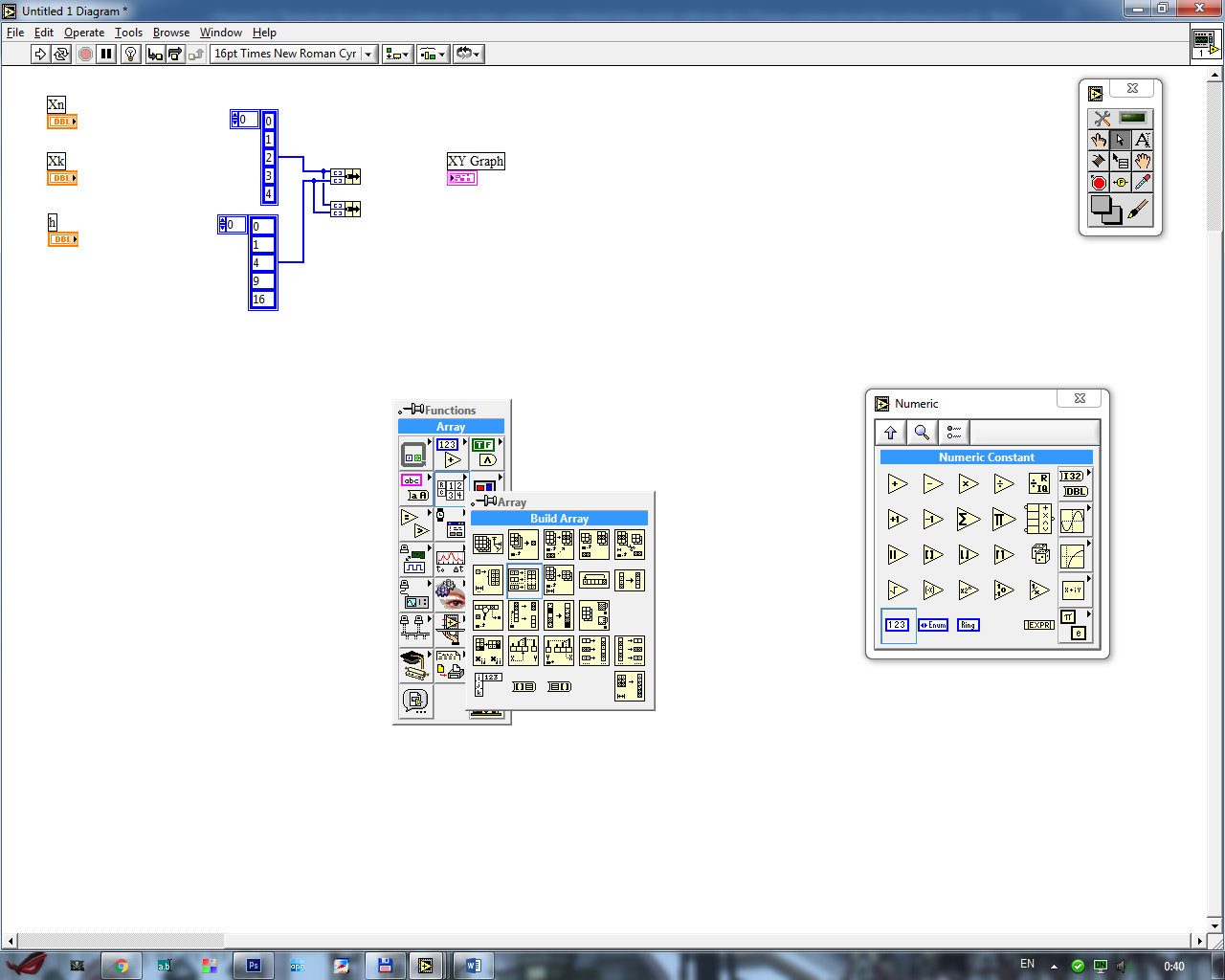


Рисунок 25 – Маркировка элемента построения массива (*Build Array*) в разделе массивов (*Array*) перечня функций, размещаемых на блок-диаграмме

Разорвём существующую связь, ведущую к *XY Graph*, и на её месте установим *Build Array*. По умолчанию этот элемент содержит один вход и один выход (Рисунок 26), однако он может и должен быть модифицирован при использовании инструмента для позиционирования / изменения размерности / выбора.

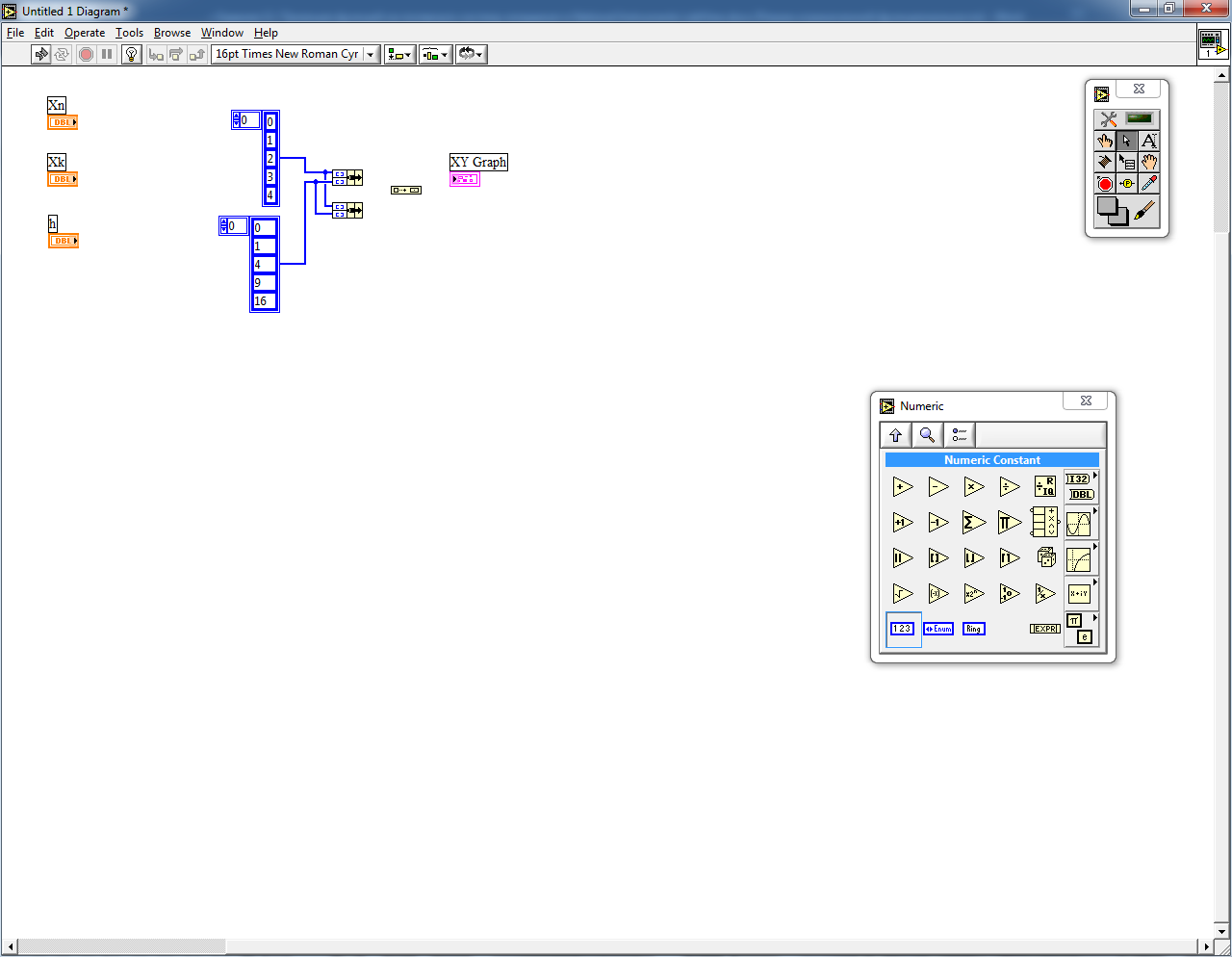


Рисунок 26 – Размещение элемента построения массива на блок-диаграмме

Для модификации элемента достаточно «схватить» его за нижнюю границу и вытянуть на требуемое количество позиций (по количеству используемых трассировок). В рассматриваемом простейшем случае необходимо иметь в наличии только две трассировки. После модификации на каждый из входов конструктора массива подаются выходы с каждого элемента сцепления. Сигнал от конструктора массива подаётся на вход *XY Graph* (Рисунок 27).

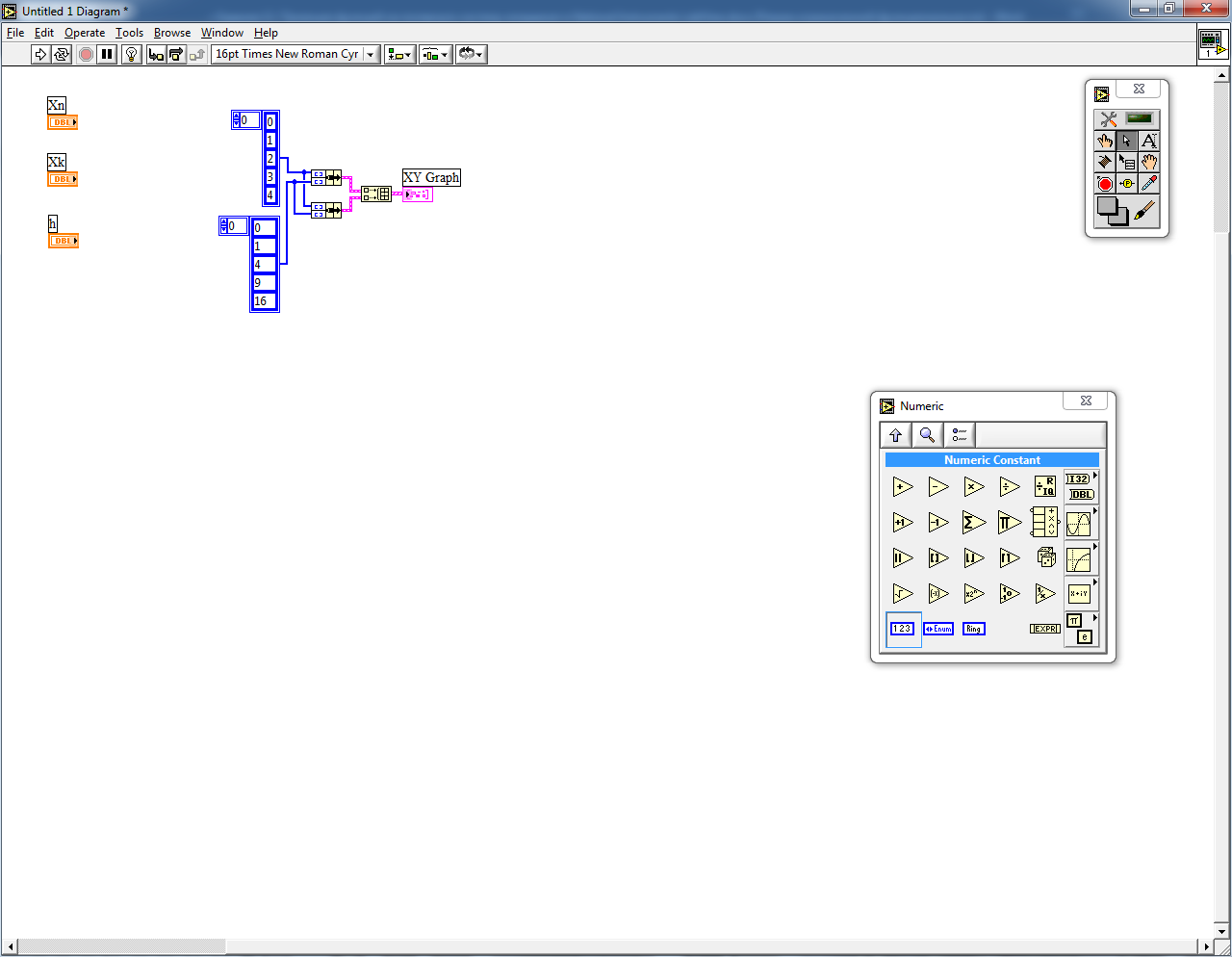


Рисунок 27 – Результат объединения двух кластеров в один массив, передаваемый в область для построения графиков функциональных зависимостей

В результате произведённых изменений на блок-диаграмме после запуска виртуального прибора на однократное исполнение можно увидеть и белые точки (первая трассировка), и красную линию (вторая трассировка).

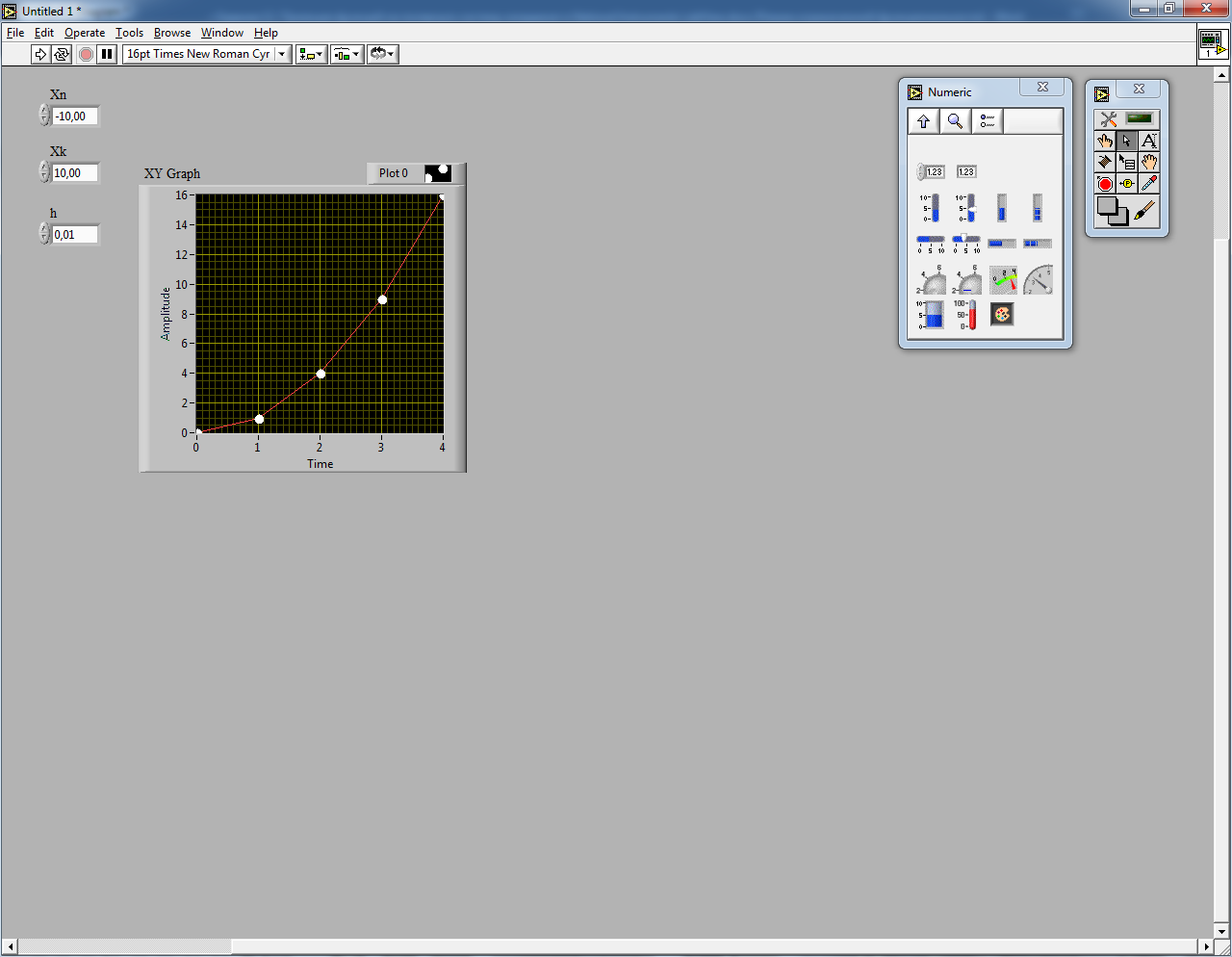


Рисунок 28 – Результат построения одних и тех же точек по различным трассировкам с различными настройками

Для настройки отображения второй трассировки необходимо при выбранном инструменте позиционирования / изменения размерности / выбора «ухватиться» за правый верхний угол фрагмента трассировки, принадлежащего области для построения графиков функциональных зависимостей и вытянуть вверх на нужное количество позиций, соответствующих количеству трассировок (Рисунок 29).

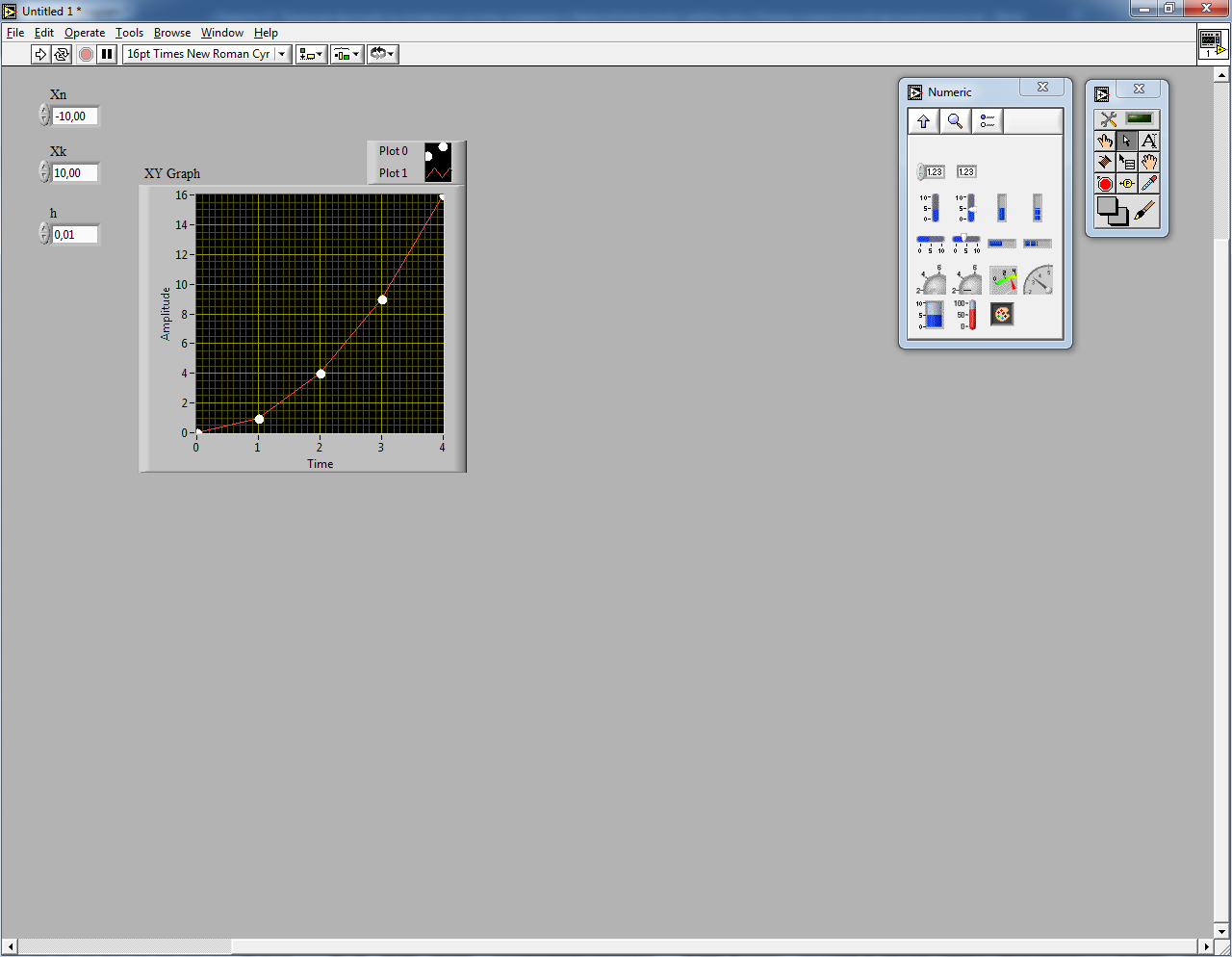


Рисунок 29 – Расширение списка трассировок для обеспечения возможности редактирования второй трассировки

И для закрепления изученного материала разместим в области для построения графиков функциональных зависимостей ещё одну трассировку, которой будет соответствовать смещённый параллельным переносом по оси ординат вверх график, основанный на тех же массивах данных. Как известно, сложение массива с константой даёт в результате массив той же размерности, что и у исходного массива, каждый элемент которого увеличен на значение этой константы. Воспользуемся указанным выше правилом и дополним существующий код на блок-диаграмме ещё одним сцеплением данных и ещё одной размерностью конструктора массива (Рисунок 30).

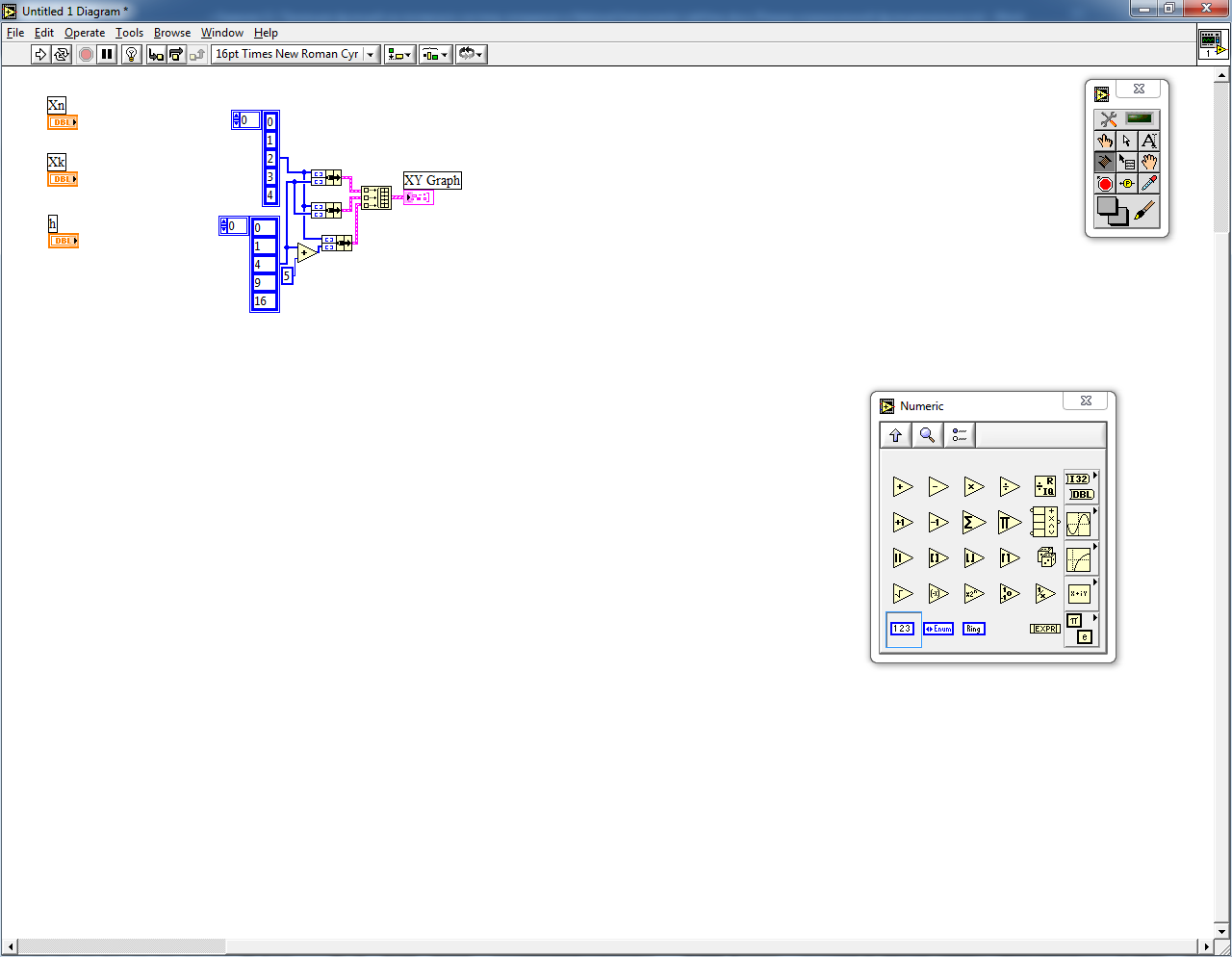


Рисунок 30 – Добавление третьей трассировки со смещением тех же точек на 5 единиц в положительном направлении по оси ординат

В результате получим именно то, что и ожидалось увидеть – линию, смещённую по оси ординат вверх (Рисунок 31).

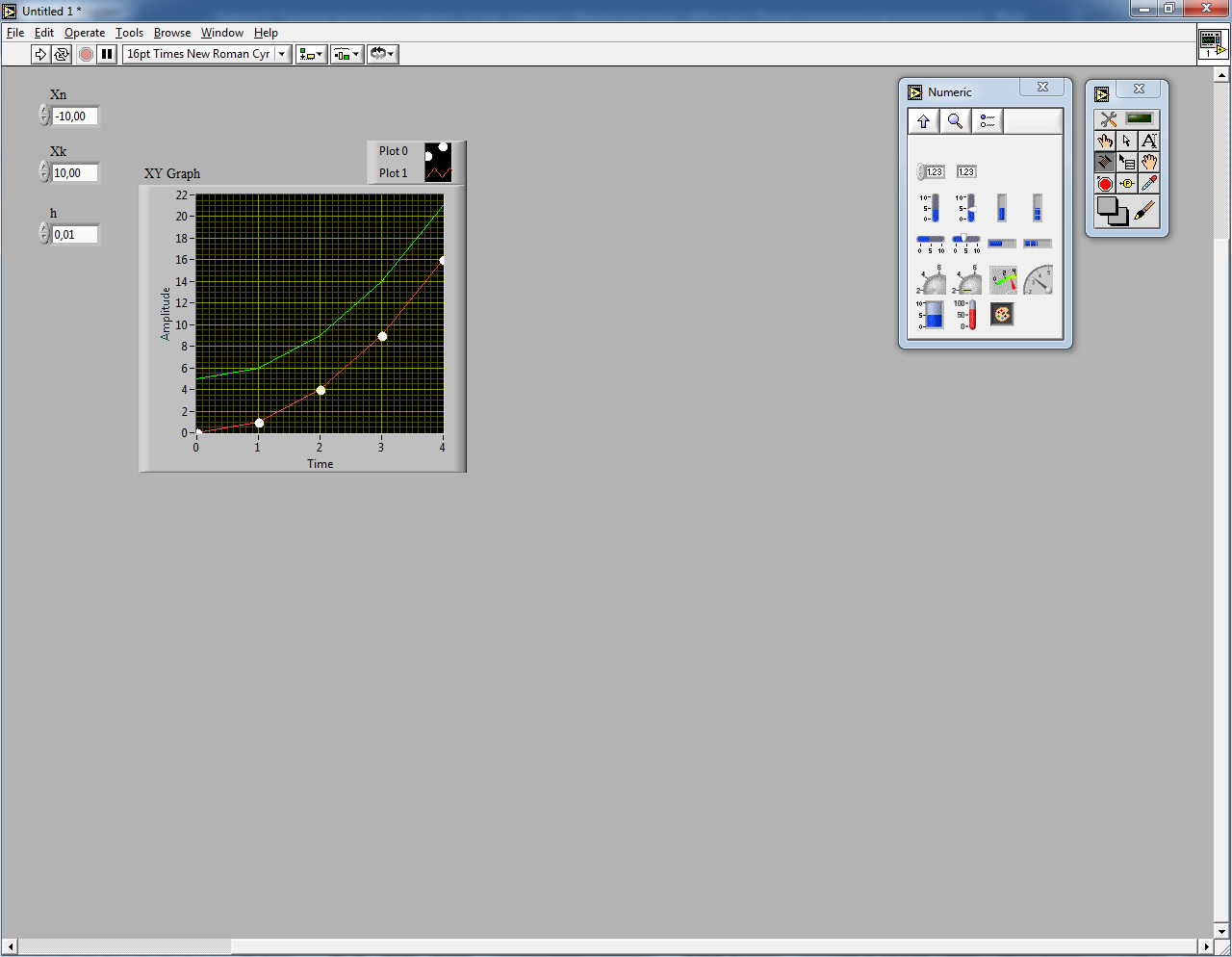


Рисунок 31 – Результат вывода трёх кластеров, объединённых в один массив и выведенных в область для построения графиков функциональных зависимостей

На Рисунке 32 фрагмент области для построения графиков функциональных зависимостей вытянут вверх ещё на одну позицию для отображения нужной (третьей) трассировки.

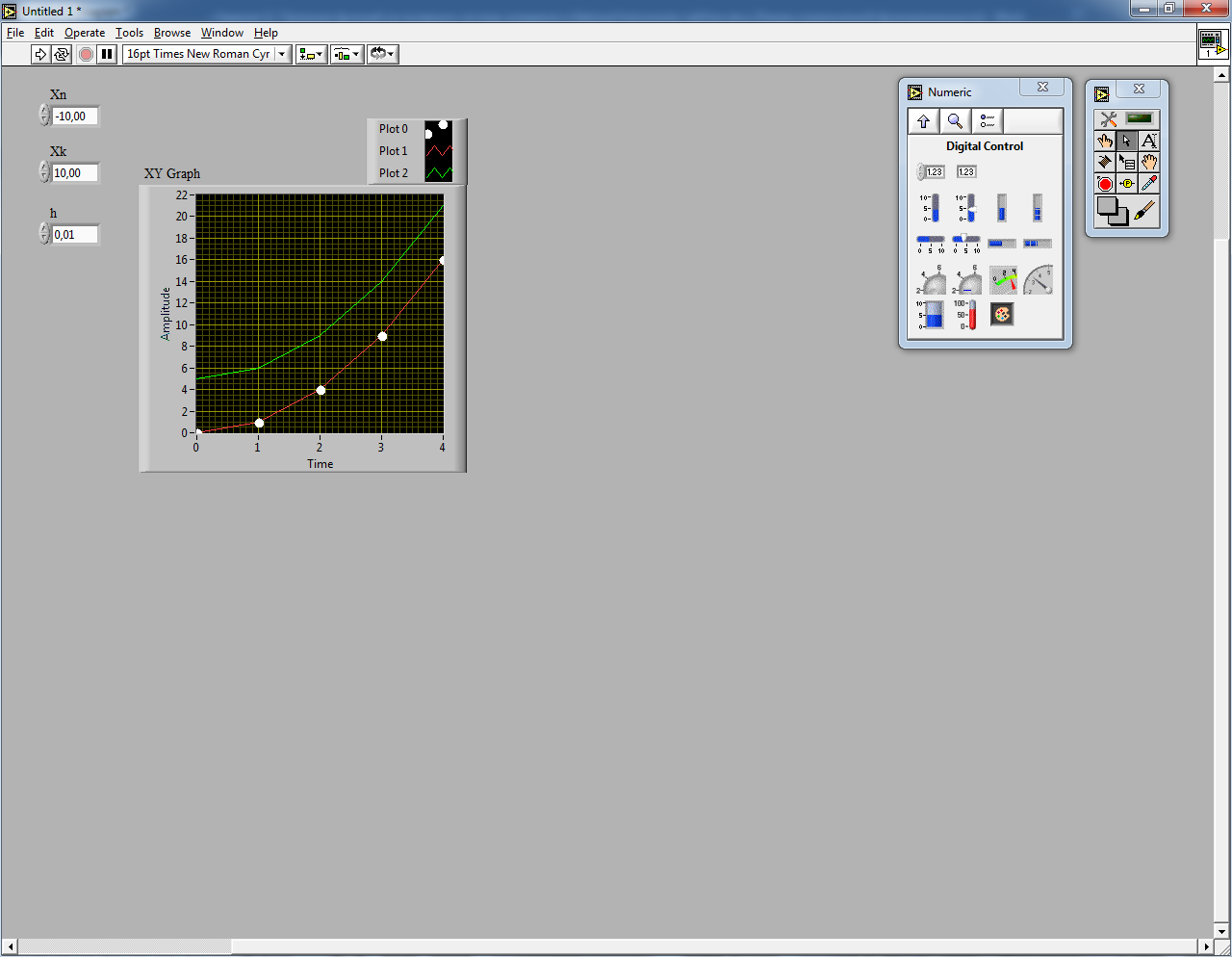


Рисунок 32 – Расширение списка трассировок для обеспечения возможности редактирования третьей трассировки

По аналогии с увеличением размера точек увеличивается и толщина линии (Рисунок 33).

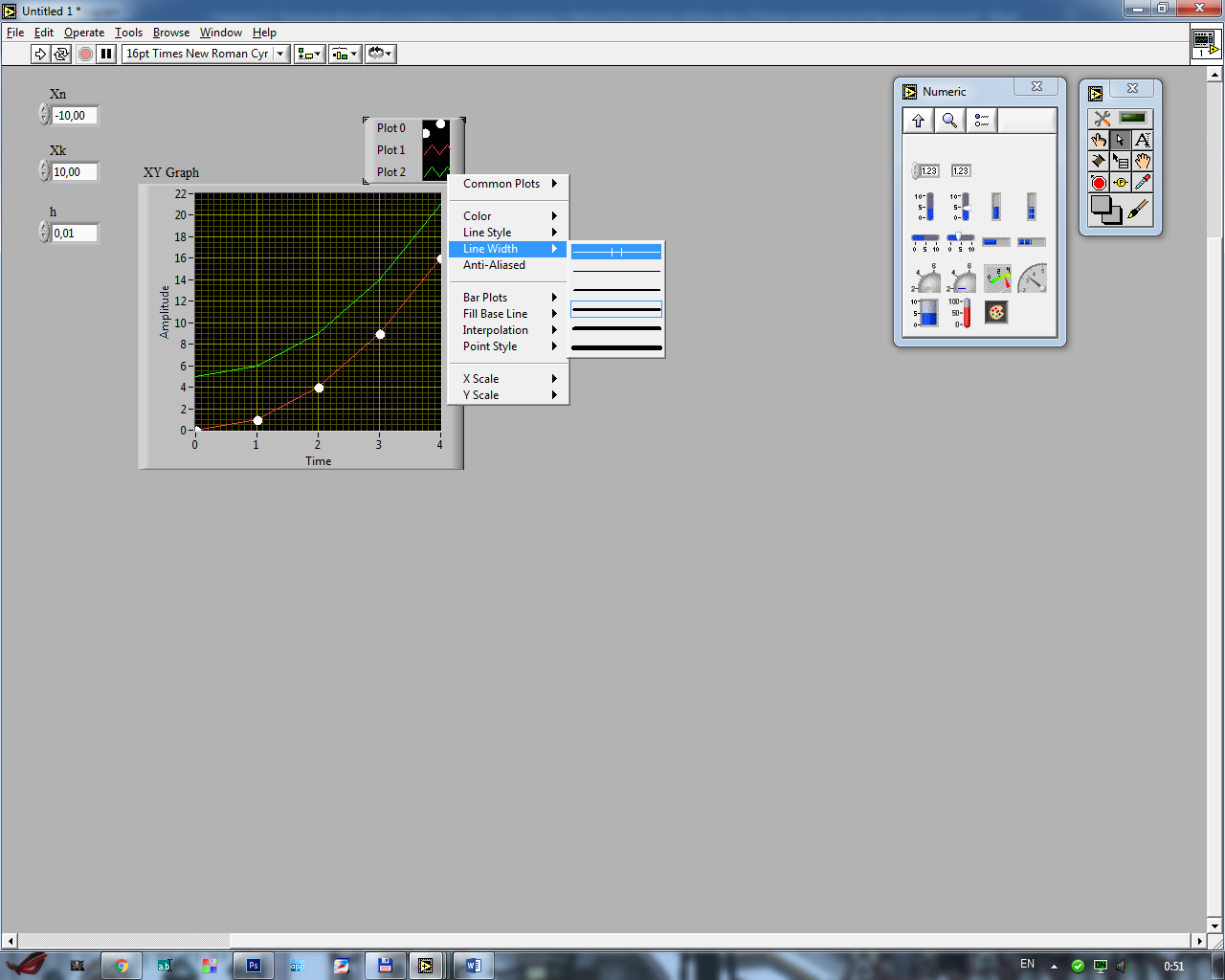


Рисунок 33 – Переход к настройке толщины линии третьей трассировки

Изменение толщины линии позволяет сделать трассировку более контрастной и, вообще говоря, толщина линий трассировки должна преобладать над толщиной линий сетки (Рисунок 34). Негласно существует подобное правило наряду с правилами вежливости программиста.

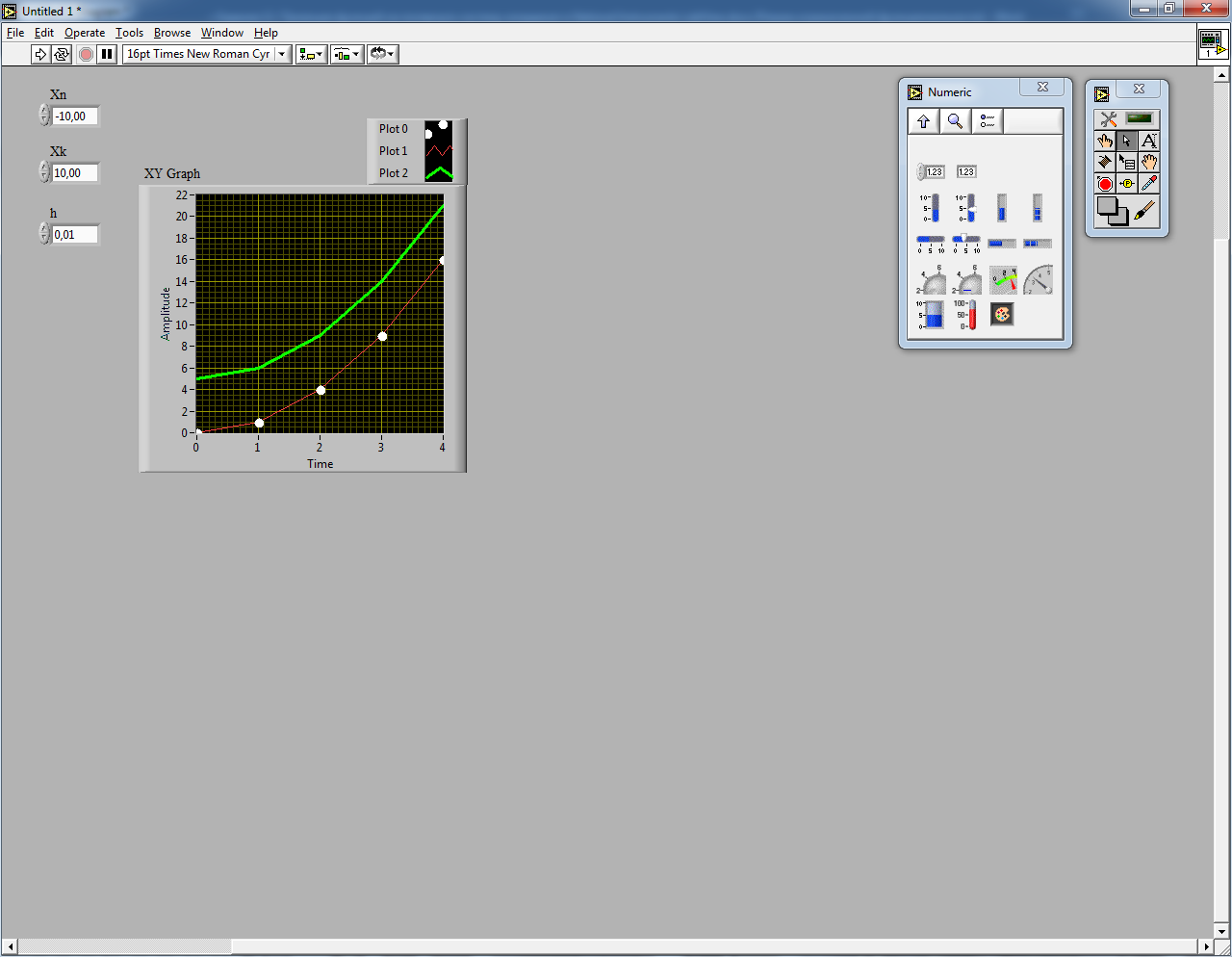


Рисунок 34 – Результат вывода трёх кластеров, объединённых в один массив и выведенных в область для построения графиков функциональных зависимостей с изменёнными настройками третьей трассировки

Подразумевается, что разобравшись со всеми изложенными выше положениями, обучающиеся обладают достаточным количеством сведений для построения графиков периодических функций с развёрткой во времени, а также графиков проекций этих функций на полярную систему координат со смещением точки начала построения в начало координат и сумеют справиться с поставленной задачей.

----------------------------------------------------------------------------------------------------

**Построение графиков функциональных зависимостей в *National Instruments LabView* на базе структуры цикла *While***

Данный раздел посвящён основам работы с циклической структурой *While*. На русский язык *While* переводится «до тех пор, пока». Данный перевод в некоторых контекстах рассмотрения может быть ошибочно воспринят обучающимися, потому для краткости и простоты повествования наименование данной циклической конструкции далее и впредь будет оставлено без перевода. Как правило, циклическая конструкция *While* в некоторых языках программирования называется циклом с предусловием, поскольку служебное слово *While* записывается в начале циклической конструкции, а вслед за ним записывается логическое выражение, определяющее ключевое условие для принятия решения. В иных языках программирования в зависимости от того в начале или в конце тела циклической конструкции расположено служебное слово *While* этот цикл может являться как априорным (с предусловием), так и апостериорным (с постусловием). В *National Instruments LabView* циклическая конструкция *While* функционирует в одном из двух режимов:

- прямом, когда в теле цикла выставлен значок повтора – продолжить, если истина (*Continue If True*);

- инверсном, когда в теле цикла выставлен значок остановки – остановиться, если истина (*Stop If True*).

В данной работе используется прямой режим функционирования.

Итак, вернёмся к исходному состоянию (шаблону-заготовке) виртуального прибора для построения графиков функциональных зависимостей (Рисунки 7, 35).

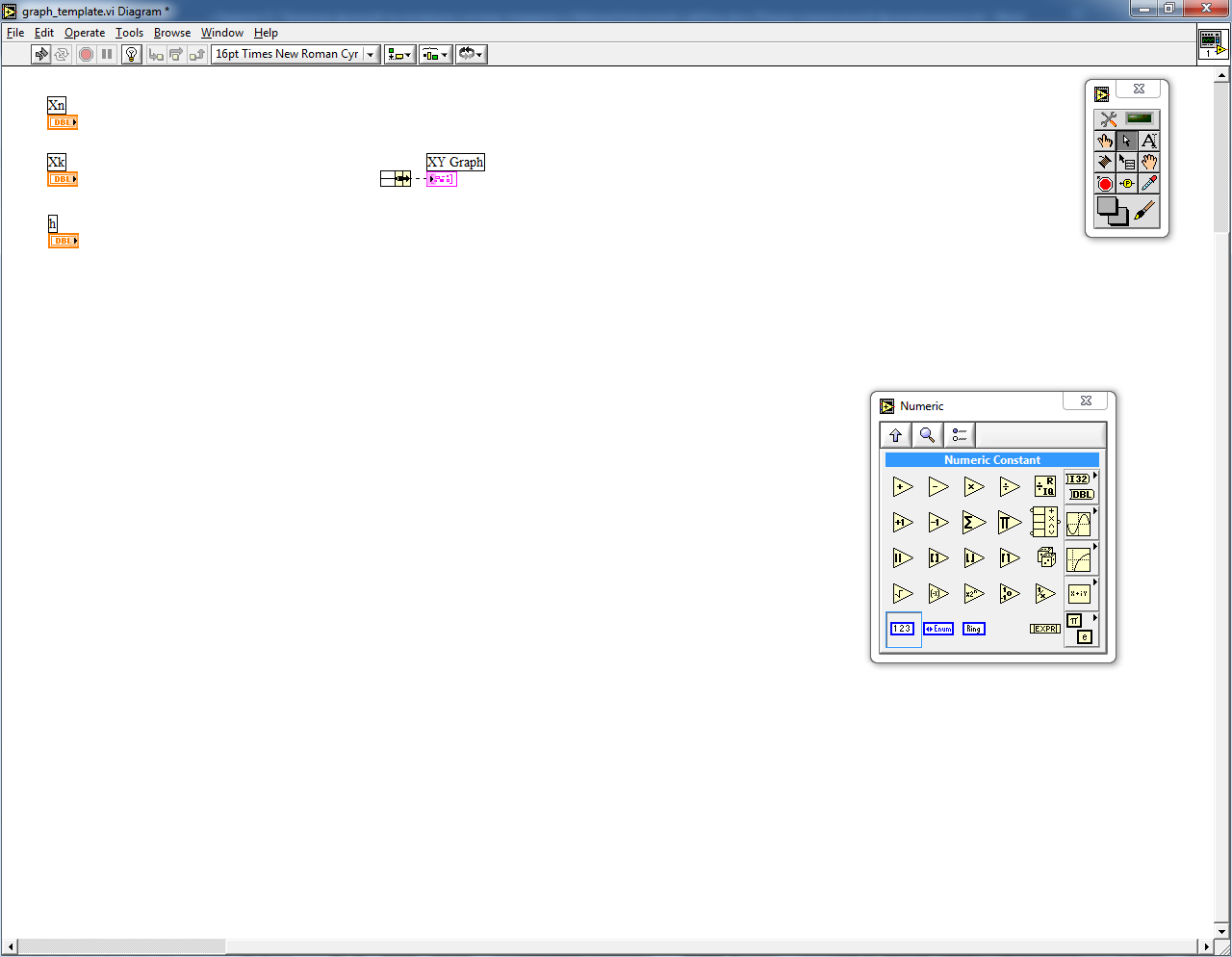


Рисунок 35 – Исходное состояние виртуального прибора для построения графиков функциональных зависимостей

Для расчёта значений функциональной зависимости при различных значениях абсциссы необходимо использовать одну из известных циклических конструкций. Идея расчёта (алгоритм) состоит в следующем:

1. Отсчёт начинается от левой границы диапазона – это первая из имеющихся абсцисс.

2. Значение абсциссы пересчитывается согласно заранее определённым правилам (эти правила и есть функциональная зависимость), в результате чего получается значение ординаты.

3. Абсциссе и ординате ставится в соответствие порядковый номер или позиция в списке – полученные значения сохраняются в некоторой структуре данных или выводятся в таблицу.

4. К абсциссе прибавляется величина шага.

5. В том случае, если новое значение абсциссы не превышает значения правой границы по абсциссам, то происходит возврат к шагу 2, в ином случае цикл прекращает свою работу.

6. Расчётные значения точек выводятся в область построения графика функциональной зависимости.

На Рисунке 36 отмечено расположение циклической структуры *While* в перечне функций, размещаемых на блок-диаграмме.

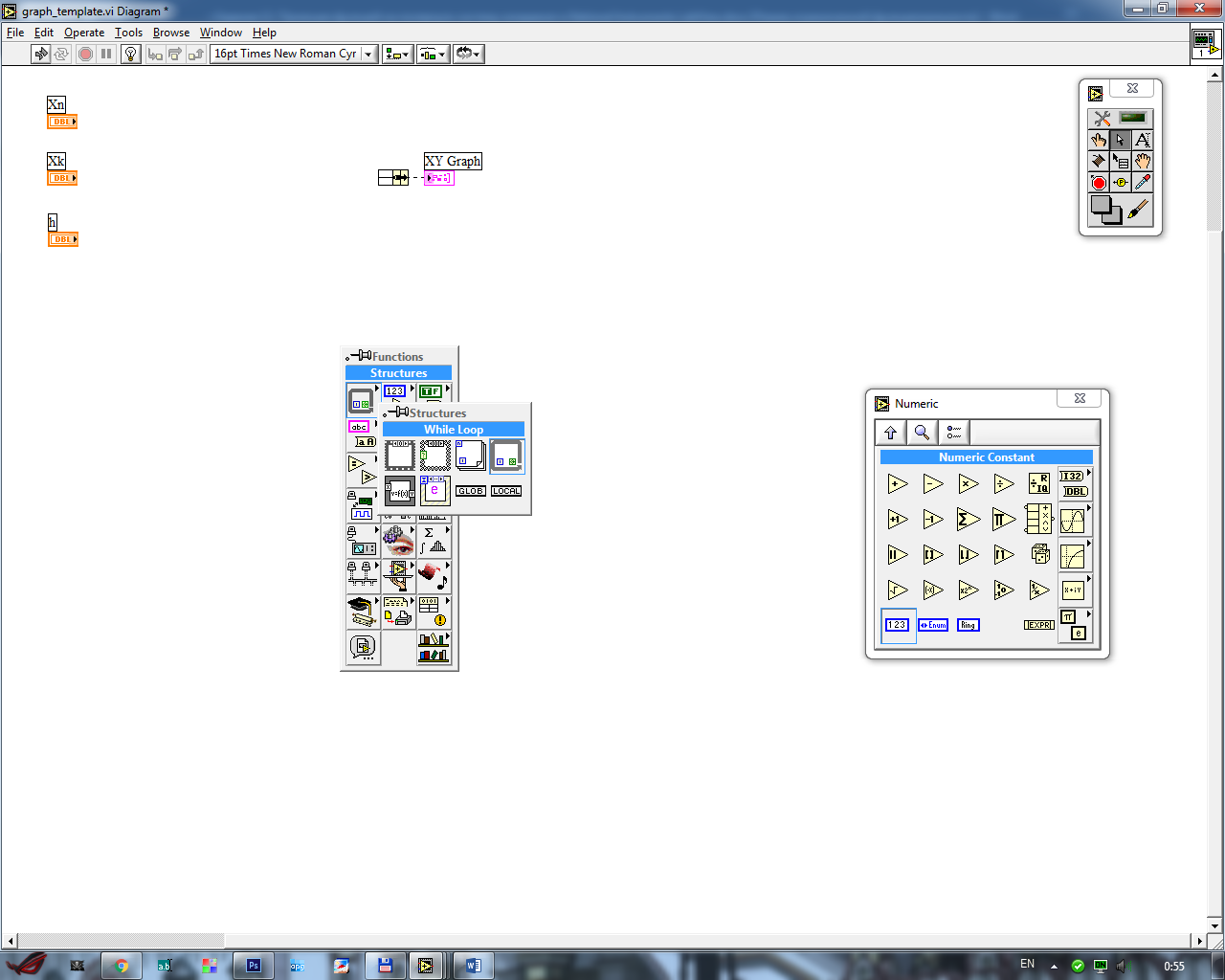


Рисунок 36 – Маркировка структуры цикла *While* (*While Loop*) в разделе структур из перечня функций, размещаемых на блок-диаграмме

Циклическая структура имеет прямоугольную форму, потому согласно принятым для большинства графических редакторов правилам она размечается посредством зажатия и удержания в нажатом состоянии левой кнопки мыши в точке, где запланирован левый верхний угол и отпускания её в точке, где запланирован правый нижний угол. Не возбраняется построение в обратном направлении от правого нижнего угла к левому верхнему (Рисунок 37).

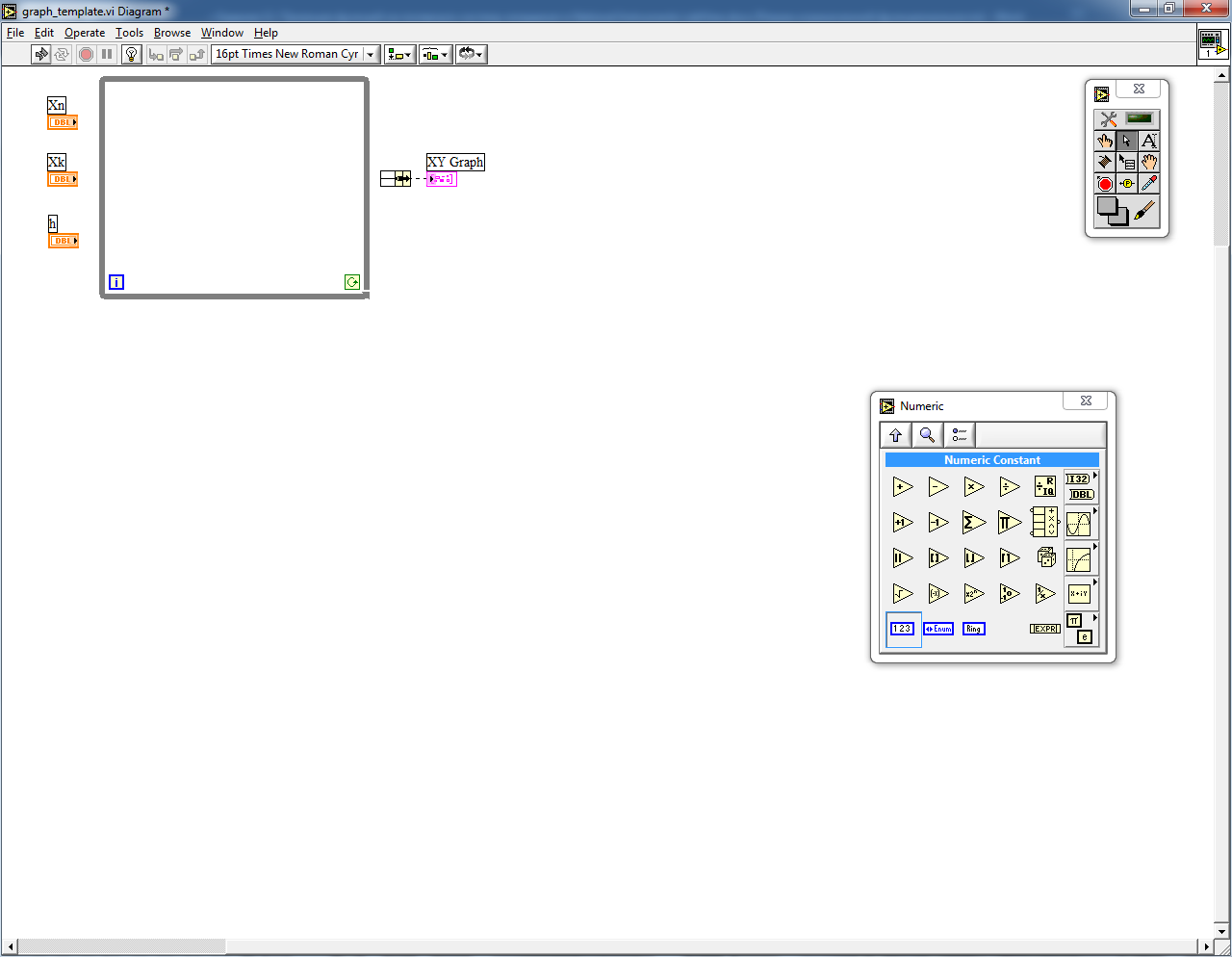


Рисунок 37 – Размещение цикла *While* на блок-диаграмме

Для переноса на следующую итерацию значения, рассчитанного на текущей итерации, в циклических конструкциях используются сдвиговые регистры (*Shift Register*). Элемент добавляется через контекстное меню, вызванное на границе циклической конструкции (Рисунок 38).

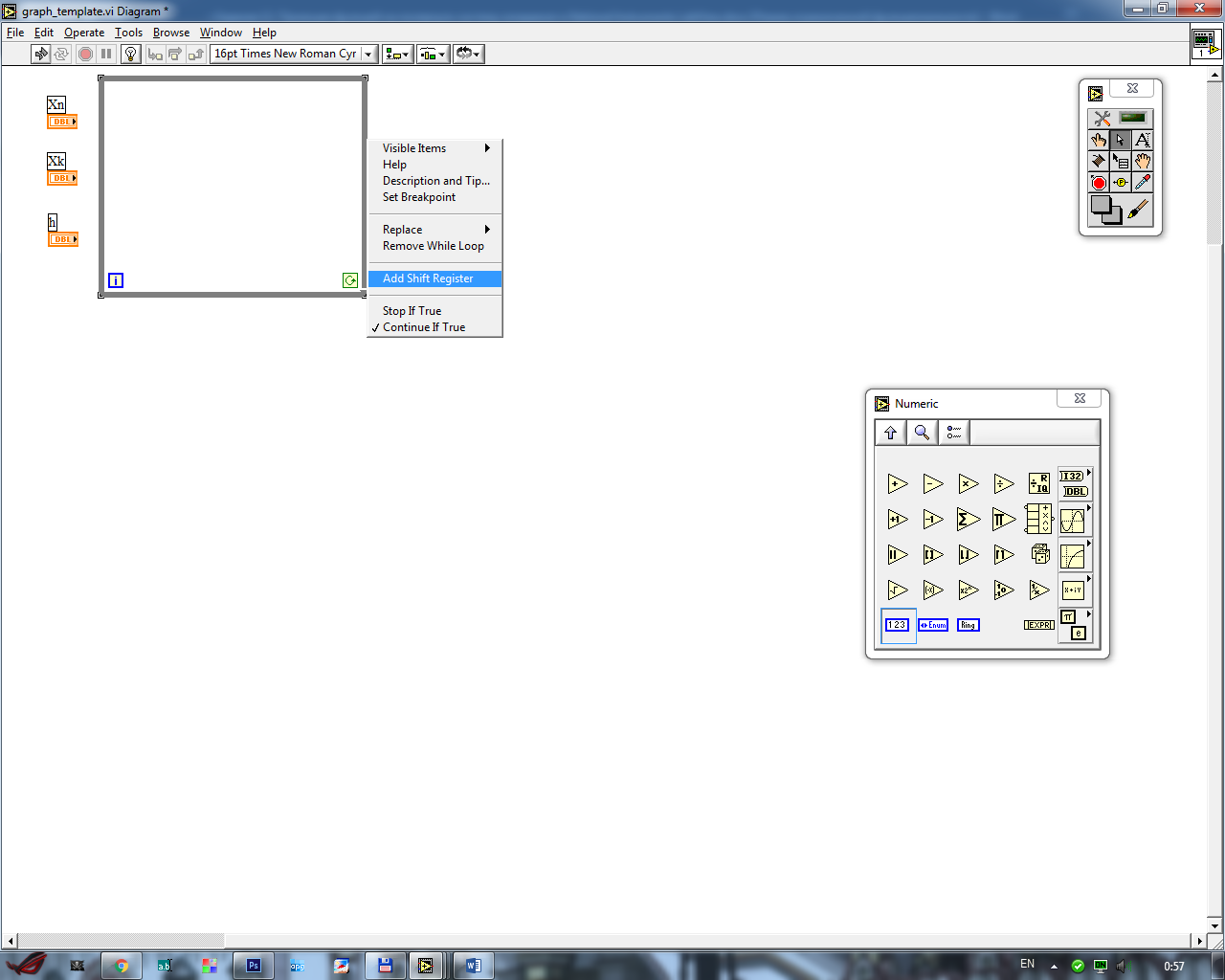


Рисунок 38 – Переход к добавлению сдвигового регистра к структуре цикла *While*

По умолчанию сдвиговый регистр является не типизированным элементом. Входящий и исходящий элементы сдвигового регистра формируются одновременно и перемещаются по границе циклической конструкции строго вслед друг за другом (Рисунок 39). В одной циклической конструкции может быть несколько сдвиговых регистров.

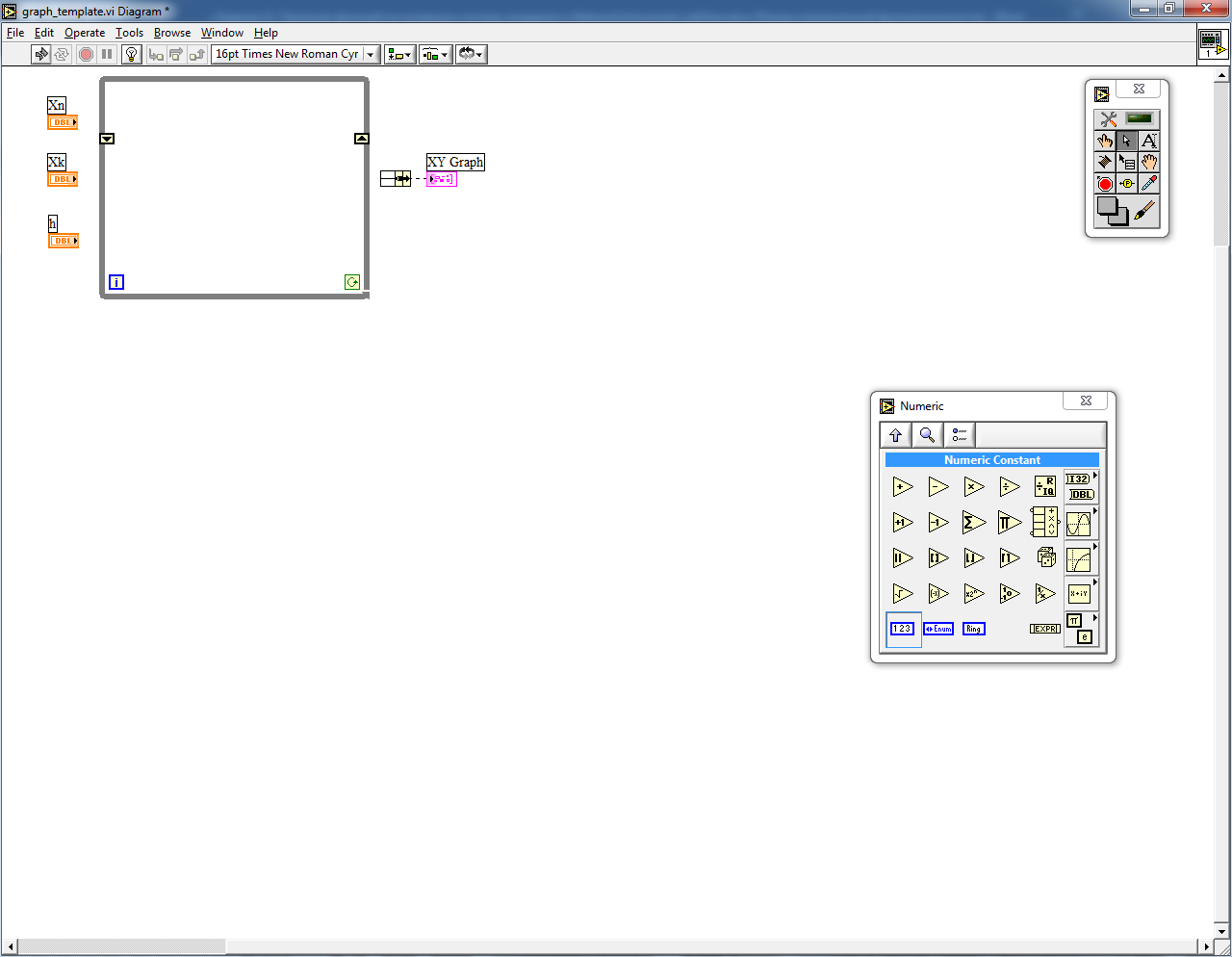


Рисунок 39 – Размещение сдвигового регистра в структуре цикла *While* (слева входящий сигнал (переданный), справа – исходящий (передаваемый))

При связывании сдвиговых регистров с другими элементами они получают свой вполне определённый тип данных и окрашиваются в соответствующий цвет. В примере на Рисунке 40 сдвиговый регистр окрашен в оранжевый цвет – он связан с вещественными значениями. Данный сдвиговый регистр в контексте рассмотрения становится ответственным за передачу меняющегося значения абсциссы от итерации к итерации.

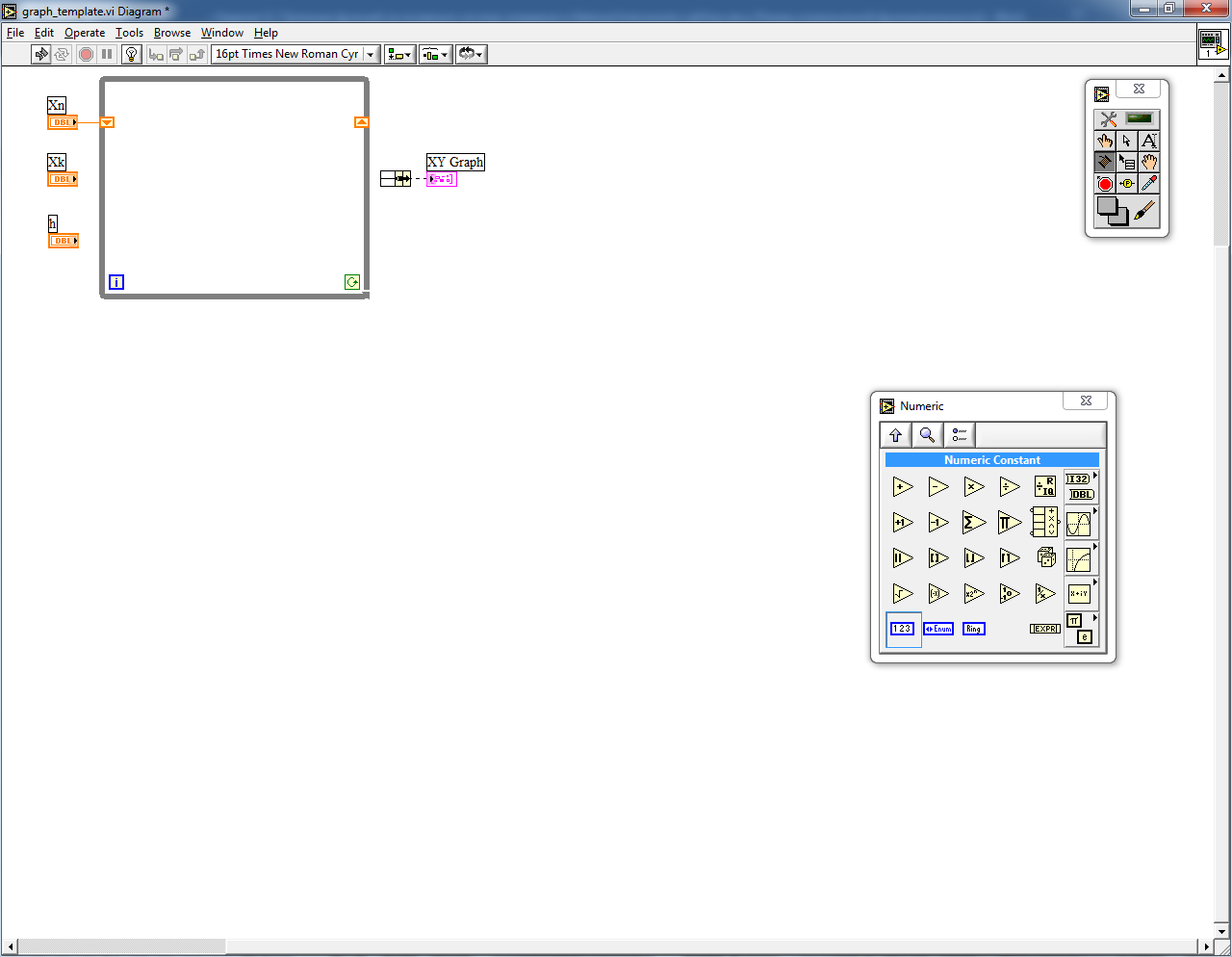


Рисунок 40 – На входящий элемент сдвигового регистра подаётся значение левой границы диапазона построения

К абсциссе, с которой производится работа на данной итерации, прибавляется шаг и затем полученное значение переносится на следующую итерацию (Рисунок 41).

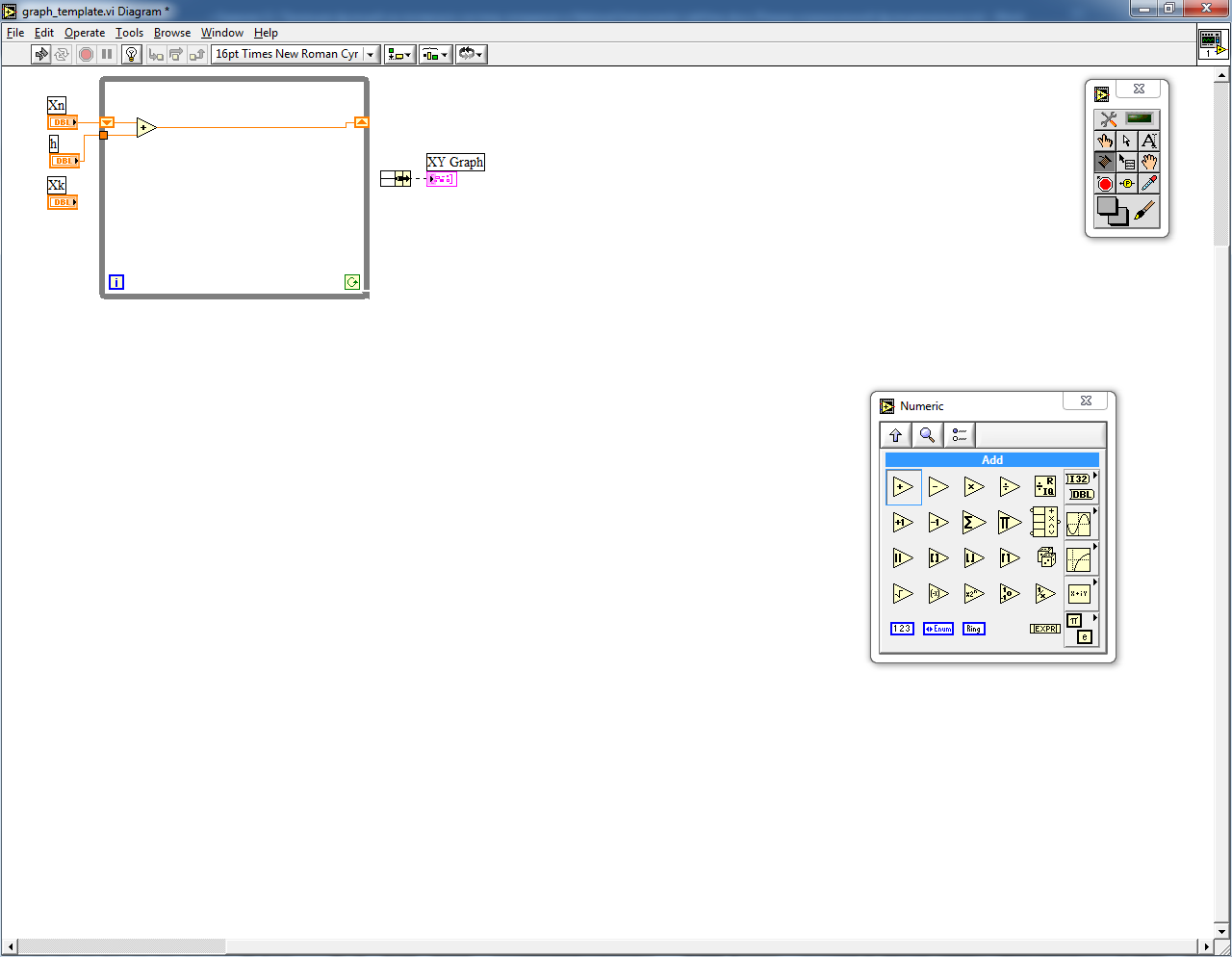


Рисунок 41 – На исходящий элемент передаётся текущее (*i*-е) значение абсциссы, смещённое на шаг построения

Вместе с тем текущее значение, содержащееся на итерации, необходимо фиксировать, потому оно в том виде, с которым с ним производится работа, передаётся на выход цикла (прикрепляется к его правой границе). Однако, в структуру цикла *While* по умолчанию заложена выдача только итогового, конечного значения, потому выход представлен сплошным квадратом, который обозначает режим «Запрет индексации» (*Disable Indexing*). Ситуация показана на Рисунке 42.

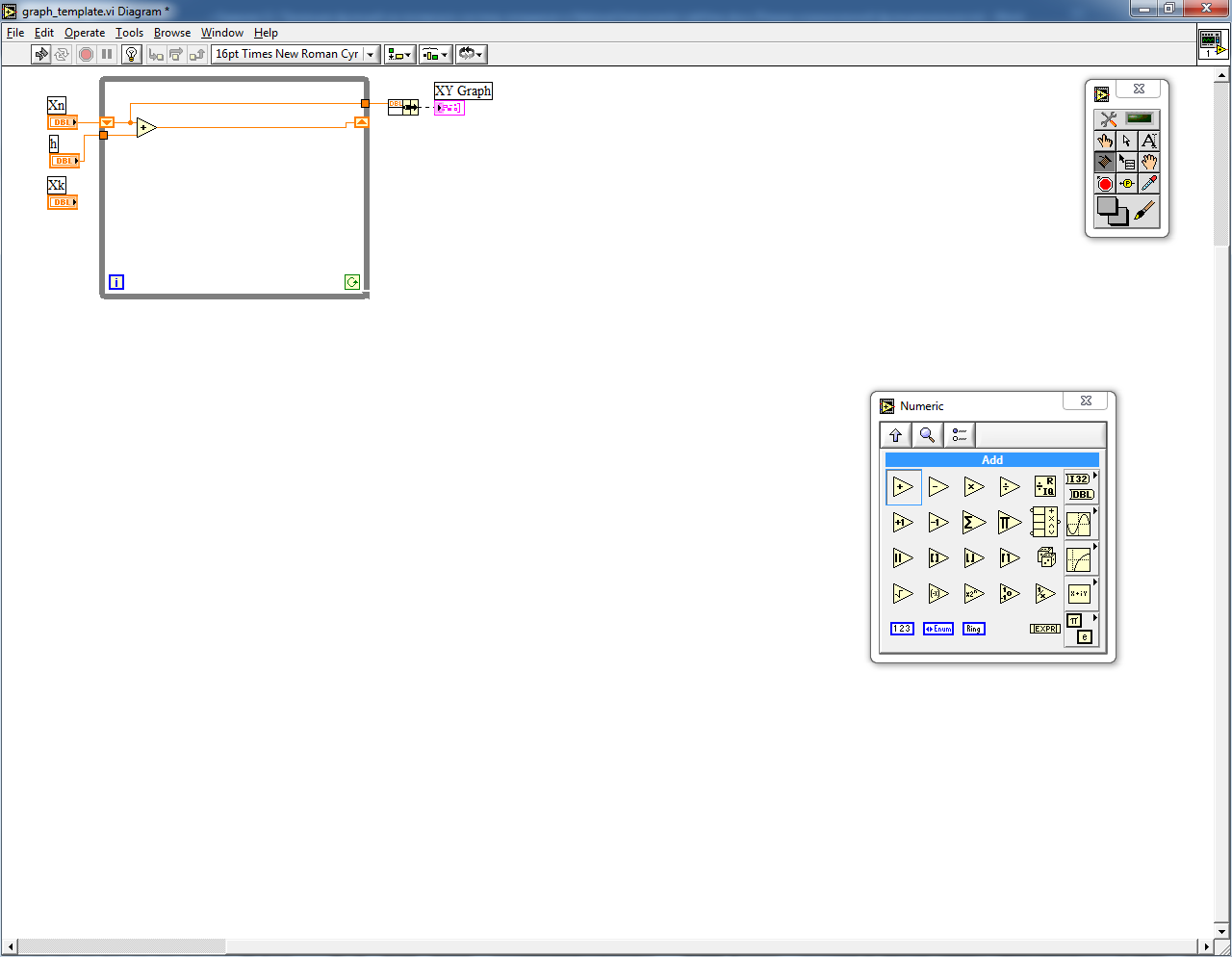


Рисунок 42 – Текущее значение передаётся на выход цикла в качестве абсциссы для сцепления в точку. По умолчанию цикл *While* передаёт на выход последнее полученное (неиндексированное) значение – прямоугольник сплошной

Режим без индексации неприемлем для решаемой задачи, поскольку запись значений должна вестись на каждой итерации цикла. В контекстном меню элемента, выводящего значение из цикла, необходимо выбрать пункт «Разрешить индексацию» (*Enable Indexing*).



Рисунок 43 – Переход к изменению абсциссы на выходе цикла с неиндексированного на индексированное (учитываемое на каждой итерации) значение – прямоугольник с квадратными скобками внутри

Выбранное изменение повлечёт за собой графическое преобразование. Выход значения из цикла станет «выколотым» квадратом с квадратными скобками внутри, что означает поэлементное формирование массива, а продолжение тонкой линии связи вне цикла преобразится в линию средней толщины, что возвестит о передаче далее одномерного массива (Рисунок 44). Для справки: среди обозначений линий существует ещё и толстая линия – многомерный массив.

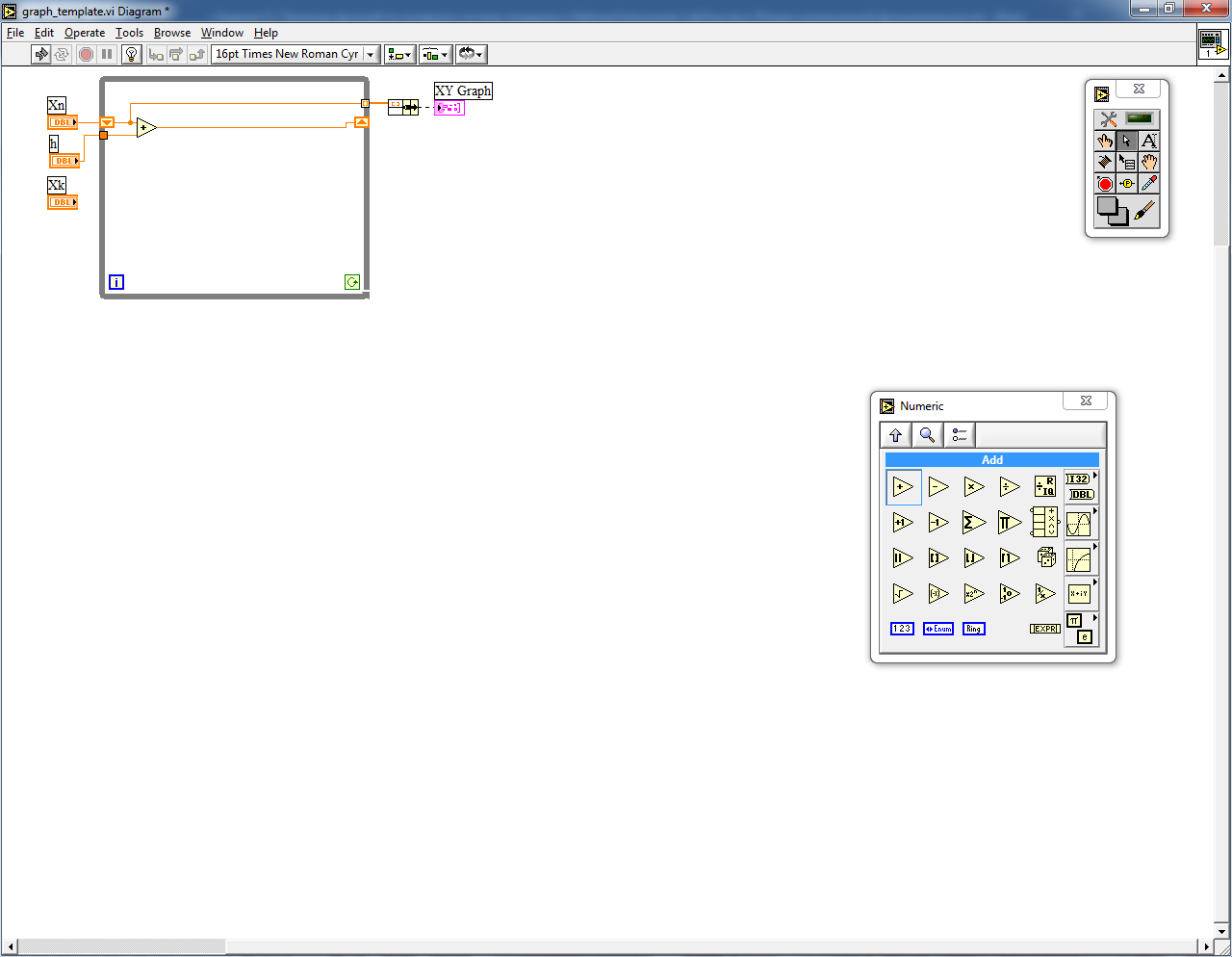


Рисунок 44 – Результат перехода к индексированным значениям абсцисс

На Рисунке 45 показано, как по аналогии выведены из цикла значения ординат, порученных в результате расчёта значения функции отсчётов для рассматриваемой на *i*-й итерации абсциссы.

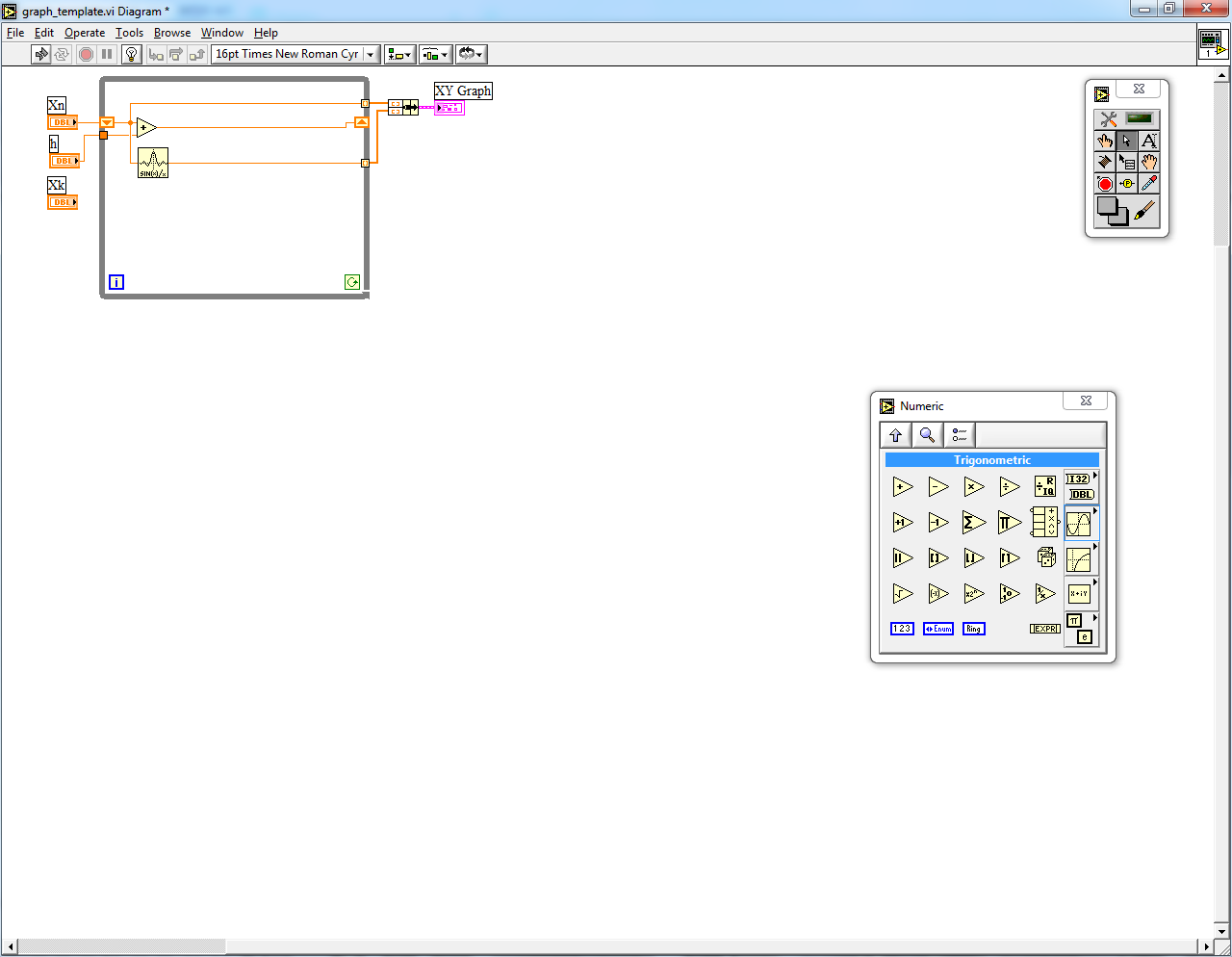


Рисунок 45 – Подключение функции отсчётов и вывод её значений в качестве индексированных ординат точек

Данные укомплектованы, но цикл в данном контексте написания является бесконечным. Для указания пределов работы цикла на его управляющий элемент должно быть подано логическое значение, получаемое, в частности, в результате выполнения или не выполнения логического выражения. Так значение абсциссы, которое планируется передать на следующую итерацию, должно быть сопоставлено с правой границей диапазона и пока оно его не превысило, продолжать выполнение расчёта координат для построения графика функциональной зависимости (Рисунок 46).

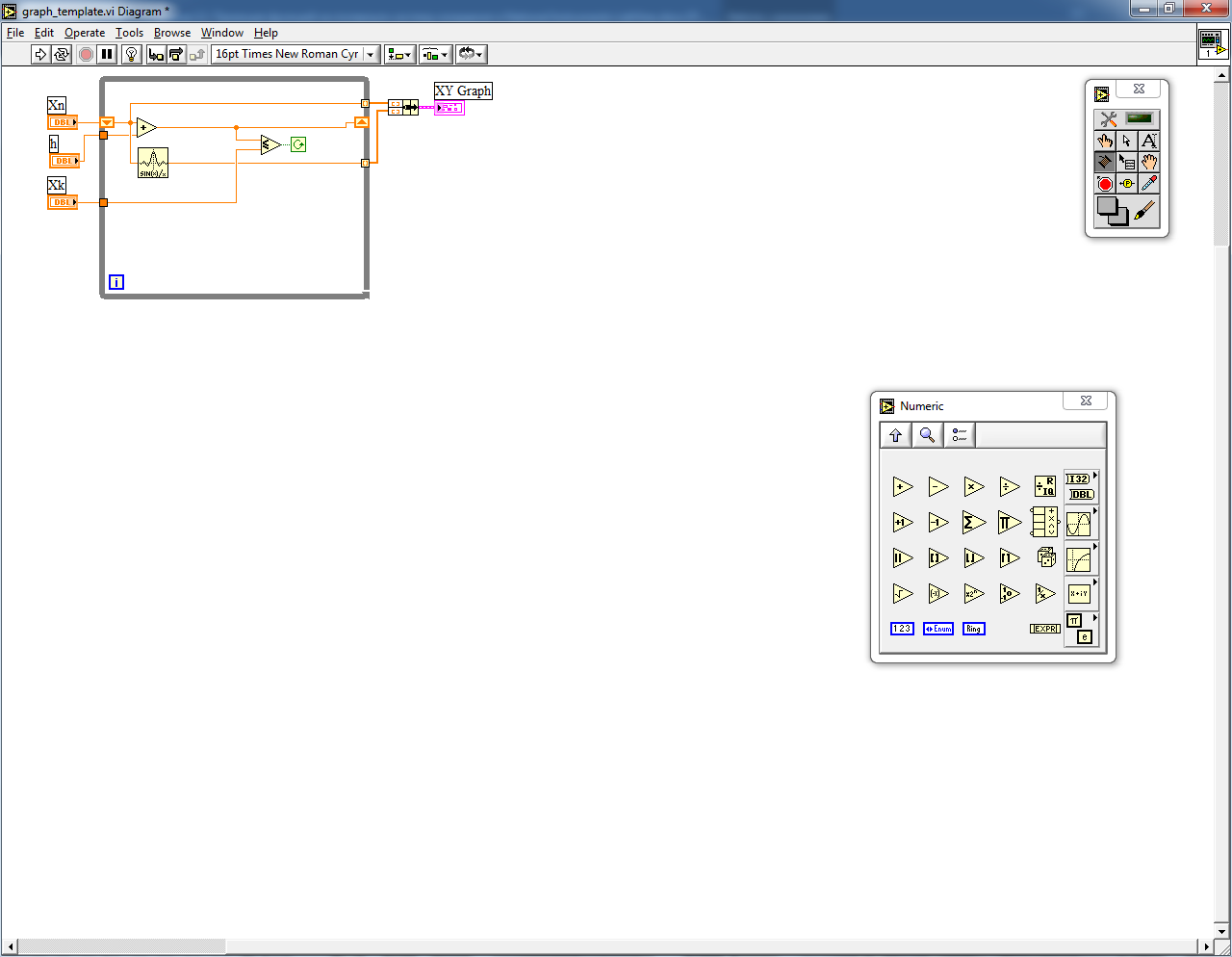


Рисунок 46 – Настройка управления функционирования цикла (продолжается пока следующее значение абсциссы не превышает значения правой границы диапазона построения)

После того, как код на блок-диаграмме начинает считаться отлаженным и работоспособным, его необходимо минимизировать – сократить занимаемое им пространство и по возможности максимально укоротить линии связи. Стоит отметить, что укорачивание линий связи – вещь субъективная и подойти к вопросу необходимо разумно, сделав так, чтобы код оставался читаемым (Рисунок 47).

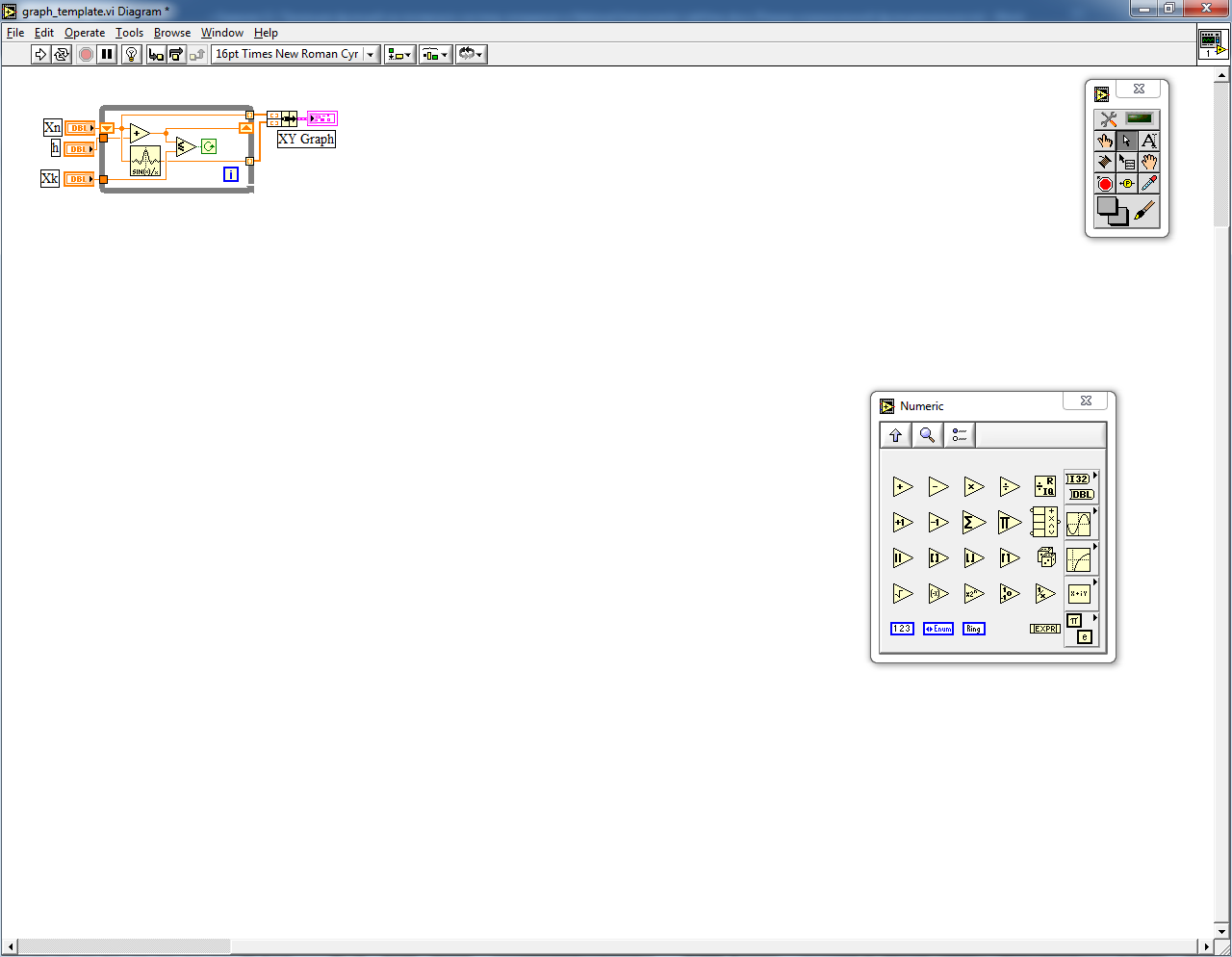


Рисунок 47 – Минимизация составленной кодовой структуры на блок диаграмме

По итогам однократного запуска на исполнение составленной программы получается график, соответствующий привычному представлению функции отсчётов. Иначе в математике функцию отсчётов называют первым замечательным пределом. При том, что у функции, казалось бы, должен получаться разрыв в точке с нулевой абсциссой, на самом деле в ней имеется неопределённость вида «0 / 0», которая при раскрытии даёт предел, равный единице. Таким образом, никакого разрыва у рассматриваемой функции нет.

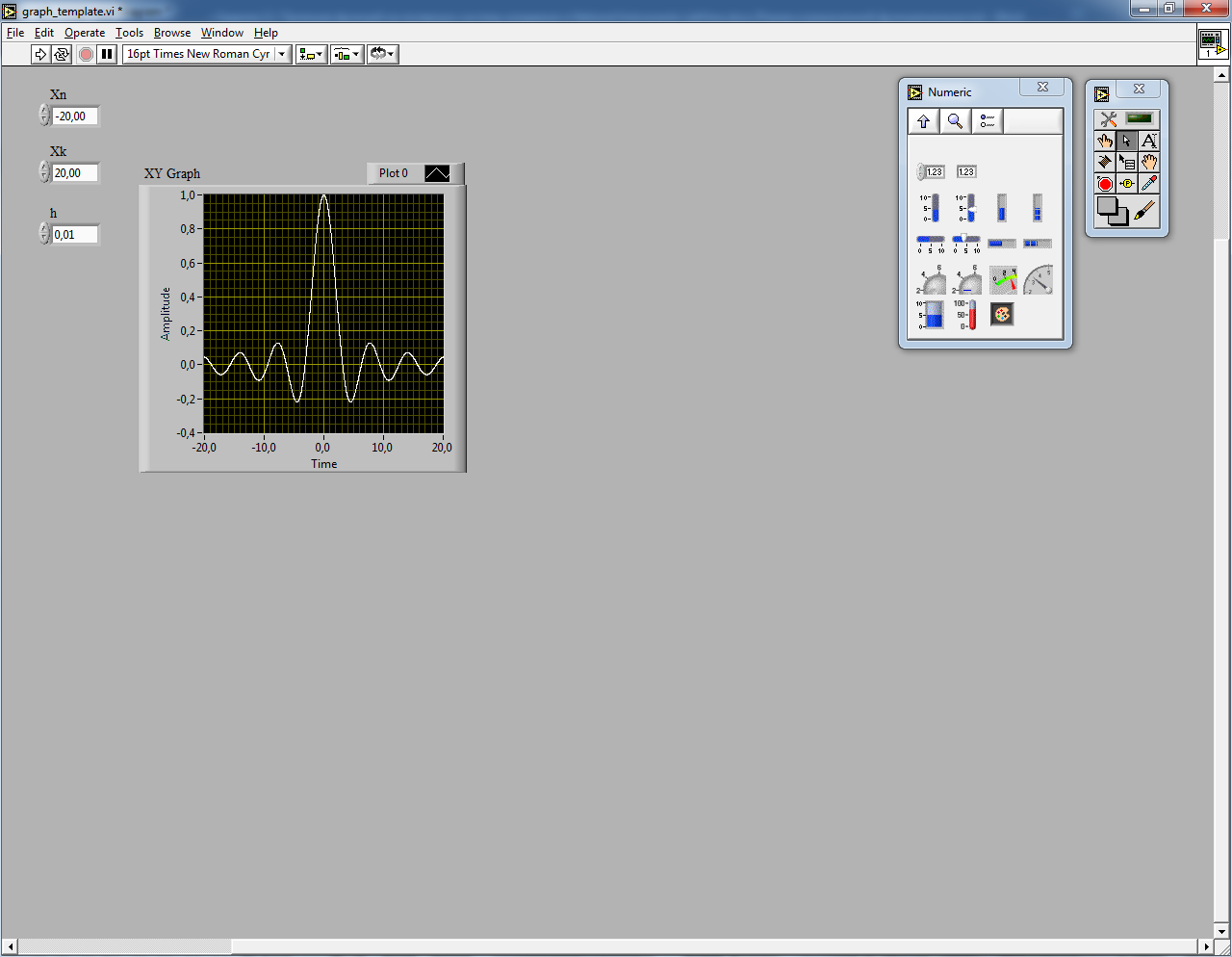


Рисунок 48 – Результат построения графика функции отсчётов на базе циклической структуры *While*

----------------------------------------------------------------------------------------------------

**Построение графиков функциональных зависимостей в *National Instruments LabView* на базе структуры цикла *For***

Данный раздел посвящён основам работы с циклической структурой *For*. На русский язык *For* переводится «для (конкретного / каждого / всех)». Данный перевод в некоторых контекстах рассмотрения может быть ошибочно воспринят обучающимися, потому для краткости и простоты повествования наименование конструкции далее и впредь будет оставлено без перевода. Циклическая конструкция *For*, как правило, называется в литературе итерационным циклом. Автор предпочитает использовать менее формализованное определение, потому применительно к *For* будет употребляться термин «цикл по известному диапазону значений». В *National Instruments LabView* циклическая конструкция *For* весьма наглядна и легка для восприятия. Она представлена в виде стопки листов бумаги с проставленными на них номерами страниц (*i*). Перелистывая страницу за страницей, пользователь, образно говоря, видит постоянно возрастающее значение *i*. Когда известное количество страниц заканчивается – выполнять становится нечего, потому и цикл останавливается.

Вернёмся к исходному состоянию (шаблону-заготовке) виртуального прибора для построения графиков функциональных зависимостей (Рисунки 7, 35, 49).

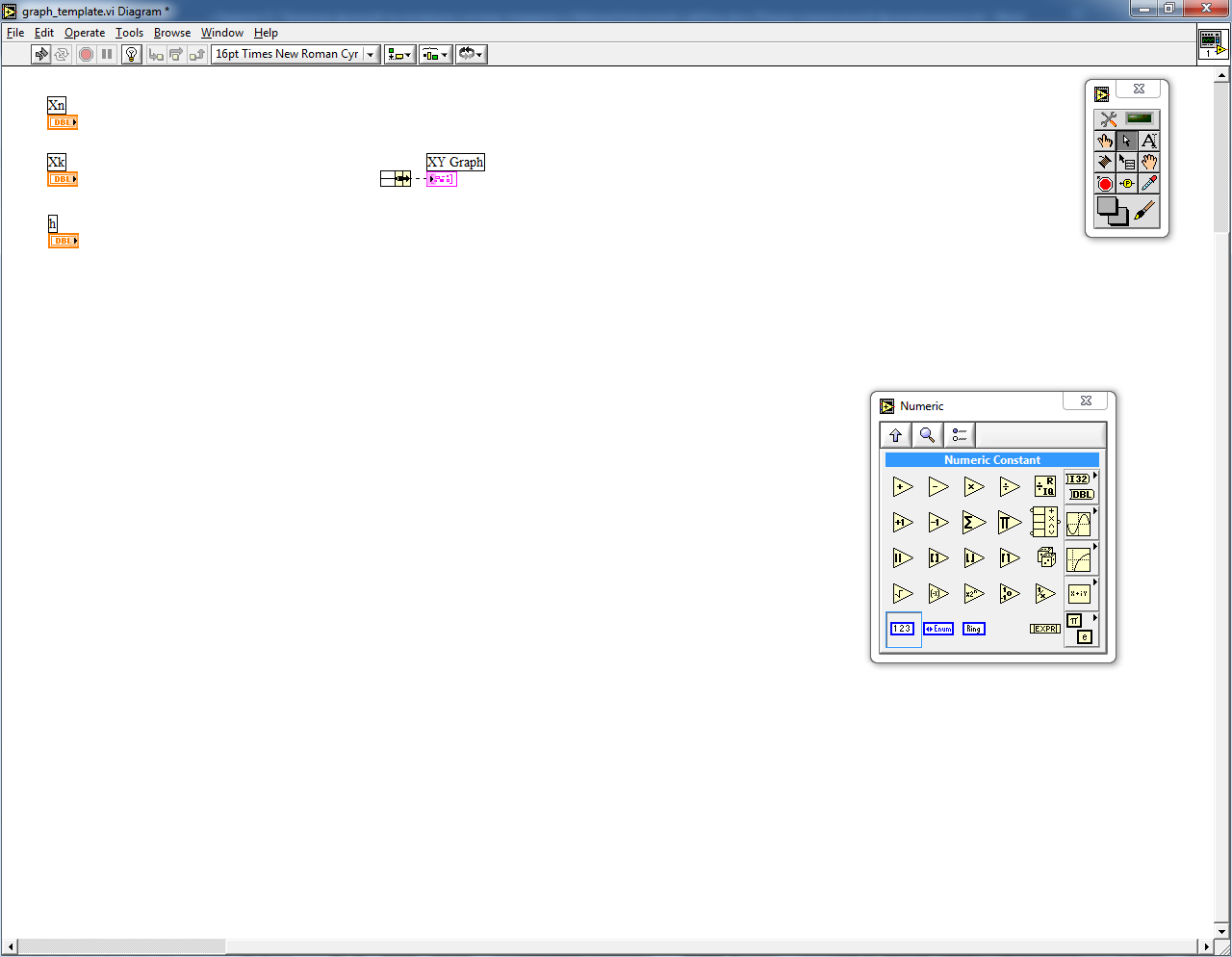


Рисунок 49 – Исходное состояние виртуального прибора для построения графиков функциональных зависимостей

Идея расчёта (алгоритм) при использовании цикла *For* состоит в следующем:

1. Вычисляется количество точек, необходимых для построения. Суть формулы, используемой для вычисления количество точек: количество шагов, которое необходимо проделать для покрытия диапазона. Из правой границы диапазона вычитается левая граница диапазона, полученный результат делится на шаг и инкрементируется. Полученное количество точек определяет количество итераций цикла *For*.

2. Отсчёт на каждой итерации цикла ведётся от левой границы диапазона. На каждой итерации цикла от левой границы диапазона делается отступ на *i* штук шагов.

3. Значение абсциссы пересчитывается согласно заранее определённым правилам (эти правила и есть функциональная зависимость), в результате чего получается значение ординаты.

4. Абсциссе и ординате ставится в соответствие порядковый номер или позиция в списке – полученные значения сохраняются в некоторой структуре данных или выводятся в таблицу.

5. Расчётные значения точек выводятся в область построения графика функциональной зависимости.

На Рисунке 50 отмечено расположение циклической структуры *For* в перечне функций, размещаемых на блок-диаграмме.

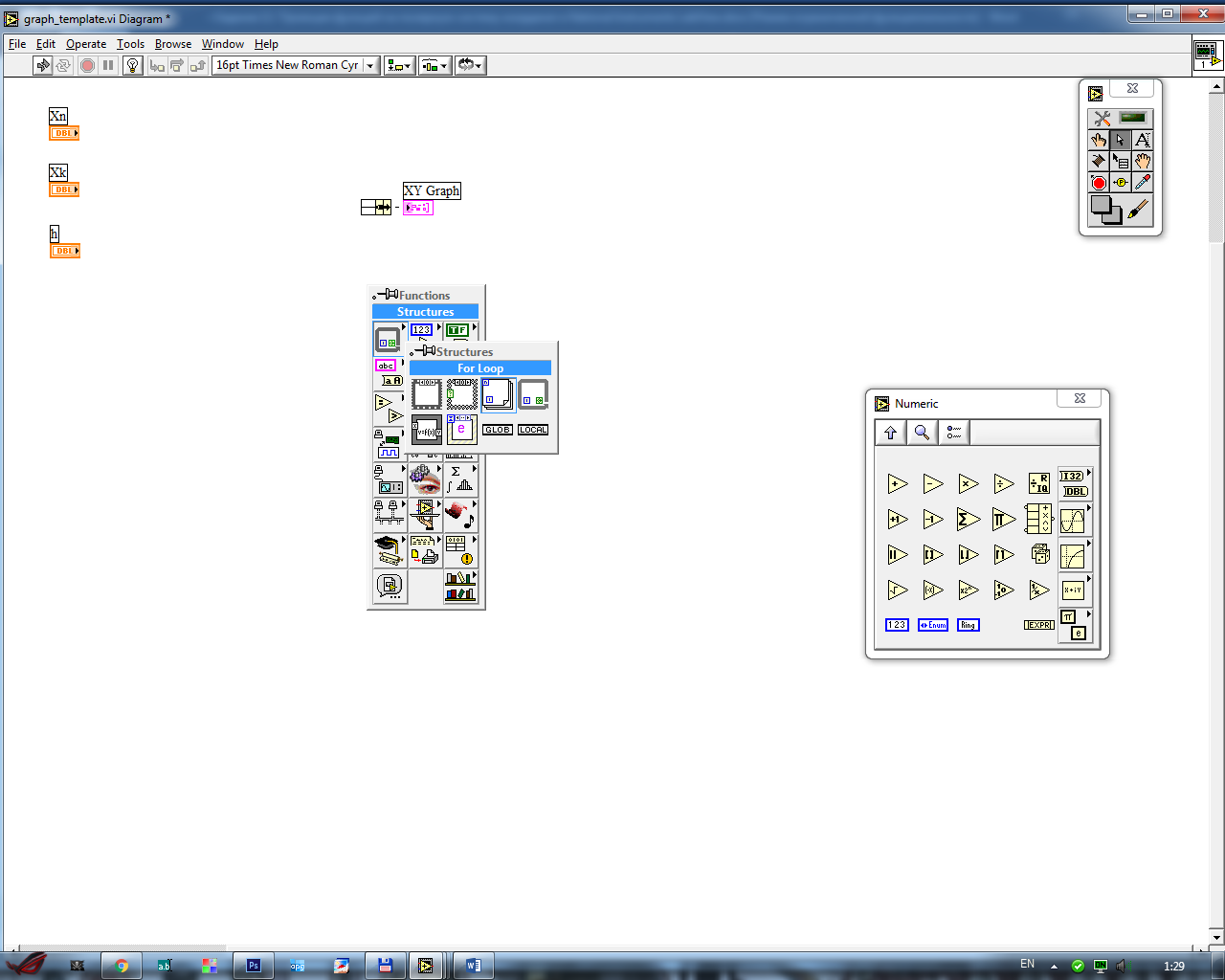


Рисунок 50 – Маркировка структуры цикла *For* (*For Loop*) в разделе структур из перечня функций, размещаемых на блок-диаграмме

Структура цикла *For* представляет собой прямоугольник, потому размещается на блок-диаграмме по принятым для большинства графических редакторов правилам (Рисунок 51). Левая кнопка мыши зажимается и удерживается в том месте блок-диаграммы, где планируется левый верхний угол структуры и отпускается в точке, где планируется правый нижний угол структуры. Возможно и инверсное размещение из правого нижнего угла в левый верхний.

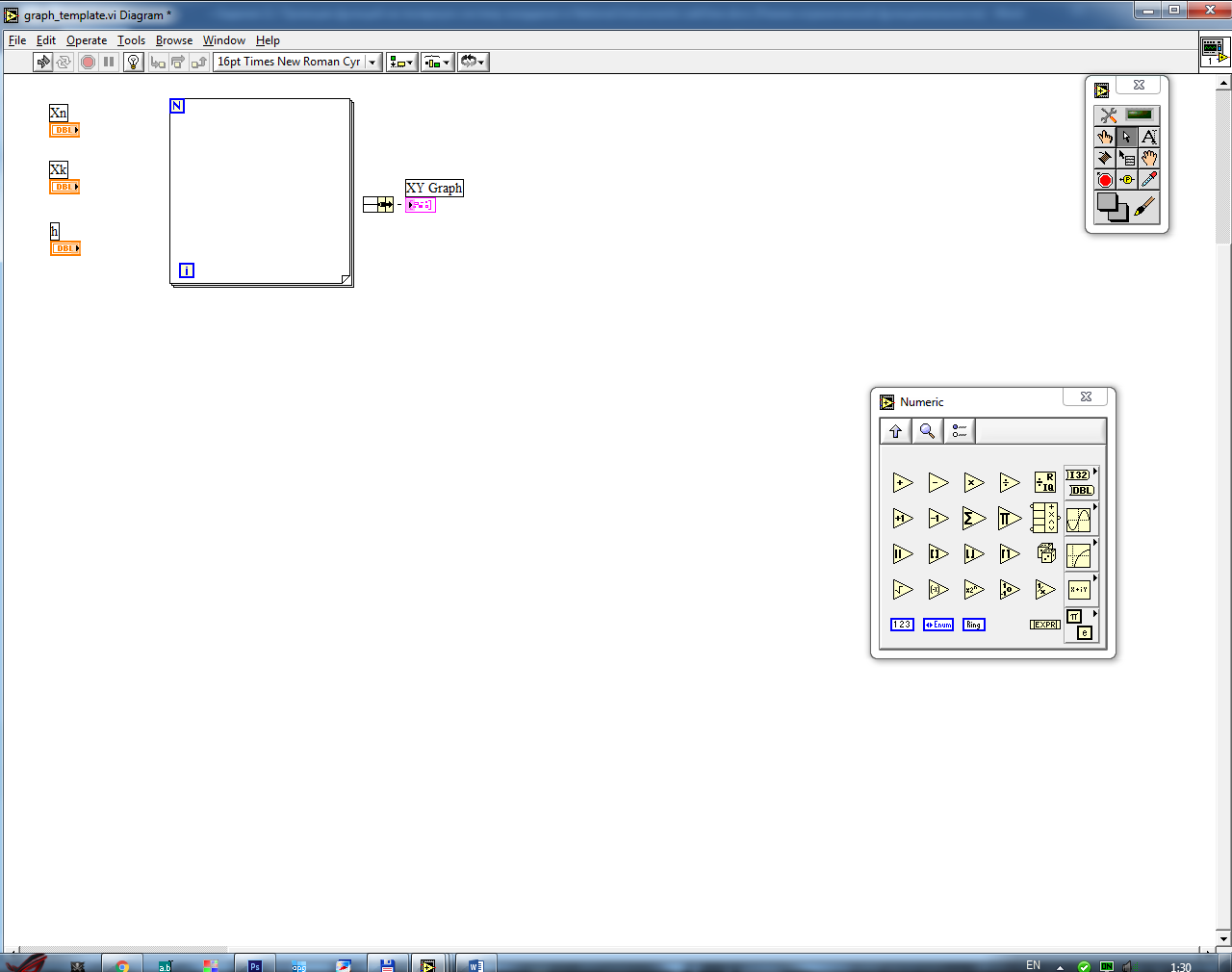


Рисунок 51 – Размещение цикла *For* на блок-диаграмме

Структуре должен предшествовать расчёт количества точек, необходимых для покрытия рассматриваемого диапазона значений по оси абсцисс. Расчётная формула имеет следующий вид: *N = ((Xk – Xn) / h) + 1*. Из конечного значения в диапазоне вычитается начальное значение в диапазоне, полученный результат делится на шаг и инкрементируется. Далее докажем необходимость инкрементирования. Рассмотрим диапазон от «–3» до «3», проходимый с единичным шагом. Запишем все точки, для которых необходимо рассчитать значение ординаты:

*-3, -2, -1, 0, 1, 2, 3*

Рассмотрению подлежат 7 точек. Проверим. Произведём расчёт без инкрементирования: *N = (3 – (–3)) / 1 = 3 + 3 = 6* точек. Из проведённого вычисления следует, что для покрытия требуемого количества значений не хватает одной единственной точки, потому инкрементирование результата выполнять обязательно.

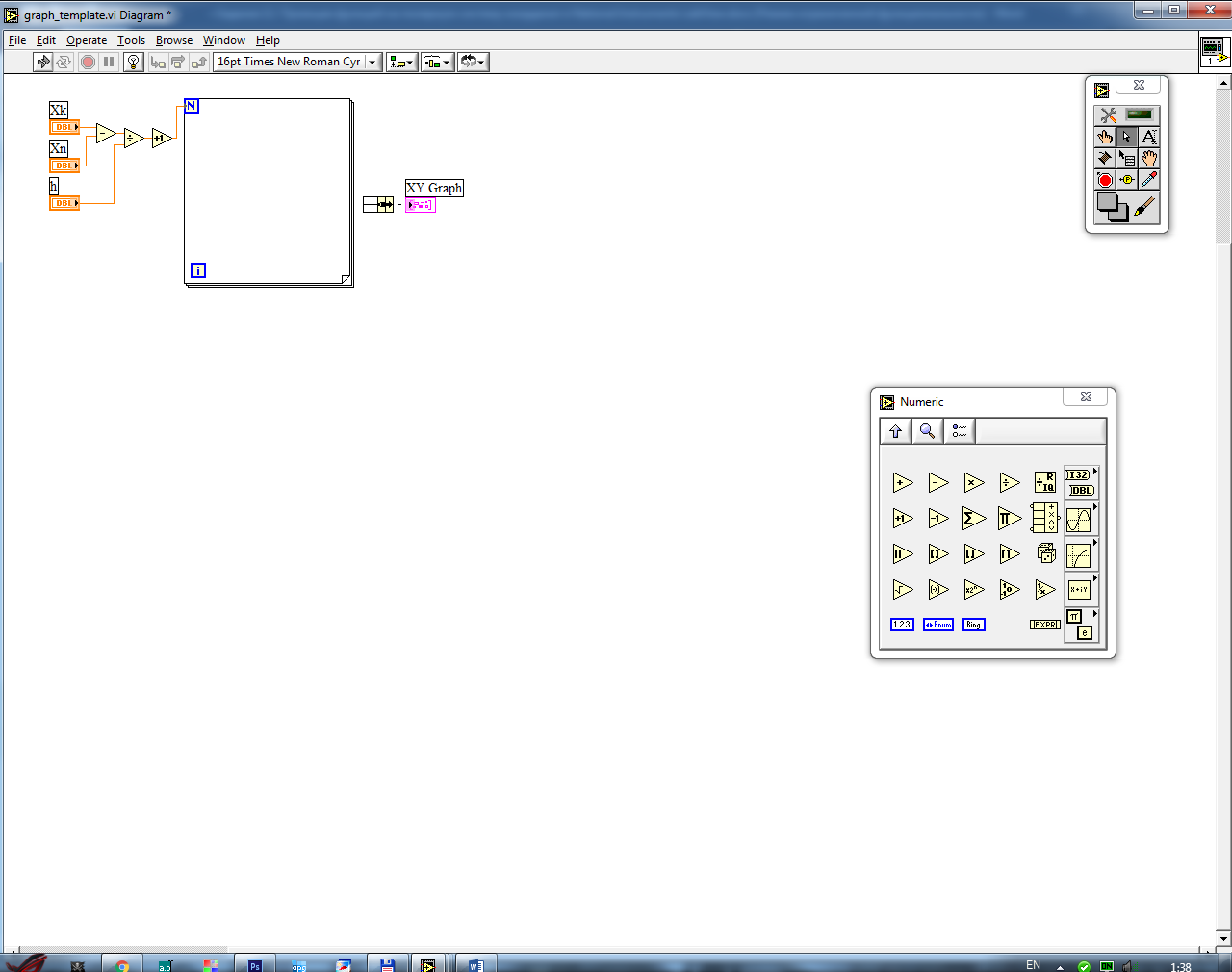


Рисунок 52 – Вычисление и подключение количества расчётных точек к структуре цикла *For*

Продвигаемся дальше по алгоритму. К начальному значению в диапазоне необходимо прибавить *i* штук шагов. Значение *i* уже содержится в циклической структуре, потому внесём в цикл недостающие элементы, протянув линии связи с циклом от начального значения в диапазоне, а также от шага построения. *i* штук шагов означает, что шаг построения должен быть умножен на значение *i*, а далее полученный результат должен быть добавлен к начальному значению в диапазоне. Таким образом, они должны быть объединены операцией сложения. На выходе операции сложения располагается текущее значение по абсциссе на *i*-м шаге (Рисунок 53). Важно помнить, что счётчик итераций *i* начинается с нулевой отметки, потому на первой итерации цикла имеем в точности абсциссу, совпадающую с началом диапазона построения:

*1. x0 = Xn + i\*h = Xn + 0\*h = Xn*

*2. x1 = Xn + i\*h = Xn + 1\*h = Xn + h*

*3. x2 = Xn + i\*h = Xn + 2\*h*

*… … …*

*N+1. xN = Xn + N\*h = Xk*

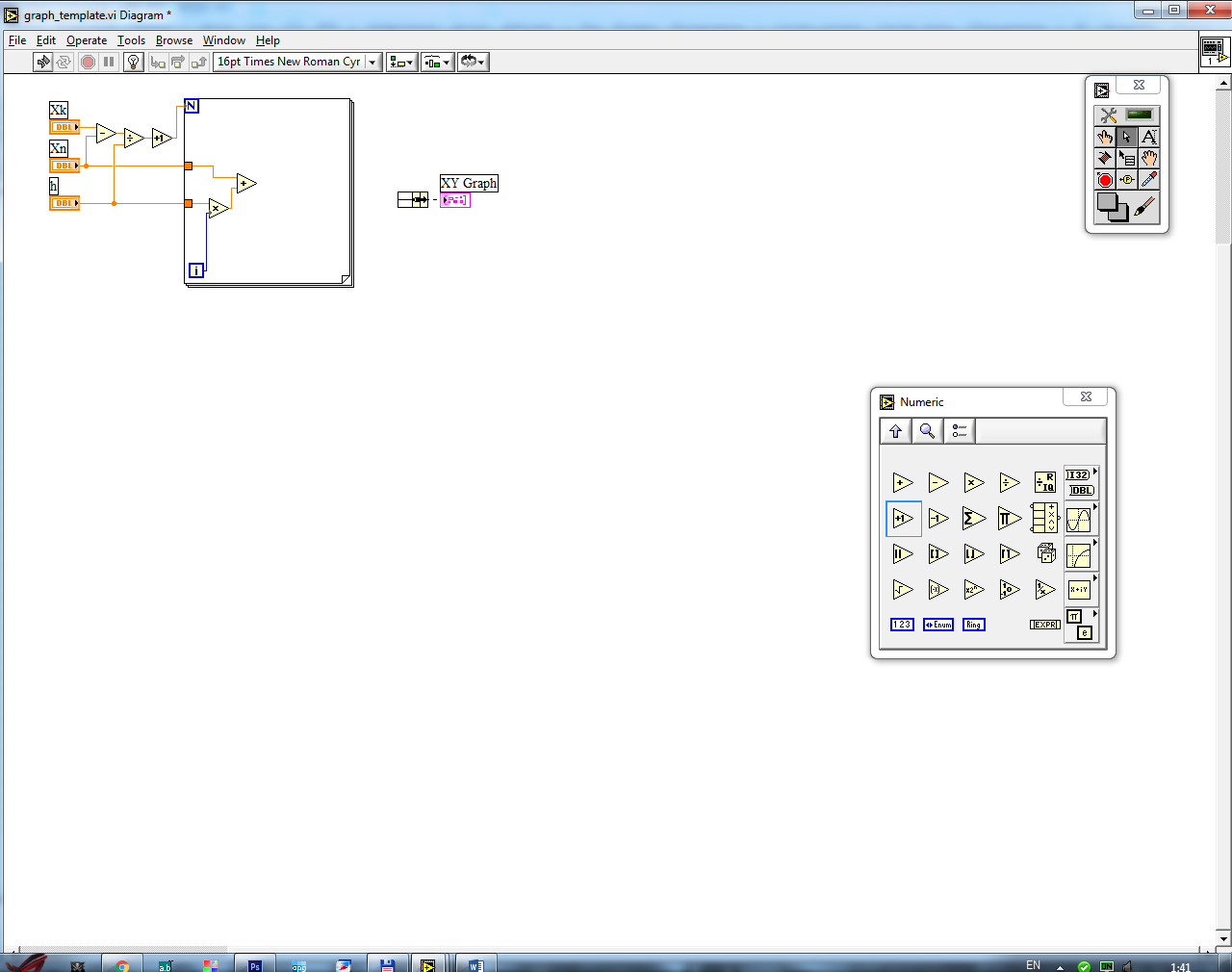


Рисунок 53 – Формирование текущей (*i*) абсциссы – абсцисса, смещённая относительно левой границы диапазона на *i* штук шагов

Полученное значение абсциссы можно передать на выход цикла. Оно по умолчанию проиндексировано – на выходе массив абсцисс, который можно направить на элемент «*Bundle»* для последующего сцепления в массив точек (Рисунок 54).

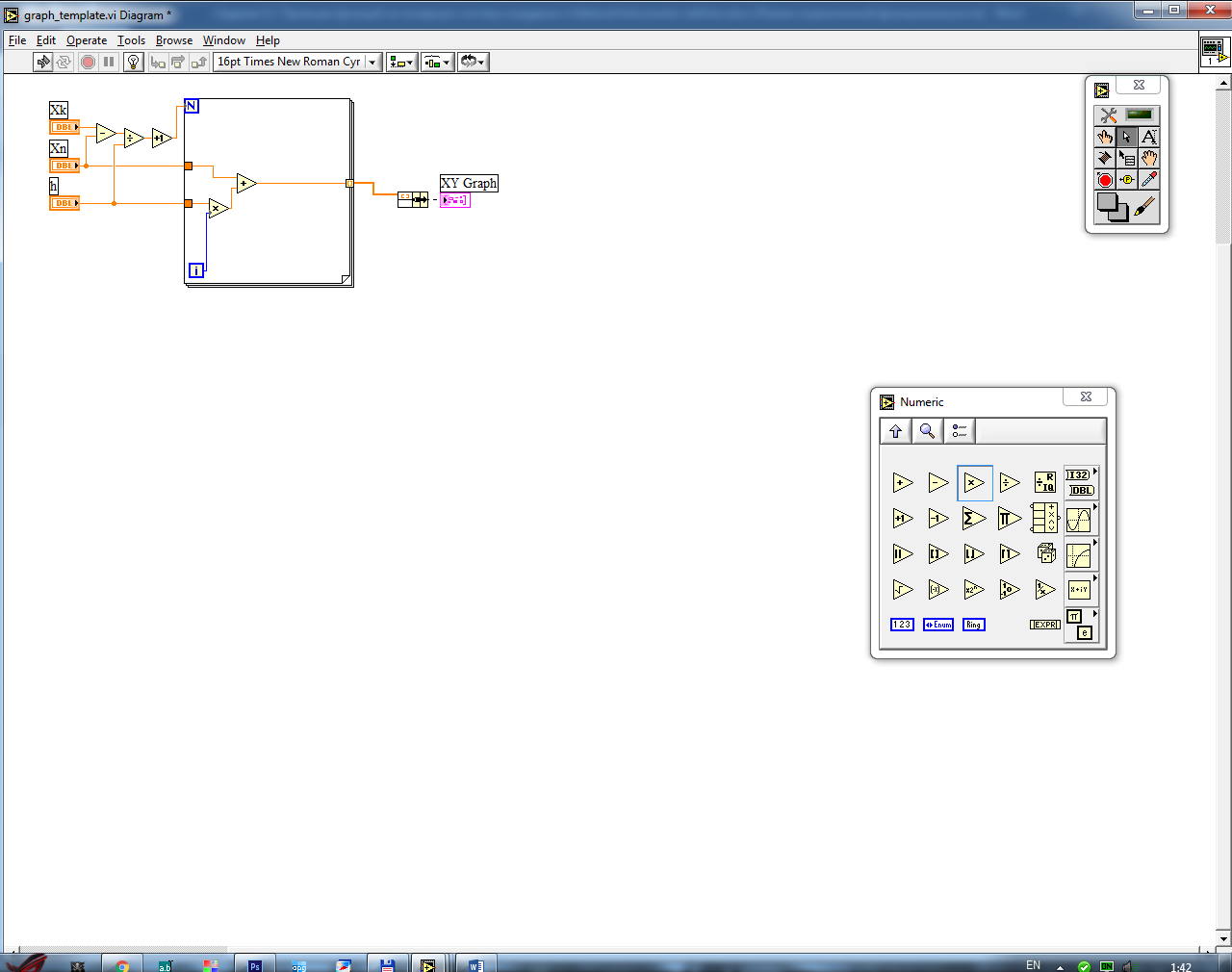


Рисунок 54 – Передача абсцисс на «сцепку» в точку

В качестве примера функции возьмём всё ту же функцию отсчётов. Пропустим через неё значение абсциссы, выведем из цикла, где получаем массив ординат. Массив ординат подключаем ко второму входу элемента «*Bundle*» и получаем на выходе массив точек, по которым «*XY Graph*» готов выполнить построение графика функциональной зависимости (Рисунок 55).

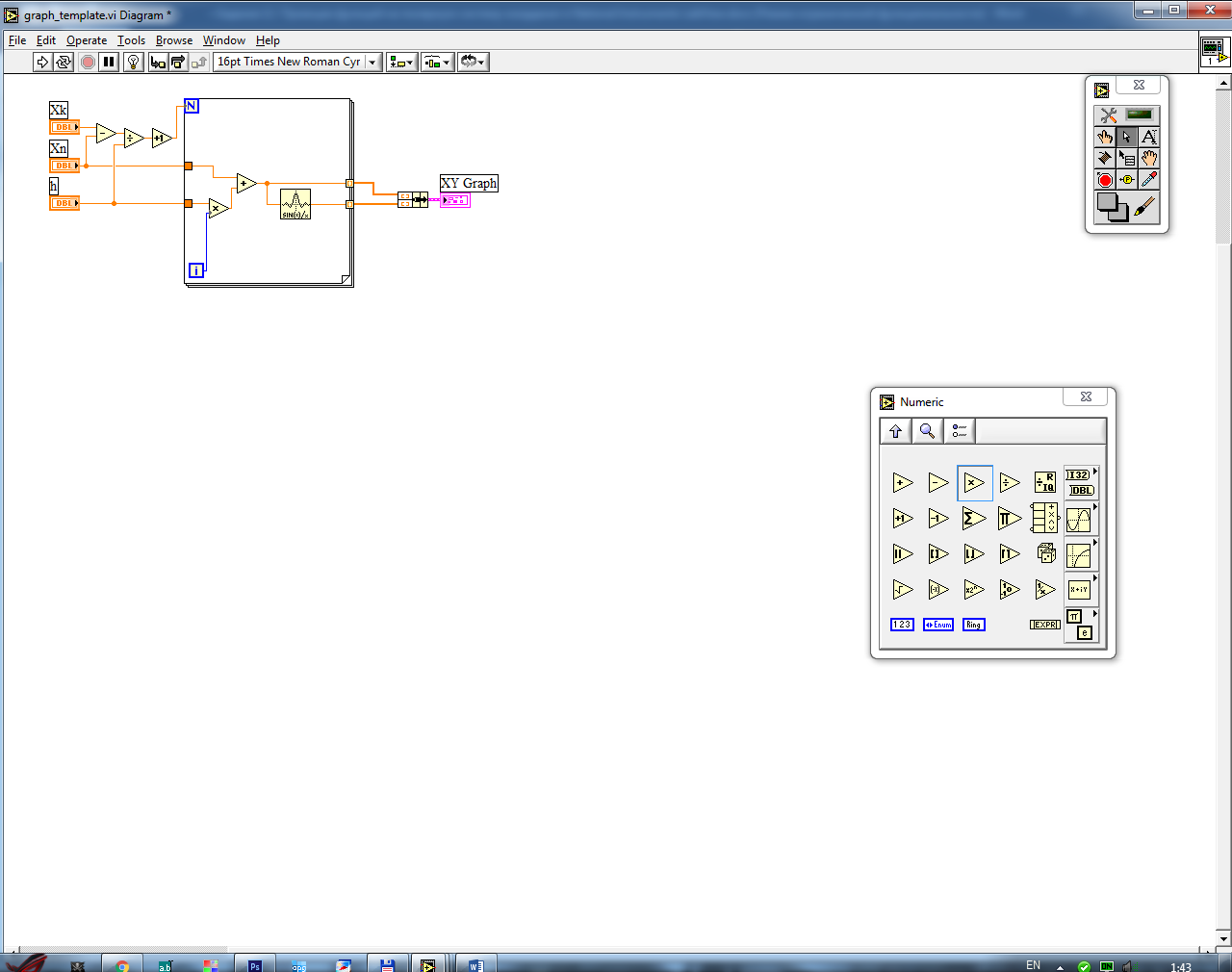


Рисунок 55 – Подключение функции отсчётов и выдача её в «сцепку» в качестве ординат точек

На этом процесс составления кода на блок-диаграмме завершён. Осталось только выполнить минимизацию кода по известным правилам (Рисунок 56).

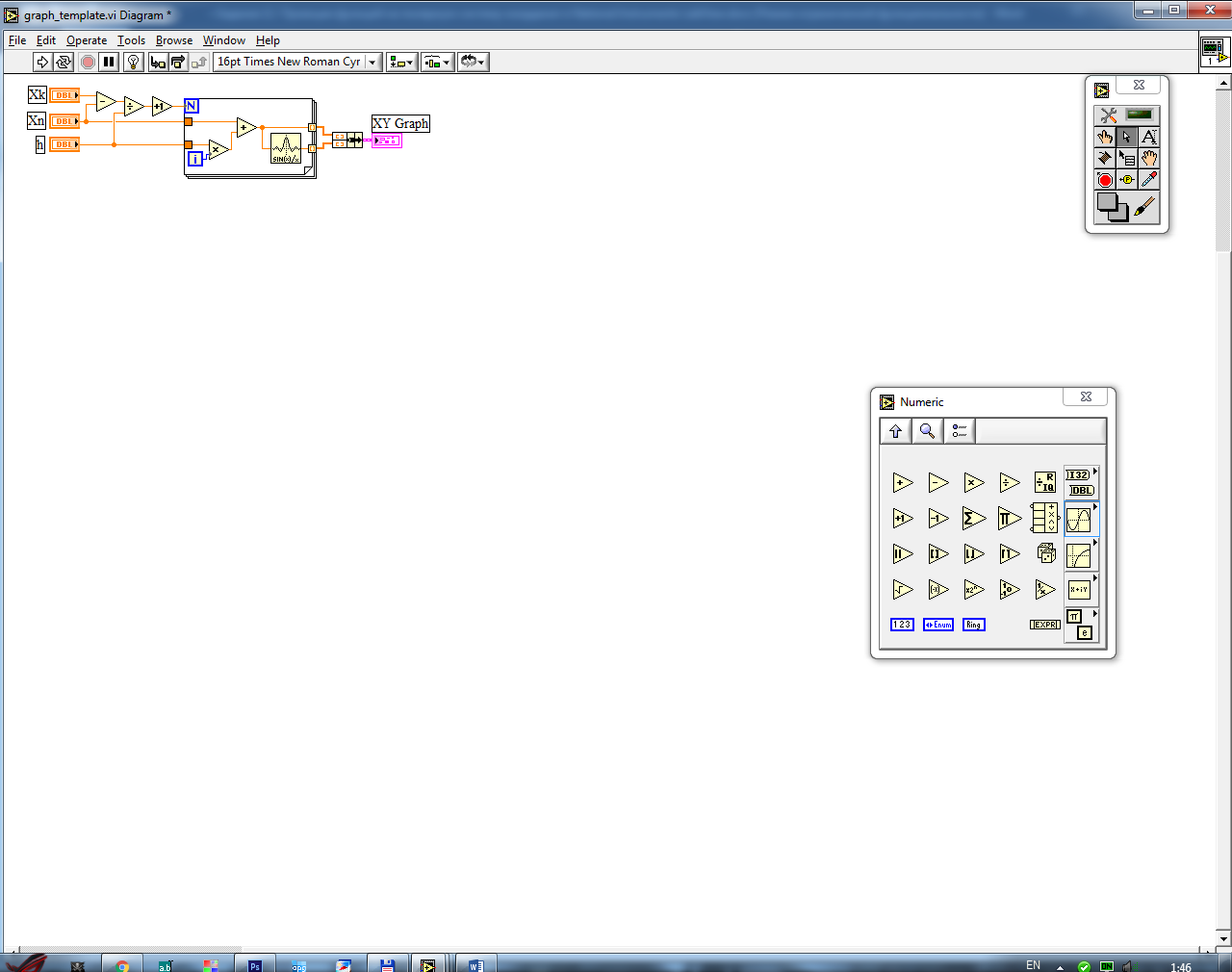


Рисунок 56 – Минимизация составленной кодовой структуры на блок диаграмме

Хорошо видно, что при использовании структуры цикла For получить результат построения графика функциональной зависимости проще и быстрее, нежели с использованием структуры цикла *While*. Это связано со спецификой потоковых вычислений *National Instruments LabView*. С точки зрения времени и объёмов вычислений структура, применённая к циклу *While*, экономичнее, поскольку там проводятся быстрые операции суммирования, а к *For* применена медленная и объёмная операция перемножения. Далее каждый сам должен принять для себя решение о том, какую структуру ему удобнее использовать и задумываться ли об экономии чего-либо, или же нет. Результат с *For*, как показано на Рисунке 57, идентичен результату, полученному на Рисунке 48 c *While*. Их можно и нужно между собой сравнить.

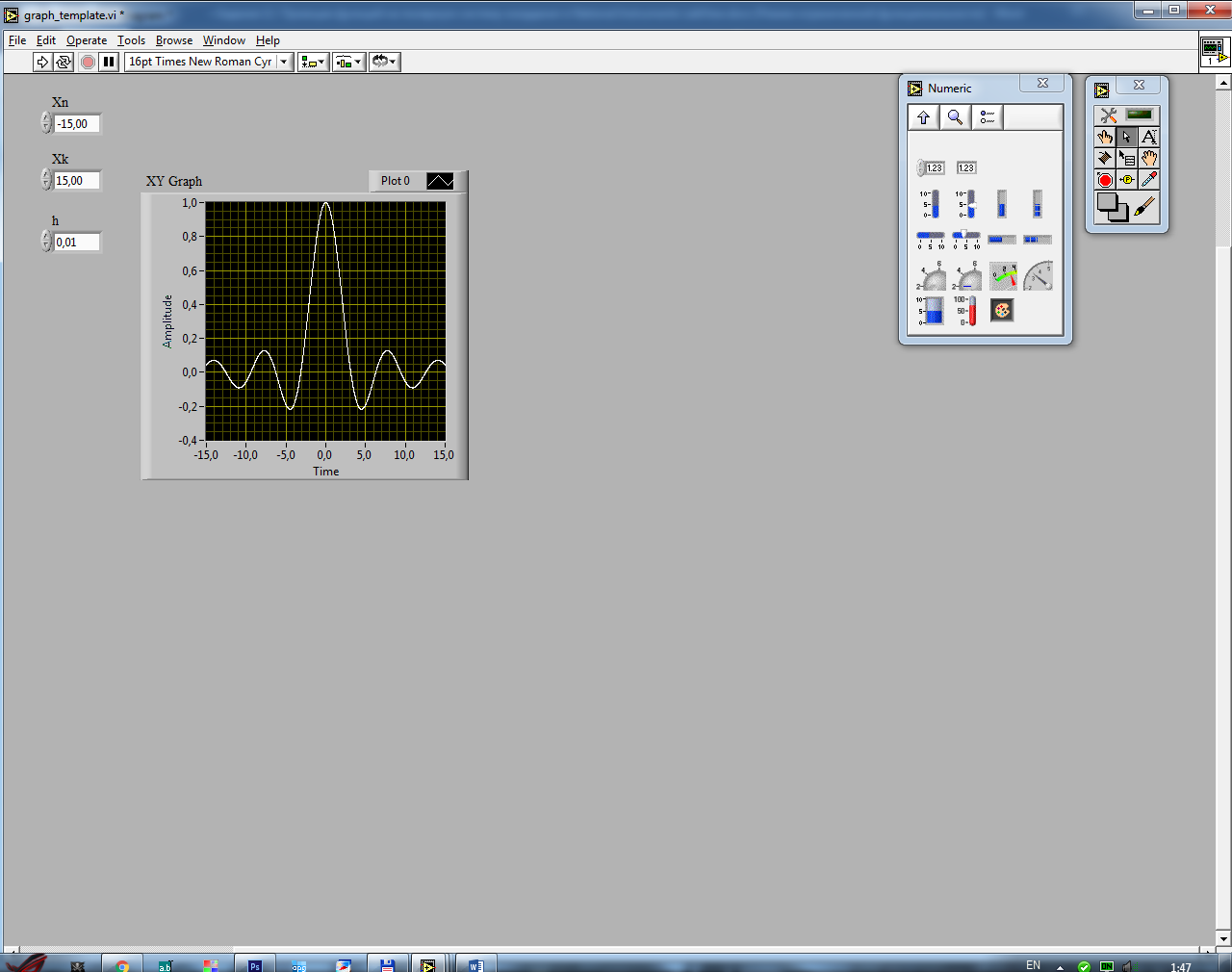


Рисунок 57 – Результат построения графика функции отсчётов на базе циклической структуры *For*

----------------------------------------------------------------------------------------------------

**Способы защиты виртуального прибора от запуска пользователем при нулевых значениях шага построения**

Наиболее часто встречающейся ошибкой обучающихся при решении задач, связанных с построением графиков функциональных зависимостей в *National Instruments LabView* является запуск виртуального прибора с нулевым значением шага построения, при котором поток отвечает исключением типа «переполнение памяти» (Рисунок 58). Возникновение ошибки в состоянии сбить с толку и даже напугать пользователя. Таким образом, данный раздел призван помочь обучающимся справиться с возникновением ошибок подобного характера.

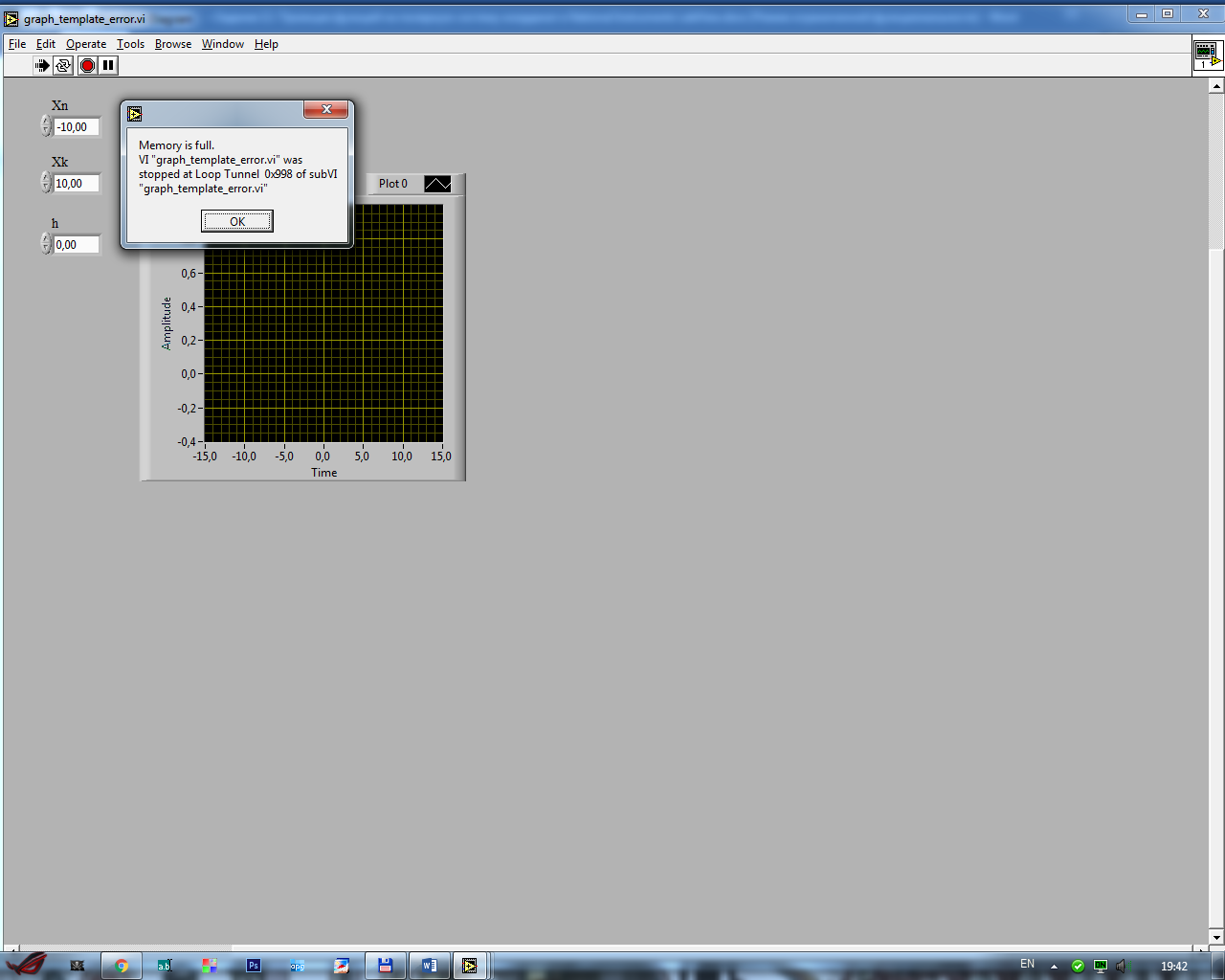


Рисунок 58 – Ошибка, возникающая при запуске виртуального прибора с нулевым шагом построения графика функциональной зависимости

Рассмотрим быстрый и наиболее простой способ справиться с исключением – остановить программу при подаче нулевого шага на вход виртуального прибора. В этом случае при подаче на вход нулевого шага программа будет прервана и ничего не сообщит пользователю. Он, соответственно, не будет получать в ответ никакого значения в качестве конечного результата. Автор виртуального прибора быстро сообразит, что дело в исходных данных, и либо запустит программу в пошаговом режиме, да заметит, к чему сводится решение (на каком шаге и в какой ветке всё останавливается, и почему это происходит), либо сходу поменяет значение шага, зная ограничения, заложенные в виртуальный прибор. Однако, рядовой пользователь не будет смотреть в код виртуального прибора, ему надо предоставить на передней панели чёткую и ясную последовательность его дальнейших действий, чтобы он мог исправить ошибку самостоятельно. Итак, для обеспечения простого и быстрого способа потребуется элемент, именуемый «остановом» (*Stop*). Располагается этот элемент в разделе управления приложением как показано на Рисунке 59.

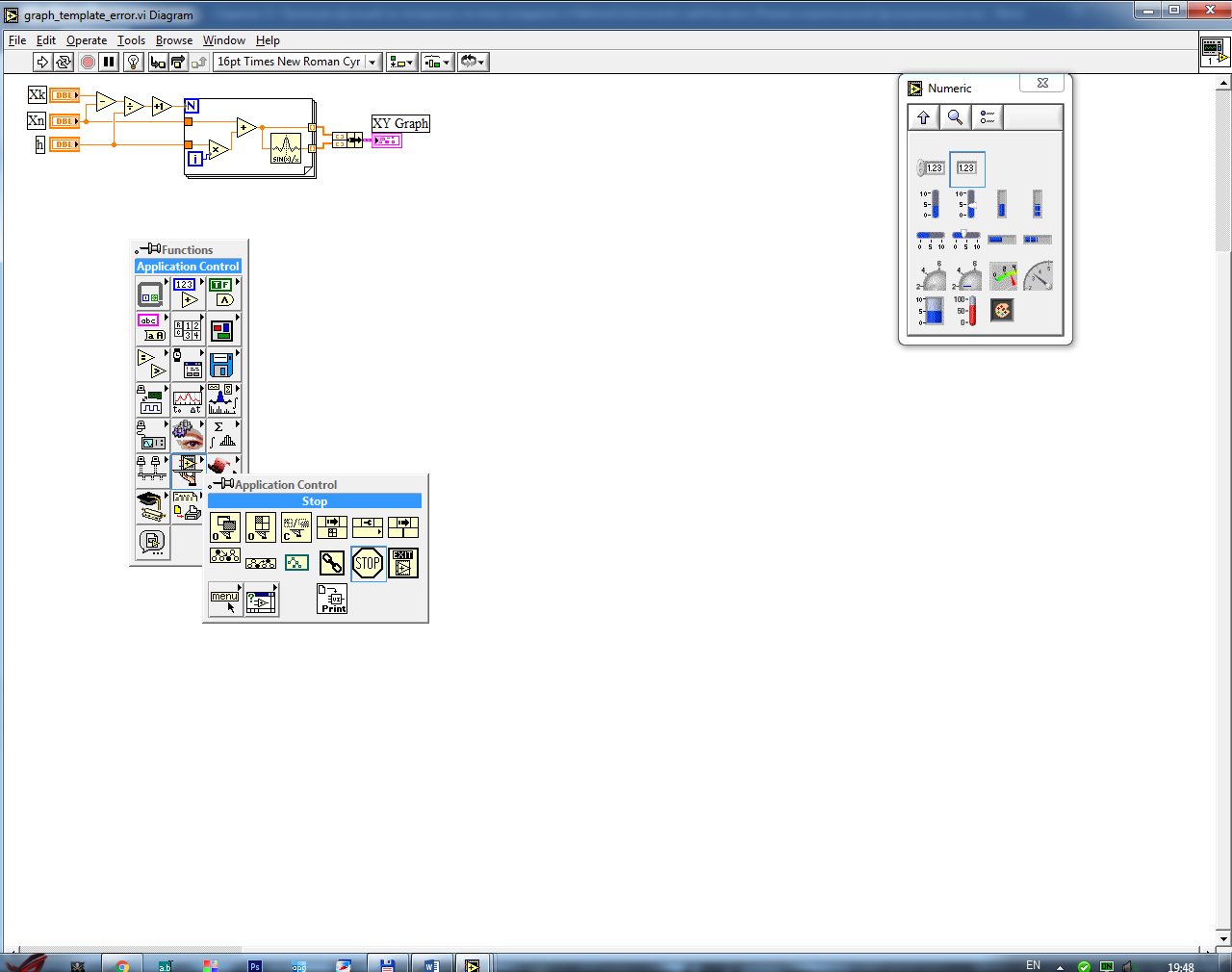


Рисунок 59 – Маркировка оператора прерывания работы виртуального прибора (*Stop*) в разделе управления приложением (*Application Control*) из перечня функций, размещаемых на блок-диаграмме

«Останов» подключается к логике сравнения шага в точности с нулём. Если шаг нулевой – истина, стало быть, нужно остановить работу программы, если шаг не нулевой – ложь, то программа может продолжить свою нормальную работу (Рисунок 60). Этот способ хорош только для автора-разработчика. Для выдачи продукта на суд и использование конечным пользователем он неприменим.

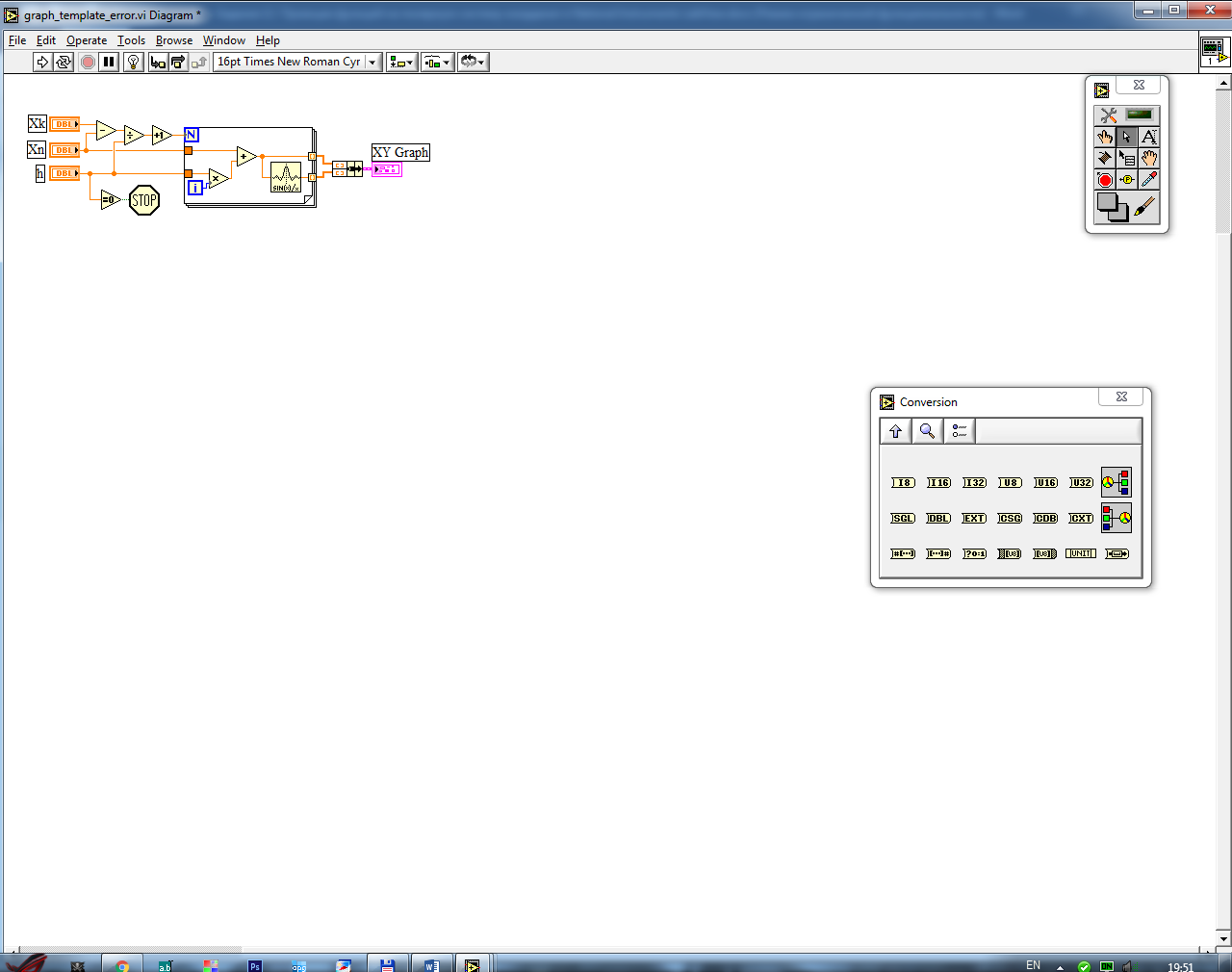


Рисунок 60 – Пример отлова ошибки нулевого шага без уведомления пользователя об «останове» программы

Чтобы сделать продукт удобным для пользователя, с ним необходимо организовать диалог. Потому потребуется воспользоваться элементом из раздела «Время и Диалог» (*Time & Dialog*), который называется однокнопочным диалогом (*One Button Dialog*). Этот элемент (Рисунок 61) обеспечивает ознакомление пользователя с некоторой инструкцией о его дальнейших действиях для того, чтобы он мог самостоятельно обойти ту или иную ошибку. Прочитал – согласился – нажал на кнопку «*ОК*». Элемент является полностью настраиваемым, потому в условиях отсутствия сигнала на входе «Надпись на кнопке» элемент будет давать в ответ только «*ОК*», но можно предусмотреть какой-то иной текст и подключить его к элементу.



Рисунок 61 – Маркировка оператора вывода диалогового окна с одной кнопкой (*One Button Dialog*) в разделе времени и диалогов (*Time & Dialog*) из перечня функций, размещаемых на блок-диаграмме

Для нормальной настройки диалога без *CASE*-структуры уже не обойтись (Рисунок 62). В случае, если шаг на входе нулевой – выдаём побуждающее сообщение нужного вида пользователю, он с ним соглашается – программа останавливает свою работу.

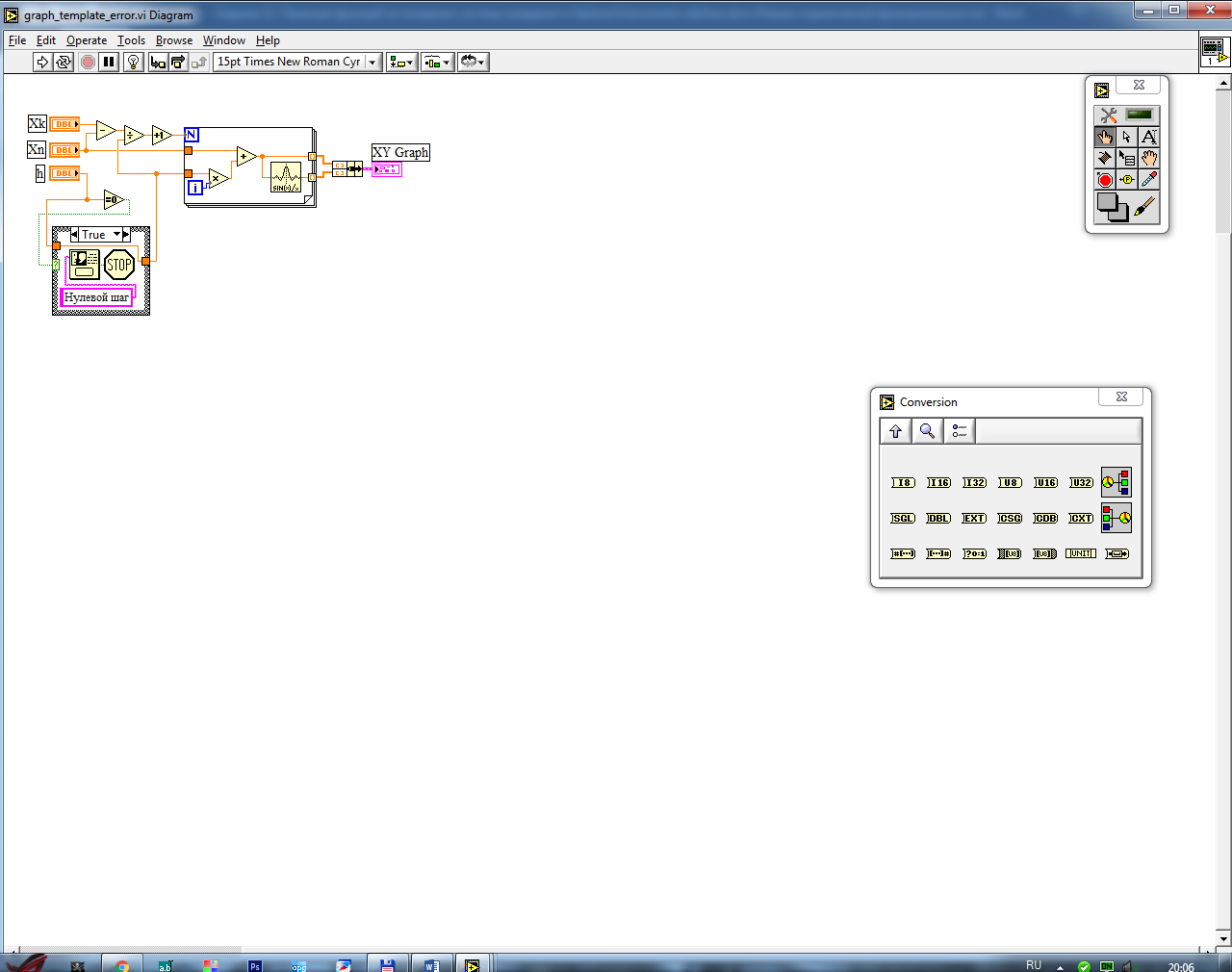


Рисунок 62 – Пример отлова ошибки нулевого шага с уведомлением пользователя об «останове» программы (блок истинного результата – запуск с нулевым значением шага построения)

В ином случае, если программа запущена с ненулевым значением шага в соответствующем блоке ложного результата *CASE*-структуры, должна быть размещена «закоротка» – шаг передаётся далее в программу без каких-либо особенностей (Рисунок 63). Ничто не должно этому препятствовать.

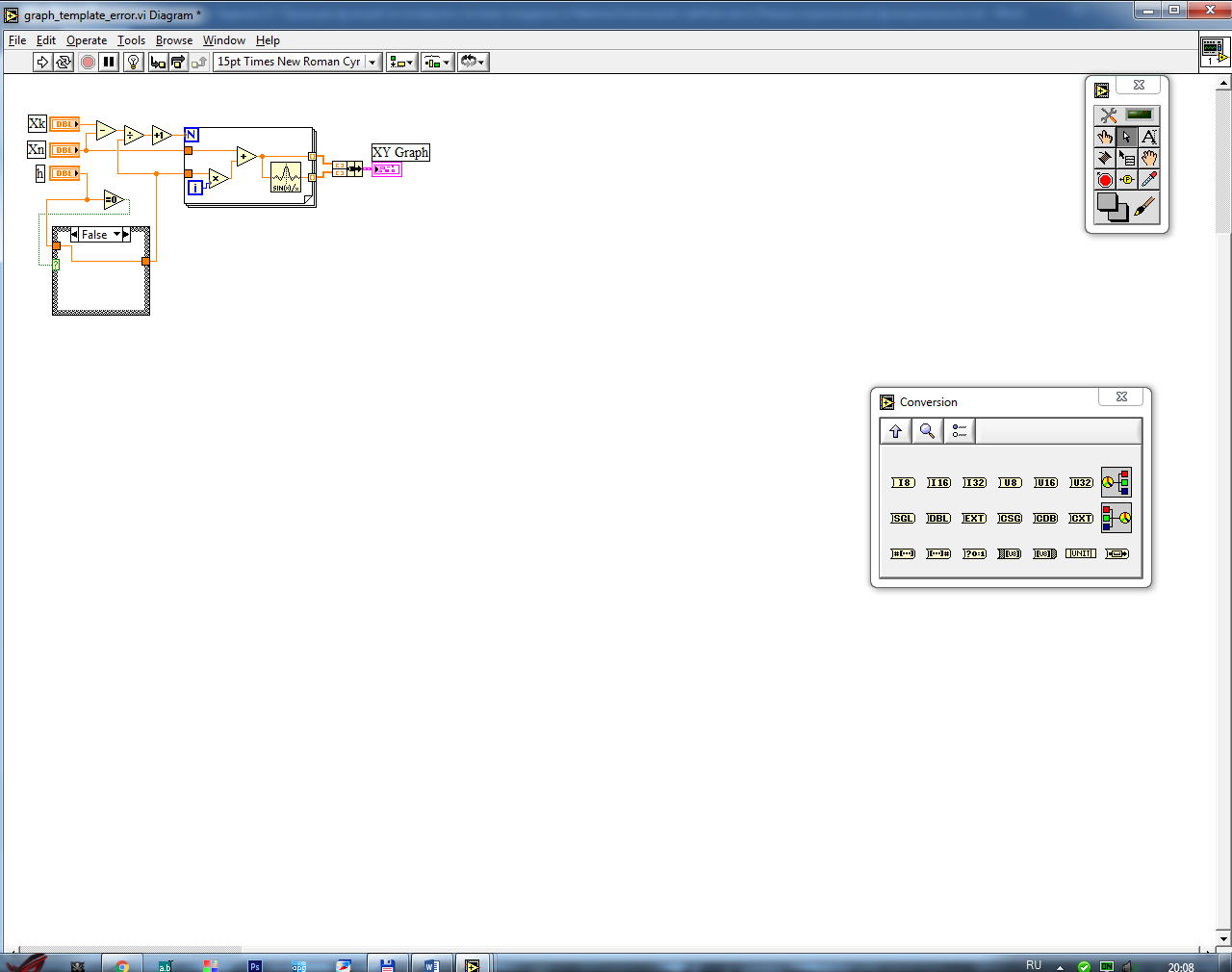


Рисунок 63 – Пример отлова ошибки нулевого шага с уведомлением пользователя об «останове» программы (блок ложного результата – запуск с ненулевым значением шага построения)

Результат работы диалога при нулевом значении шага на входе показан на Рисунке 64. Хорошо видно, что при отсутствии подключения чего-либо ко второму входу однокнопочного диалога, на кнопке содержится надпись «*ОК*».

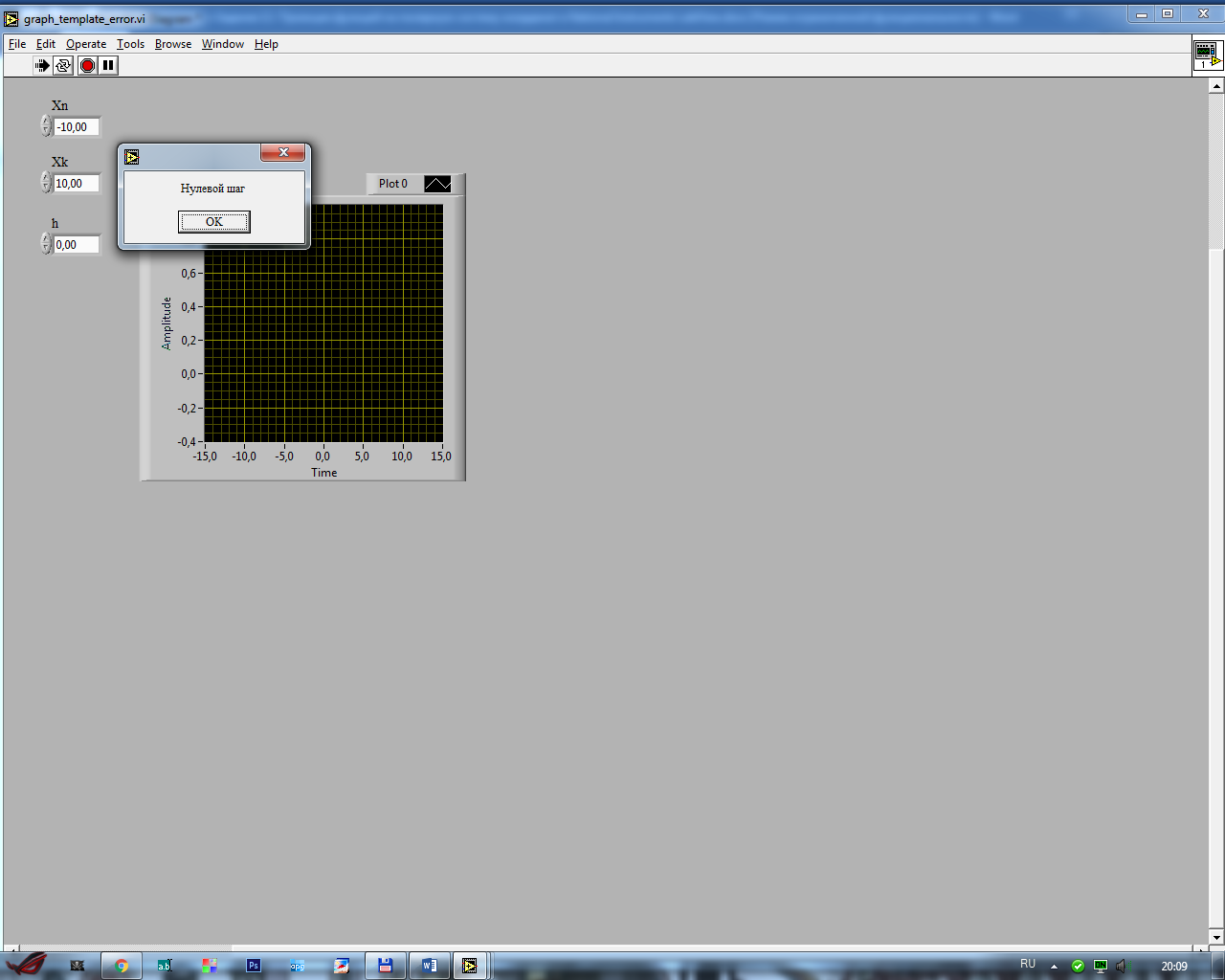


Рисунок 64 – Результат отлова ошибки и демонстрации диалога при запуске виртуального прибора с нулевым шагом построения графика функциональной зависимости

----------------------------------------------------------------------------------------------------

**Контроль разрыва периодической функциональной зависимости:**

Материалы, собранные в данном пункте, не относятся напрямую к выполняемой работе, поскольку варианты индивидуального задания в нём подобраны таким образом, что в них отсутствуют какие-либо точки разрыва. Однако, данный пункт может потребоваться обучающимся для выполнения других заданий Учебной практики.

В качестве примера «разрывной» функции рассмотрим функциональную зависимость вида *y(x) = 1 / sin(x)*. Будучи запрограммированной в исходном виде в цикле *For* (Рисунок 65), она даёт в результате картинку, не поддающуюся нормальному анализу (Рисунок 66).

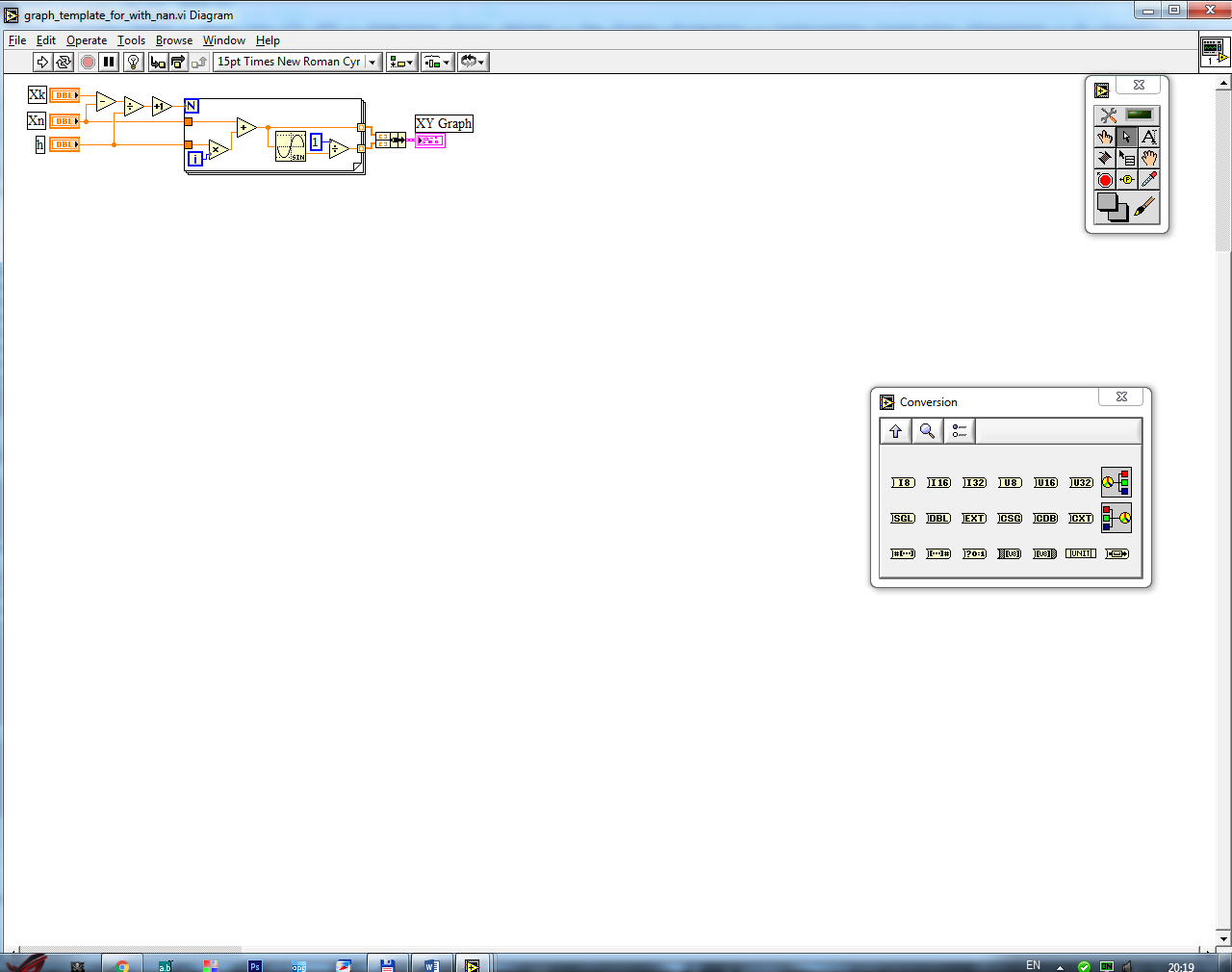


Рисунок 65 – Код «разрывной» функции *y(x) = 1 / sin(x)* на блок-диаграмме без учёта разрывов



Рисунок 66 – Результат построения графика функции *y(x) = 1 / sin(x)* без учёта разрывов

Для отлова точек разрыва вычислительный процесс должен быть подвержен разветвлению. В данном случае применение *CASE*-структуры ради выдачи одной единственной альтернативной вычислению константы избыточно, потому должен быть применён обыкновенный условный оператор, который в *National Instruments LabView* называется оператором выбора (*Select*). Он размещается в разделе сравнения как показано на Рисунке 67.

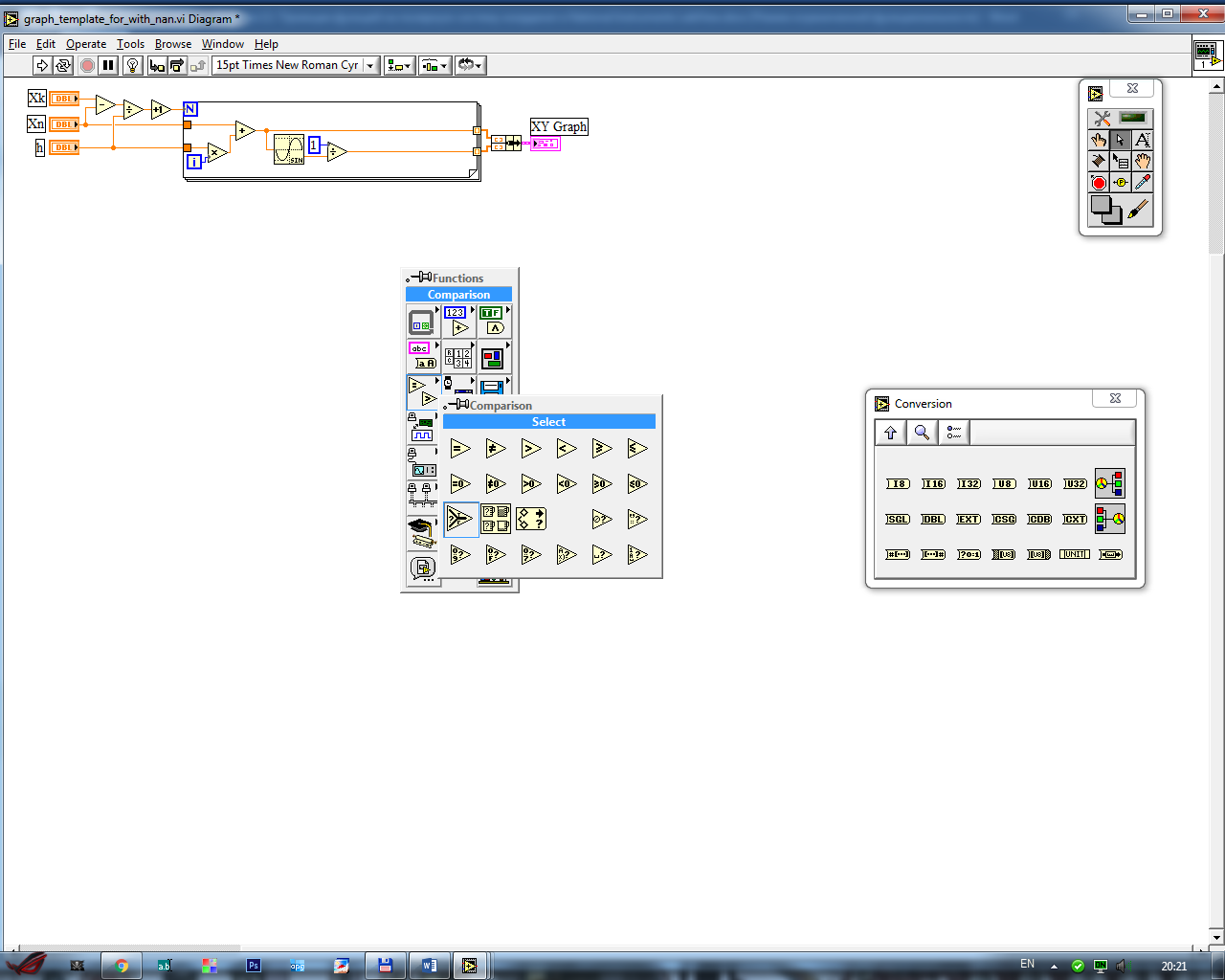


Рисунок 67 – Маркировка условного оператора *Select* в разделе сравнения из перечня функций, размещаемых на блок-диаграмме

Важно проконтролировать возникающий ноль в знаменателе, однако, при движении из одного произвольного значения в другое произвольное значение в точности ноль в знаменателе можно с лёгкостью проскочить, однако максимально к нему приблизиться, что тоже, вообще говоря, не очень хорошо. Это обстоятельство и даёт на выходе высокие всплески в сторону «+∞» и «-∞», но не ошибку. Таким образом отсекать нужно не только в точности ноль знаменателя, но и близкое приближение к нулю знаменателя. И здесь математика, как правило, призывает говорить о том, что существуют значащие и не значащие величины. Собственно, не значащими называются величины, точными значениями которых можно пренебречь. Например, при шаге 0.01 значение 0.0025 пренебрежимо мало и тоже может рассматриваться как 0 знаменателя.

Таким образом, далее рассматривается схема вычислений с заданной точностью. В ней шаг построения диктует эту самую, заданную точность. В этом случае всё, что оказывается меньше шага по абсолютной величине / модулю (нам не важно, слева ли вблизи нуля мы оказались или справа) считается нулём.

На Рисунке 68 представлен код, размещённый на блок-диаграмме, соответствующий вычислениям с заданной точностью.

Для того, чтобы разрыв детектировался именно как разрыв в области для построения графиков функциональных зависимостей, необходимо вводить точку нечисловых значений *Not a Number* (*NaN*). В этом случае среда начинает распознавать различные фрагменты одной и той же функциональной зависимости в рамках одной трассировки. То есть точка (точки) в массиве существует(-ют), однако значение(-я) в ней (них) не определено(-ы). Если эти точки просто проигнорировать, не вычислять их значения, то разрыва всё равно не получится, поскольку подряд будут записаны числовые значения. Два соседних числовых значения могут быть соединены линией – постулат. Когда два числовых значения разделены не числовым значением, то нечего соединять – никому не ясно, как такое соединять. С этим не справится даже *LabView*.

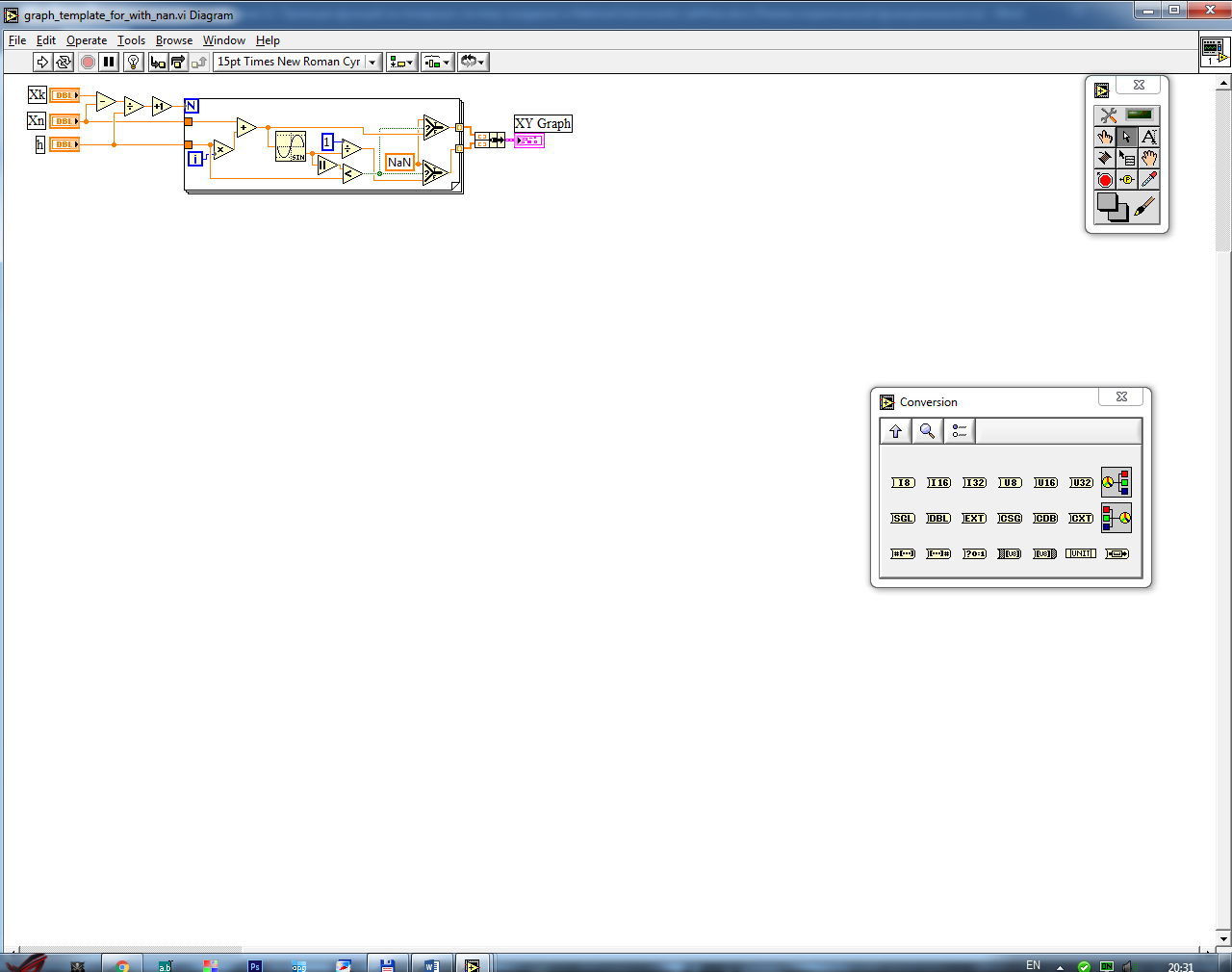


Рисунок 68 – Код «разрывной» функции *y(x) = 1 / sin(x)* на блок-диаграмме с учётом периодических разрывов

График, построенный с заданной точностью, приобретает приличный вид и может быть подвергнут анализу (Рисунок 69).

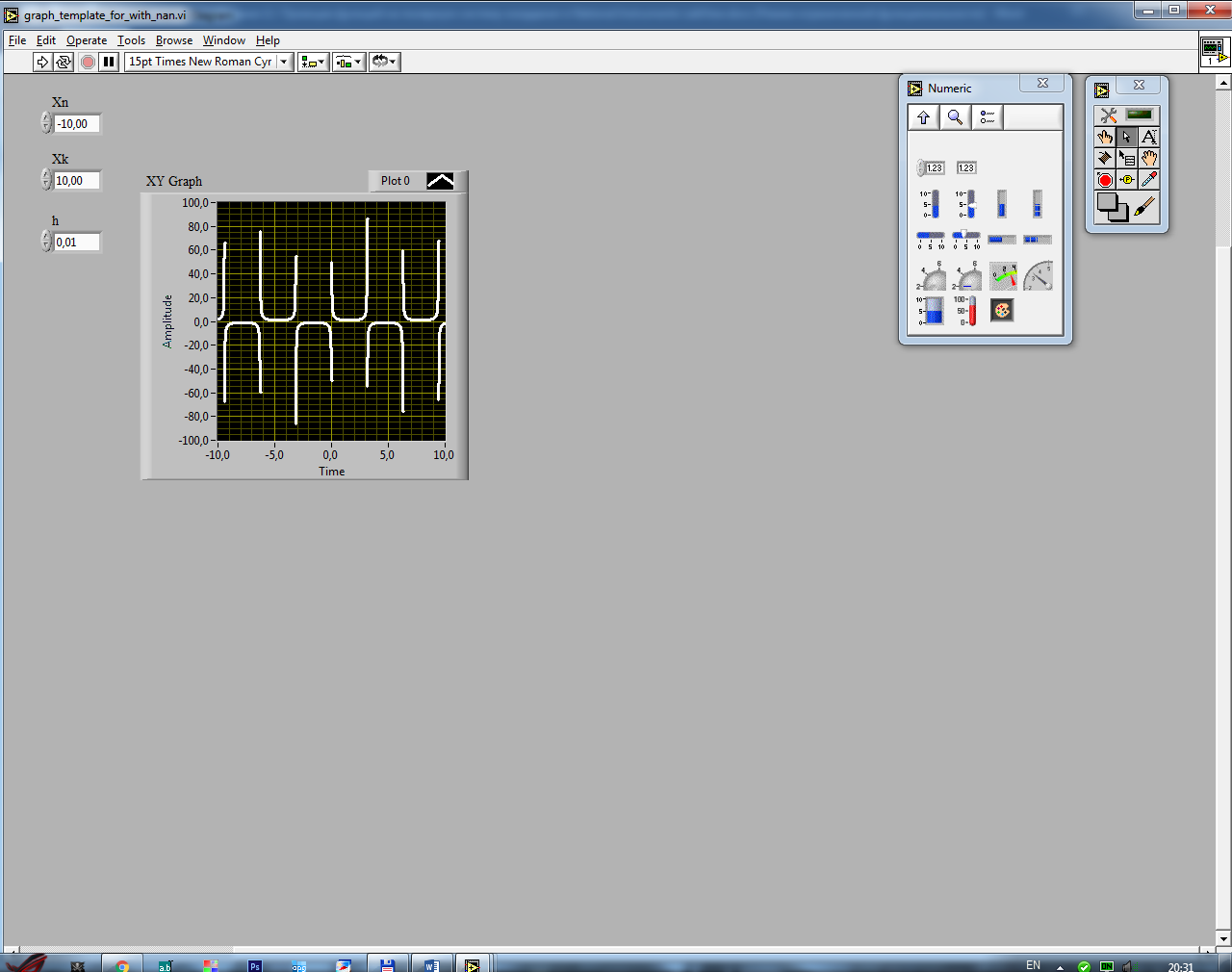


Рисунок 69 – Результат построения графика функции *y(x) = 1 / sin(x)* с учётом разрывов

----------------------------------------------------------------------------------------------------

**Примеры построения графиков функциональных зависимостей в полярной системе координат:**

Рассмотрим проецирование функциональной зависимости в полярную систему координат, воспользовавшись указанными ранее полезными соотношениями. В качестве функциональной зависимости взята всё та же функция отсчётов и, задавшись целью не плодить большого количества элементов, возьмём объединённую функцию синуса и косинуса. Умножение значения функции на косинус абсциссы – это проекция, соответственно, на абсциссу; умножение значения функции на синус абсциссы – это проекция на ординату (Рисунок 70). Для численного построения использован метод, основанный на структуре цикла *For*.

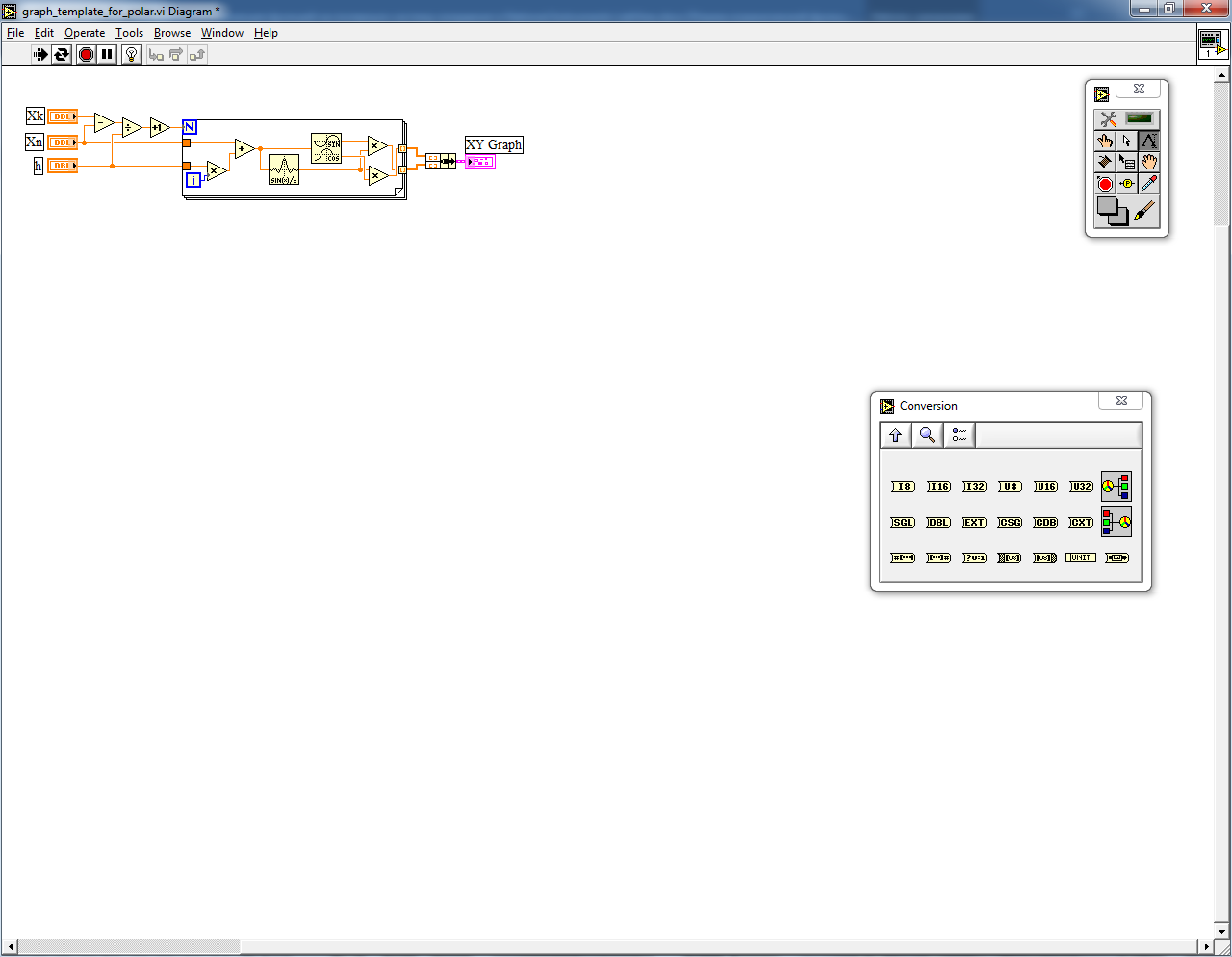


Рисунок 70 – Проецирование в полярную систему координат функции отсчётов

Для симметричного представления графика функциональной зависимости в полярной системе координат область построения задаётся по направлению из отрицательной в положительную. Для примера рассмотрим диапазон от -2π до +2π (Рисунок 71).

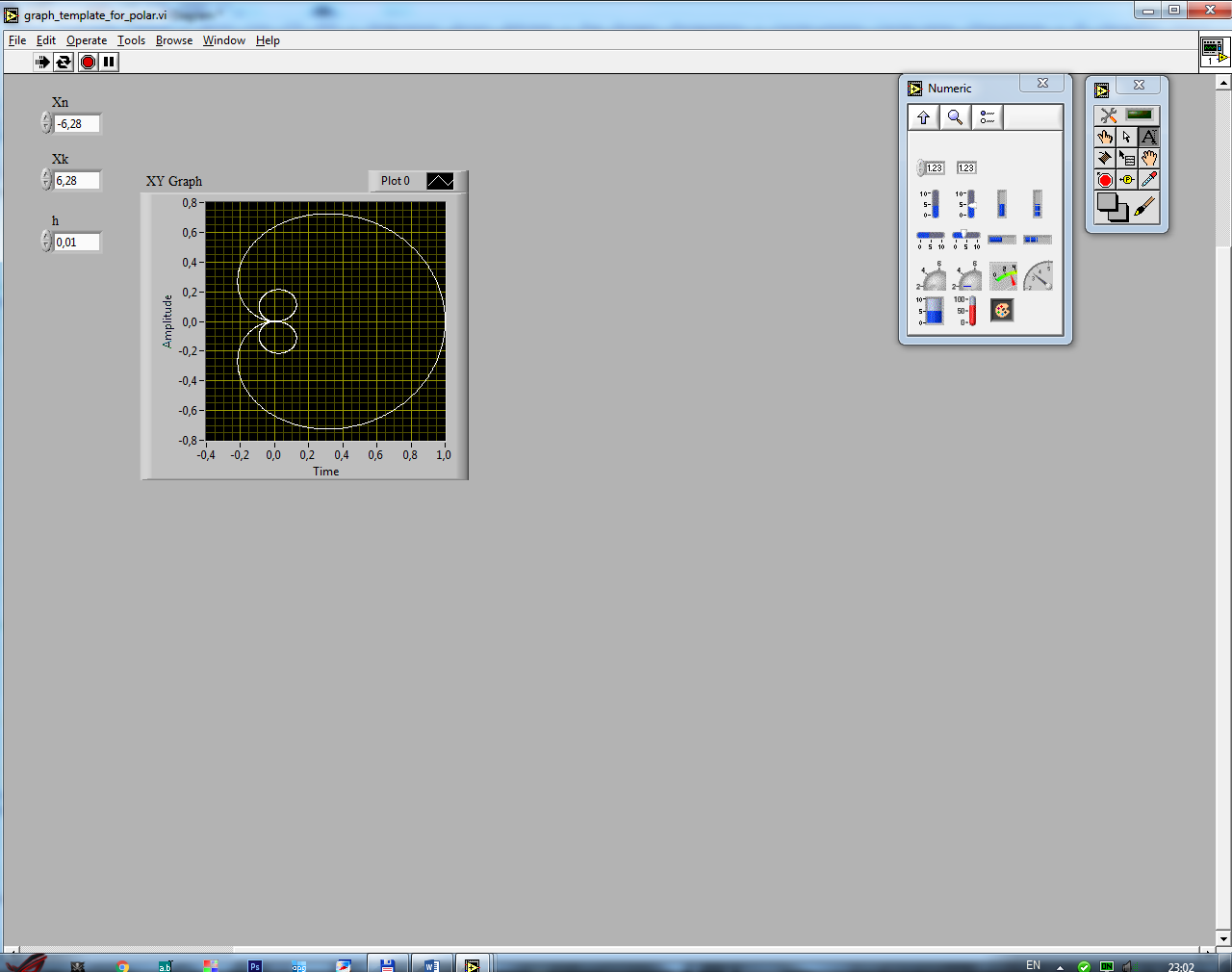


Рисунок 71 – Симметричное представление функции отсчётов в полярной системе координат

Как правило, в технике используется не симметричное представление функциональных зависимостей в системе полярных координат – рассмотрение ведётся от нулевого значения по абсциссе, принимая во внимание тот факт, что если, например, абсцисса – это время, то ни один реальный процесс на Земле не существует в отрицательном времени. Отрицательное время может быть лишь математической условностью. Таким образом, реальное представление рассматриваемой функции отсчётов в полярной системе координат имеет вид, представленный на Рисунке 72.

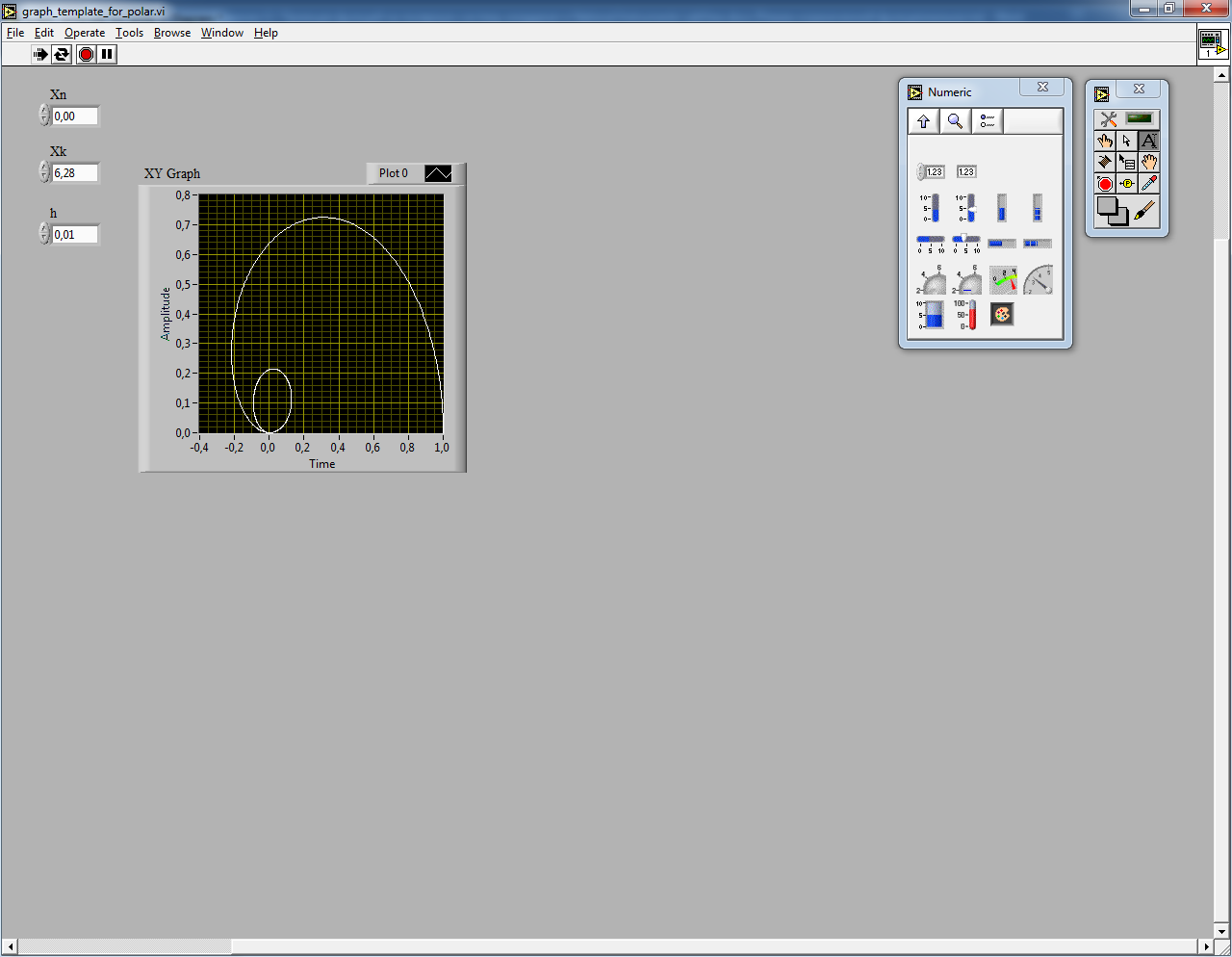


Рисунок 72 – Реальное, не симметричное представление функции, спроецированной в полярную систему координат

Наиболее часто встречающейся ошибкой обучающихся при построении проекции графика функциональной зависимости в полярной системе координат является подмена полярной функцией полярного аргумента и наоборот. При этом график функциональной зависимости поворачивается на 90 градусов от истинного своего представления. В связи с этим не лишними бывают проверки результатов построения при использовании различных пакетов прикладных программ, позволяющих выполнять проекцию напрямую, без использования полезных соотношений. Одним из таких пакетов прикладных программ является *MathCAD*.

Для проверки полученных результатов построение выполнено и в осях Декартовой системы координат в не симметричном представлении, а также в полярной системе координат с использованием специализированного компонента, учитывающего только реальные представления функций. По результатам проверки хорошо видно, что в *National Instruments LabView* результат получен правильный (Рисунок 73).

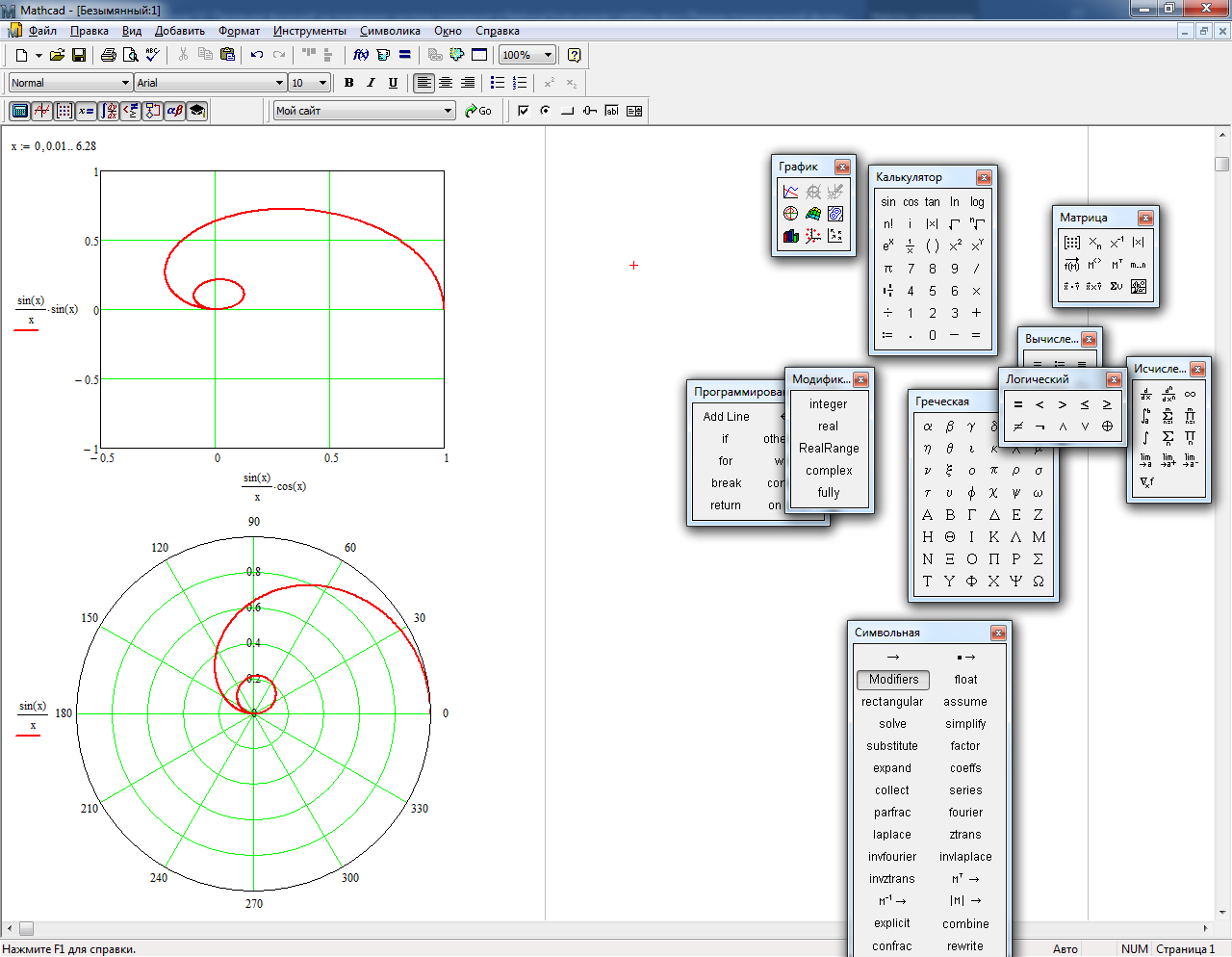


Рисунок 73 – Проверка построения графика функциональной зависимости в пакете прикладных программ *MathCAD*.