Эти советы помогут вам успешно запустить КМОП матрицу EV76C661 (или нет).

Читайте внимательно datasheet. Если вы не поняли с первого раза, как запустить матрицу, то не переживайте, поскольку его оформление, мягко говоря, неоптимальное. Матрица не запустится даже с default настройками :)

ГЛАВА 1. НАЗНАЧЕНИЕ РЕГИСТРОВ.

Прежде чем что-либо делать, изучите внимательно значения регистров. Их много, но большинство предназначены для режимов ROI. Для нас важен полный кадр без обрезки. Следовательно, выбор падает на ROI1. Также нам важна настройка клокового дерева, поскольку значения default делителей частоты не позволяют запустить матрицу на частоте 10-12 МГц. Выбор данного диапазона связан с особенностями испытаний на ТЗЧ.

Следующие регистры должны быть задействованы в обязательном порядке.

h07

Данный регистр необходим для проверки вывода кадра. Значение default 2561 позволяет выводить реальный кадр. Однако, например, значение 2593 выводит test pattern, примеры которого указаны в datasheet (другие значения указаны в описании регистра на стр. 48). Если вы правильно оперируете данными кадра, то изображение должно сохраняться.

h08

Данный регистр позволяет настроить частоту CLK_CHAIN и CLK_CTRL. Значение default 56087, при этом оно не позволит запустить матрицу на частоте CLK_REF = 12МГц. Рекомендуемое значения регистра 56217. Также можно использовать максимальное значение 56319, при этом значительно повышается чувствительность матрицы к освещению.

h09

Данный регистр отвечает за настройку коэффициента CLK_PPL = Коэф × CLK_REF. Значение default 24869, при этом оно не

позволит запустить матрицу на частоте CLK_REF = 12МГЦ. Рекомендуемые значения 25976 (Коэф = 9), 25961 (Коэф = 8). Подсказка: чтобы настроить частоту CLK_REF на 12МГц, настройте частоту Timed Loop на 48 МГц (частота 12 МГц в FPGA имеет некратное двум деление, вследствие чего период частоты будет иррациональным числом) и создайте счетчик от 0 до 2, который будет по отсчету 3х тактов выдавать новый сигнал. Тем самым, таким нехитрым образом создать делитель частоты.

h0A

Данный регистр позволяет настроить выбор режима ROI. Значение default 512 уже настроено на режим ROI1, при этом в течении запуска оно не изменялось. Если вам требуется использовать другой режим ROI или биннинг, то уточняйте значение в описании регистра (стр. 53).

h0B

Данный регистр отвечает за включения TRIG, что крайне важно для получения кадра. Если вы поспешили и решили поверхностно разобраться с назначением регистров, то вы могли аппаратно подавать TRIG, но не получать кадр. Этот момент был крайне мучительным :) Значение default 5. Для возможности подачи TRIG требуется записать значение 260. Если вы хотите использовать TRIG для получения кадра в режиме постоянного снятия кадра без аппаратной подачи сигнала TRIG, то запишите в регистр значение 262. ВАЖНО. Если сигнал TRIG не был подан, то все регистры, кроме регистра h0B, будут ПРЕДЗАПИСАНЫ. После подачи сигнала TRIG значения будут ЗАПИСАНЫ. Также следует добавить возможность регулирования времени TRIG. Примечание: поскольку вы можете подать сигнал TRIG в то время, когда вывод кадра еще не завершился (в это время TRIG игнорируется), следует аппаратно подавать сигнал для синхронизации с записью и выводом кадра.

h0E

Данный регистр отвечает за время экспозиции. Значение default 512. Можно подобрать подходящее для вас время для светлого/темного кадра.

ГЛАВА 2. РАБОТА SPI.

При программировании SPI контроллера следует иметь ввиду, что у матрицы существуют как регистры с данными длиной 8 бит, так 16 бит. В связи с этим необходимо создать гибкий код FPGA, который сможет сам переключаться между режимами.

ВАЖНО. Если вы создаете контроллер без использования FIFO для общения по SPI, то имейте ввиду, что чтение FPGA отстает от записи в зависимости от частоты цикла Timed Loop. До частоты 10 МГц отставание составляет 3 итерации цикла, на частоте 40 МГц 5 итераций, на частоте 80 МГц 6 итераций. Если, к примеру, установить частоту 10 МГц, замкнуть электрически выводы чтения и записи и подавать логическую единицу на чтение с нулевой итерации, то только на третью итерацию FPGA сможет прочитать значение. Почему так работает, только господу богу известно, однако это надо учитывать, иначе вы сделаете идеальную диаграмму сигналов в симуляции FPGA, а на деле будете видеть сдвиг битов в MISO.

Объяснение данной ситуации представлено на рисунках 1 и 2. Сделан цикл, где чтение и запись разделены в кейсе. В "0" подаем сигнал, в "1" читаем, в "2" выключаем сигнал. Тут можно выставить итерацию чтения. Как видно, слева на панели индикаторы первого цикла, справа – второго. Задержка по чтению в таких циклах одинаковая.

Вердикт:

Читать и записывать одновременно = читать и записывать последовательно в данном случае. Надо только учитывать время “раздупления” канала Read длиной 3 итерации цикла с частотой до 10 МГц. Если нужна высокая частота, то при 40 МГц – 5 итераций, при 80 МГц – 6 итераций

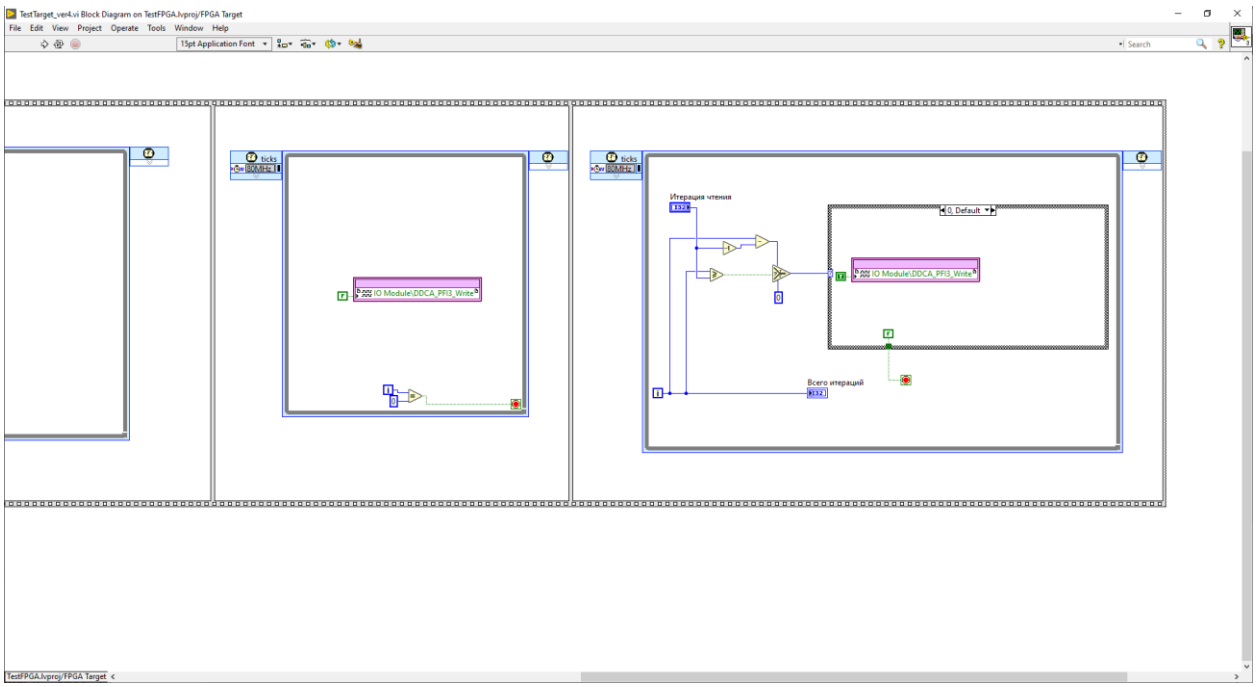

Рисунок 1а. Код программы.

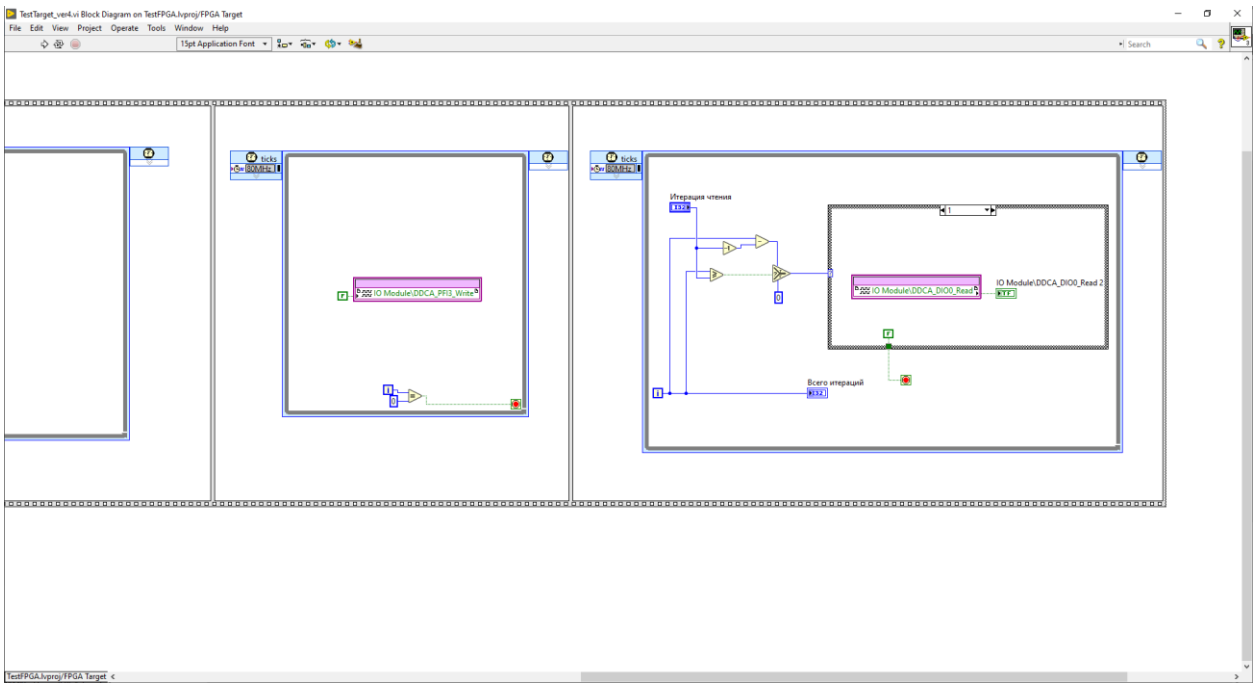

Рисунок 1б. Код программы.

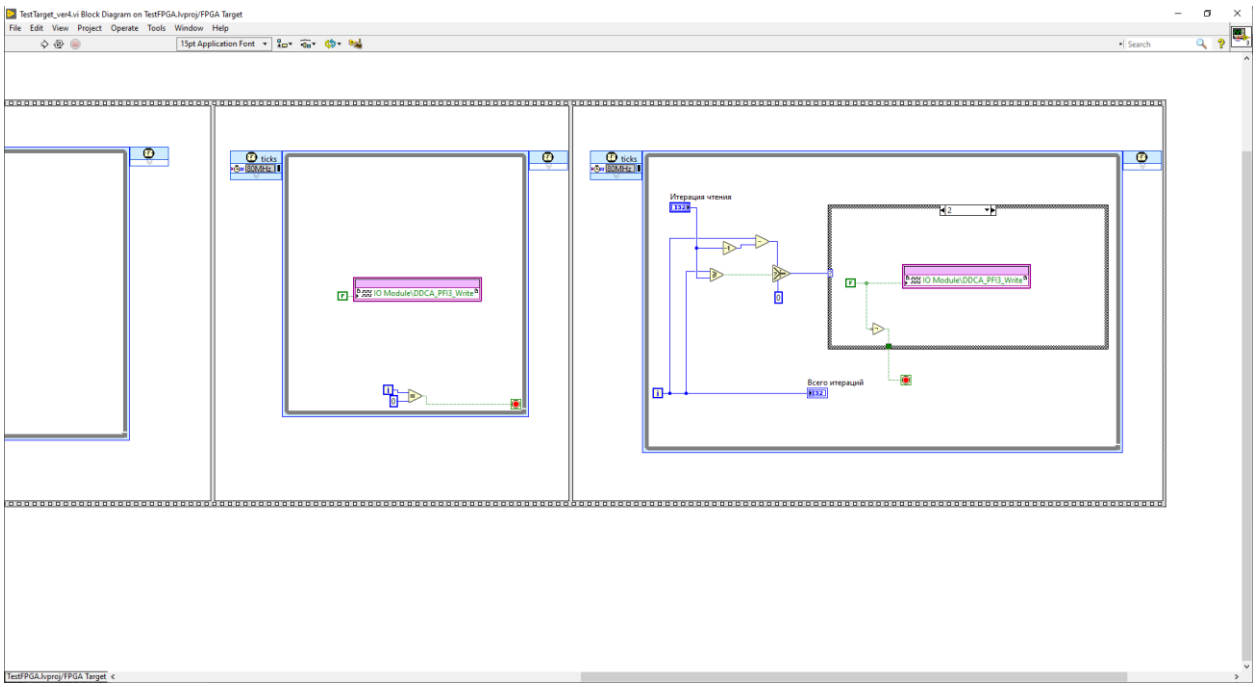

Рисунок 1в. Код программы.

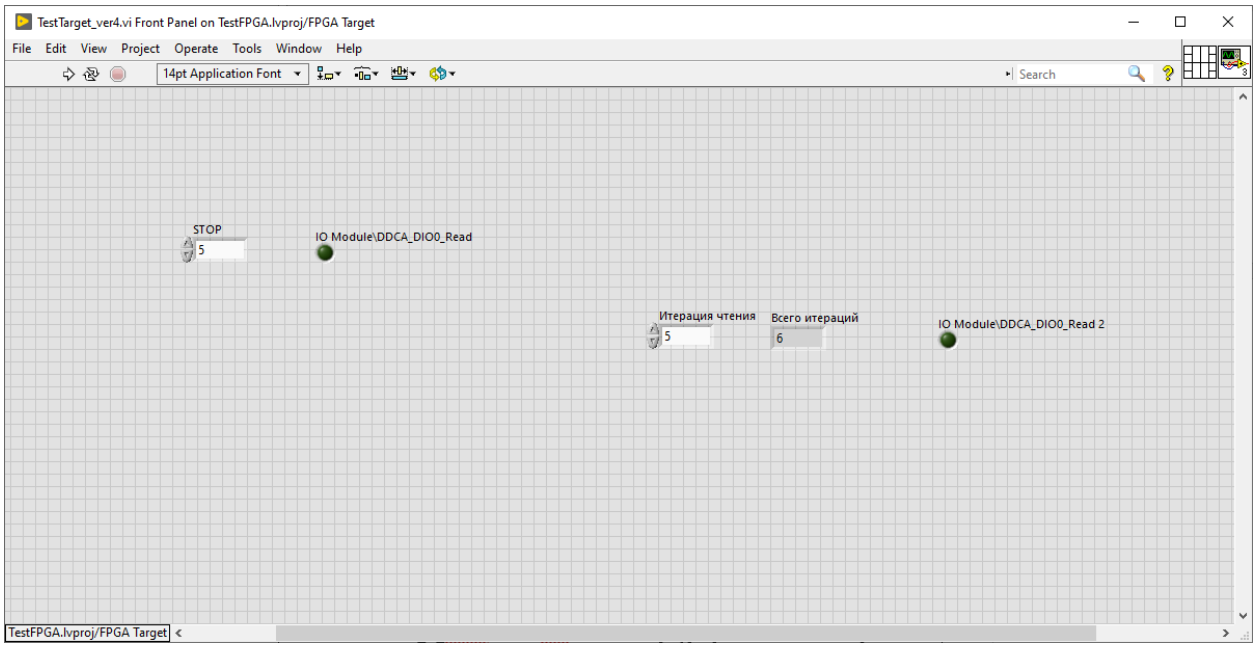

Рисунок 2а. Результат программы при STOP = 5.

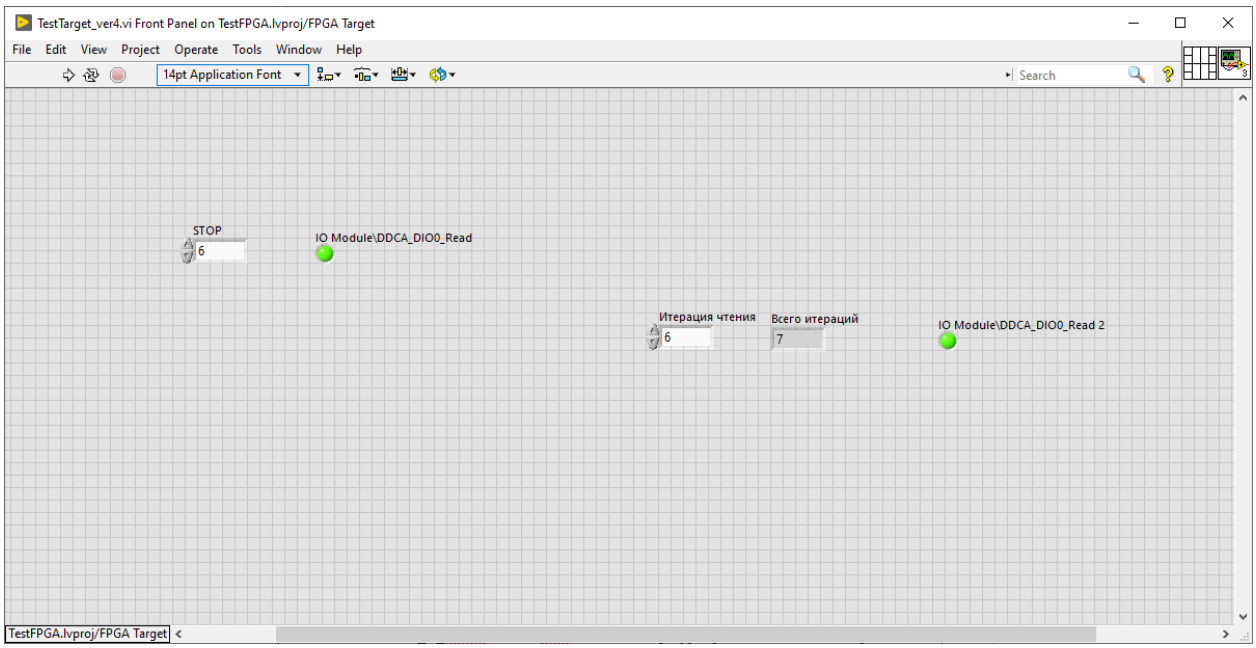

Рисунок 2б. Результат программы при STOP = 6.

ГЛАВА 3. ЧТЕНИЕ И ЗАПИСЬ.

В FPGA имеется Memory и FIFO. Первое, очевидно, память FPGA, куда можно записать в адрес переменную. Второе тоже память, однако она работает по принципу “первый зашел – первый вышел”. FIFO будет выступать в роли буфера из Host в Target, при этом в отличие от Memory будет доступно гораздо меньшее адресное пространство. Тем самым кадр записывается весь в Memory, а данные из Host в Target отправляются по частям. Перед созданием автомата FIFO поймите, сколько всего адресов у вас будет задействовано в Memory, и после уже поймете, сколько всего раз вам придется вызывать буфер. Примечание: с увеличением требуемого значения бит для записи переменных в FIFO уменьшается и доступное количество его памяти.

ГЛАВА 4. РЕКОМЕНДАЦИИ ДЛЯ СОЗДАНИЯ HOST

В отличие от Target, сследует оформлять Host по принципу параллельных потоков. Вы можете делать программу и по принципу конечных автоматов, но тогда ваша логика будет жестко завязана на происходящих

событиях, при этом все действия будут выполняться не параллельно, а последовательно, что снижает оптимальность кода Host.

В коде Host будет содержаться множество циклов While, которые отвечают только за свой функционал. Например, есть отдельные циклы для управления источниками питания, включения/выключения питания, записи файла логов и парирования и пр. На рисунке 3 представлен пример оформления кода программы. Здесь наверху представлен отдельный цикл контроля сбоев при испытательном воздействии на образец, а ниже расположен цикл прошивки регистров. Если все регистры были проверены и при этом обнаружен хотя бы один сбой, то матрице подается сигнал RESET, который возвращает значения default. При этом по сигналу RESET начинает работать и кадр в цикле, отвечающем за прошивку регистров. В данном случае регистры можно прошить в любой момент за счет разделения процессов по циклам.

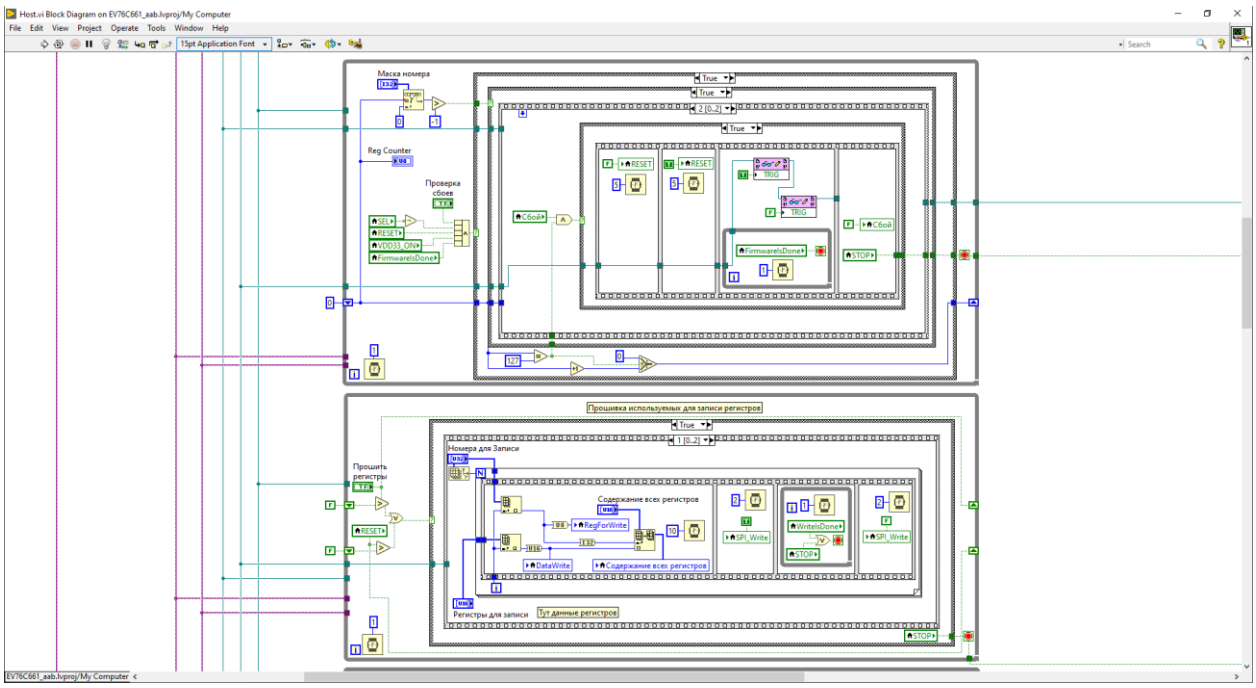

Рисунок 3. Пример оформления кода в Host.

Такой подход помогает создавать отдельные элементы управления, как например, отдельные команды чтения и записи, которые будут созданы один

раз и далее будут вызываться при помощи сигнала вызова. Иначе говоря, каждый цикл – отдельная функция, которую вы применяете в определенный момент или на протяжении всего времени выполнения программы. Однако для синхронизации процессов необходимо создавать множество индикаторов, свидетельствующих о готовности перехода к тому или иному процессу. Например, на рисунке 4а представлен кадр, где индикатор FirmwarelsIsDone сообщает о готовности прошивки регистров. В это время в цикле контроля сбоев на рисунке 4б в цикле While происходит ожидание готовности индикатора. Таким образом, не надо думать о конкретных временных задержках, поскольку цикл сам продолжит работу, когда будет сигнал. Примечание: для внештатных ситуаций следует добавлять отключение таких циклов по команде STOP.

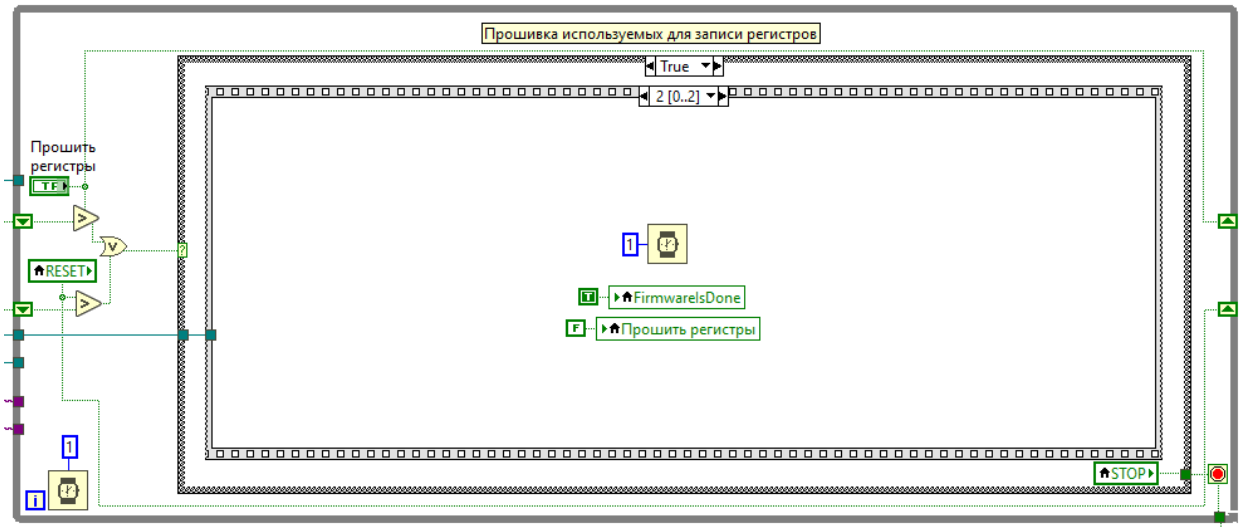

Рисунок 4а. Применение сигналов синхронизации.

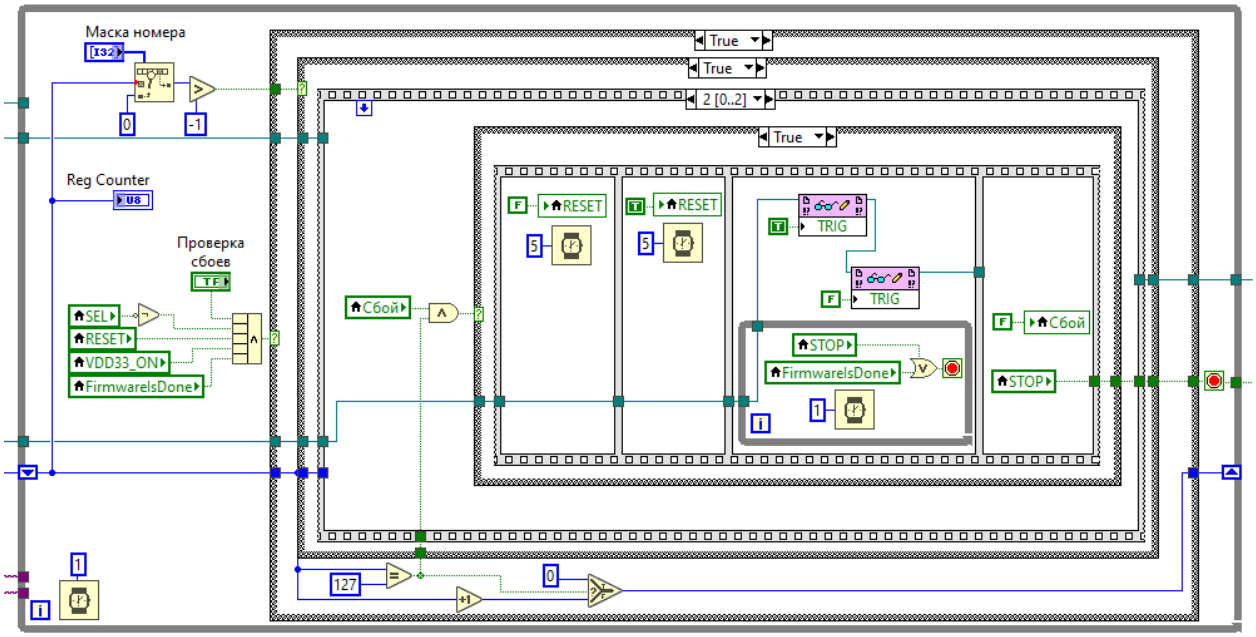

Рисунок 4б. Применение сигналов синхронизации.