ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В.ЛОМОНОСОВА»

ФИЗИЧЕСКИЙ ФАКУЛЬТЕТ

КАФЕДРА ОБЩЕЙ ЯДЕРНОЙ ФИЗИКИ

БАКАЛАВРСКАЯ РАБОТА

«РАЗРАБОТКА ТОПОЛОГИИ ПЛИС, СОВМЕЩАЮЩЕЙ СХЕМУ БЫСТРЫХ СОВПАДЕНИЙ И ПРЕОБРАЗОВАТЕЛЬ ВРЕМЯ-КОД ДЛЯ КОМПАКТНОГО СПЕКТРОМЕТРА»

Выполнила студентка
413 группы
Максимова Надежда Евгеньевна

_____

Научный руководитель:
к.ф.-м.н. Чепурнов Александр Сергеевич

_____

Допущена к защите
Зав. кафедрой _____

МОСКВА
2021

Оглавление

ВВЕДЕНИЕ

Исследование распределения ионизирующего излучения в пространстве и времени требует регистрации ионизирующих частиц - источников ионизирующего излучения с учетом их количества, энергии и типа. Это одна из наиболее распространенных задач в прикладной ядерной физике. Для ее решения используются специальные ядерно-фзические приборы - спектрометры ионизирующего излучения.

Рассмотрим классический спектрометрический тракт на основе сцитилляционного детектора, широко применяемого для регистрации ионизирующего излучения. Такой спектрометрический тракт состоит из сцинтиллятора, устройства регистрации фотонов сцинтилляции (например фото-эдектронного умножителя (ФЭУ)), усилителя сигнала с ФЭУ, спектрометрического усилителя и аналого-цифрового пребразователя (АЦП). Пролетающая через сцинтиллятор частица тратит свою энергию на ионизацию и возбуждение среды, что вызывает рождение фотонов сцинтилляции. Их количество пропорционально энергии, переданной пролетающей частицей среде детектора. Фотоны сцинтилляции регистрируются с помощью ФЭУ, площадь импульса тока на выходе которого пропорциональна количеству фотонов. Затем этот сигнал усиливается и подается на дальнейшую обработку с помощью спектрометрического усилителя и АЦП. АЦП позволяет измерить амплитуду сигнала, которая, при правильной настройке спектрометрического усилителя, пропорциональна его площади. В итоге можно записать такую цепочку пропорциональностей:

$$E \sim N_{photons} \sim S_{signal} \sim A_{signal},$$

где E - это энергия зарегисттрованной частицы, $N_{photons}$ - число фотонов сцинтилляции, S_{signal} - площадь сигнала, а A_{signal} - его амплитуда. Таким образом, вычисляя с помощью АЦП распределение по амплитуде сигнала, простым линейным преобразованием можно восстановить изначальное распределение частиц по выделенной в среде энергии. Предполагается, что внутренние нелинейности АЦП, предусилителя и ФЭУ не велики и ими можно пренебречь в определенном диапазоне энергий, выделенных в детекторе.

Необходимо отметить, что за последние 50-60 лет все элементы спектрометрического тракта до АЦП не претерпели принципаиальных изменений. Развитие же микроэлектроники и увеличение производительности компьютерных систем координальным образом расширили способы преобразования аналоговых сигналов в цифровые, методы регистрации и обработки полученыех в результате цифровых данных.

АЦП - это устройства, которые преобразуют амплитуду входного напряжения в цифровой код [1]- [2]. Этот процесс может осуществляться разными методами. Основной принцип работы - это сравнение входного напряжения с эталонным. После пропускания входного сигнала через последовательный или параллельный набор таких сравнений на выходе получается двоичное число, которое затем можно зафиксировать для дальнейшей обработки.

Основные характеристики АЦП - это его быстродействие и разрядность. Чем выше разрядность АЦП, тем точнее он может измерять значение напряжения сигнала в данный момент времени.

В последовательном АЦП сигнал многократно подается на один и тот же компаратор, но на каждом такте он имеет разные референсные значения напряжения. Входной сигнал последовательно сравнивается с ними, и на выходе получается двоичное число. Такие АЦП являются медленными: для поддержания разрядности порядка N требуется N последовательных сравнений. К тому же, за время одного такта оцифровки дожны успеть сработать несколько электронных компонентов, что также уменьшает быстродействие.

В параллельном АЦП сигнал разветвляется и подается на набор компараторв, перед каждым из которых стоит свой делитель напряжения. Это увеличивает скорость оцифровки. Однако, чтобы обеспечить разрядность АЦП порядка N, необходимо использовать $(2^N - 1)$ компараторов и столько же делителей. Например, для 12-разрядного АЦП требуется 4095 компараторов. Это увеличивает потребление тока, а также сложность внутренней топологии микросхемы. К тому же, чтобы поменять алгоритм обработки входного сигнала, необходимо физически менять конструкцию печатных плат АЦП, что в затртано и трудоемко.

Однако, если форма импульса на выходе спектрометрического тракта известна и стабильна, то для обработки сигналов можно применить гипотезу о том, что площадь импульса пропорциональна не только амплитуде сигнала, но и его ширине на определенной высоте относительно максимальной амплитуды. Если гипотеза верна, то измерение площади импульса можно проводить путем измерения длительности сигнала в основании импульса. Для реализации такого метода на практике необходимо исследовать функцию, связывающую длительность сигнала (ширину импульса) на определеной высоте и площадь сигнала. Такой метод не требует использования АЦП и позволяет заменить его на один быстрый компаратор и устройство измерения временных интервалов между последовательным срабатыванием компаратора на фронте и спаде импульса, площадь которого пропорциональна заряду. Устройство измерения длительности импульса можно эффективно реализовать на программируемой логической интегральной схеме (ПЛИС), которая выполняет функцию преобразователя время-код (Time to Digital Converter, TDC). Такой подход к конструированию спектрометра позволяет снизить число используемых электронных компонентов, что уменьшает размеры и энергопотребление всего спектрометра, а также упрощает его конструкцию.

Малое энергопотребление и компактные размеры спектрометра крайне важны при его использовании в таких приложениях, как легкие носимые приборы или малые космическихе аппараты типа кубсат. На кубсатах электроника должна быть малопотребляющей и высоконадежной,

а еще лучше - радиционно-стойкой, что достигается минимизацией количества примняемых электронных компонентов. Поэтому применение TDC на основе радстойкой ПЛИС вместо классического АЦП выглядит привлекательным и перспективным. Данный метод позволяет снизить количество потребляемых ресурсов, а также упростить архитектуру спектрометра. К тому же применение ПЛИС позволяет менять алгоритмы обработки данных без физического вмешательства в конструкцию, что делает спектрометр более гибким и универсальным.

Другое возможное применение подобной архитектуры спектрометра - это дозиметрия. Обычные дозиметры работают только в счетном режиме, измерение энергии излучения в них невозможно. В связи с этим, эффективная доза или не может быть измерена совсем или измеряется с учетоем модельных оценок и априорных знаний об измеряемых полях ионизационного излучения. Измерении эквивлентной дозы с помощью предлагаемого метода увеличивает точность и объективность ее оценки, а компактная конструкция спектрометра позволяет использовать его в носимых устройствах для персонального использования. Данное применение крайне важно для контроля безопасности персонала на радиационно опасных объектах.

1 Способы реализации TDC в спектрометрах различного назначения

Метод TDC широко применяется в различных областях для исследования физичеких процессов в которых информация содержится в длительности импульсов и времени прихода отдельных импульсов: в физике высоких энергий, медицинской физике, флоурисцентной спектроскопии в химии и биологии. Задачи, решаемые с помощью TDC определяют разные требования к разрешающей способности по времени и скорости обработки данных, поэтому реализация метода сильно зависит от конечного приложения. При этом организация работы TDC с помощью ПЛИС также весьма распространена. Это вызвано тем, что ПЛИС позволяют быстро создавать прототипы устройств, проводить реконфигурацию, а также, в зависимости от модели, они имеют невысокую цену.

Одной из областей, где широко применяется TDC, является ПЭТ-диагностика [3]- [4], где необходима быстрая обработка сигналов. ПЭТ-томограф регистрирует гамма-кванты, образованные после e^+e^--аннигиляции радиоактвного изотопа, введенного в пациента. С помощью схемы совпадений измеряет разницу во времени прилета аннигиляционных гамма-квантов в детекторы. Это позволяет построить точное трехмерное изображение локализации радиоактивного изотопа в теле человека.

Для измерения энергии гамма-квантов применяют сцинтилляционные детекторы, сигнал с которых можно обработать разными способами. Например, с помощью измерения длительности сигнала на различных высотах [5]. Для этого сигнал с детектора разветвляется и подается на вход нескольких компараторов, в которых выставлены разные уровни порогов. Затем сигналы с компараторов направляются на обработку в ПЛИС, где измеряется их длительность. С помощью такой дискретизации можно приближенно измерить площадь

сигнала, а затем - и энергию гамма-кванта. Также это позволяет точнее определять время срабатывания детектора, что необходимо для более точной реконструкции изображения.

Также метод TDC применяется в физике высоких энергий во время-пролетных экспериментах. Например, в эксперименте ALICE Большого Адронного коллайдера [6]. Здесь этот метод позволяет идентифицировать частицы, образованные в точке столкновения двух пучков протонов. Это можно сделать, измерив одновременно скорость и импульс частицы, зная которые, можно вычислить ее массу. Скорость частицы вычисляется с помощью измерения времени пролета частицы между двумя заранее известными точками. Именно в этой части обработки сигнала применяется TDC. Так как скорости частиц являются релятивистскими, разрешение TDC должно достигать пикосекундных масштабов.

Разные архтектуры TDC также активно применяются для реализации метода TOT измерений (Time Over Threshold, "Длительность сигнала выше некоторого порога") [8]- [9]. Он также используется в физике высоких энергий для идентификации частиц.

Алгоритм TDC может быть реализован и с помощью интегральной схемы специального назначаения (Application-Specific Integrated Circuit, ASIC) [7]. С ее помощью можно проводить обработку данных на высоких скоростях и при больших загрузках, что очень важно в физике высоких энергий. Это достигается за счет того, что вся конструкция ASIC разрабатывается для определенной задачи, благодаря чему в ней согласованы времена задержек между компонентами и с тактовой частотой, которая достигает сотен МГц. По этой же причине она расходует энергию более эффективно, чем ПЛИС, что делает ее перспективной для использования в компактных детекторах и на космических аппаратах.

Однако, создание ASIC является намного более сложной задачей, чем разработка конфигурационного файла для ПЛИС. Она требует не только написание кода, описывающего алгоритм работы, но и учета времен задержек между логическими блоками, что увеличивает время разработки. К тому же, ASIC не является гибкой по отношению к применению в других областях, отличных от ее изначального назначения.

По этой причине прототипирование ASIC производится с помощью ПЛИС. Это позволяет реализовать необходимые функции и протестировать работу разных алгоритмов, редактировать их под нужды разных задач. Построение такого протипа экономит время и ресурсы, необходимые для разработки конечного варианта ASIC.

В данной работе алгоритм TDC реализуется с помощью ПЛИС для отладки его работы и тестирования разных вариантов алгоритмов. В дальнейшем планируется переход от ПЛИС к ASIC для увеличения скорости работы и построение промышленного прототипа компактного спектрометра ионизирующих излучений.

2 Архитектура спектрометра на основе TDC

2.1 Общая схема устройства спектрометра

На основе идеи перехода от измерения амплитуды к измерению ширины сигнала был разработан алгоритм измерения длительности поданного на вход ПЛИС сигнала. Он основан на подсчете количества тактовых сигналов, поданных на ПЛИС в то время, пока на одном из входных каналов ПЛИС есть напряжение (рис. 2.1). Это и есть алгоритм работы TDC.

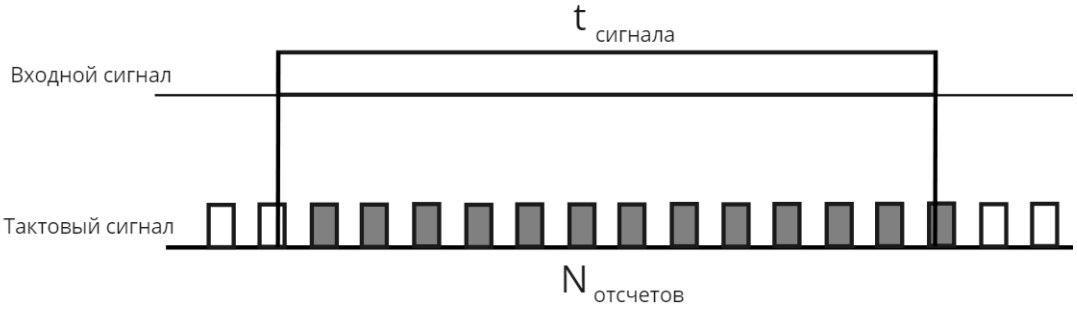

Рис. 2.1: Схема измерения длительности сигнала.

В отличие от классического спектрометрического тракта, где спектр - это распределение количества сигналов по их амплитуде или площади, спектр, измеряемый TDC-алгоритмом - это распределение количества сигналов по их длительности. В качестве ширины ширины канала спектра был выбран самый короткий интервал врмени, который возможно измерить с помощью описанного выше алгоритма - это длина тактового сигнала ПЛИС, которая составляет 20 нс. Для более сложной обработки спектра был разработан механизм взаимодействия ПЛИС и ПК. Он основан на интерфейсе RS-232 через реализацию в ПЛИС стандартного асинхронного последовательного приемо-передатчика типа

UART Universal Asynchronous Receiver/Transmitter). Общая схема работы ПЛИС представлена на рис. 2.2. Код программы размещен в приложении 1.

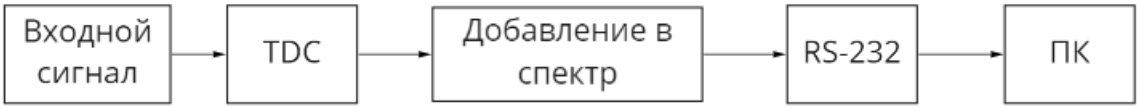

Рис. 2.2: Блок-схема сбора и обработки спектра внутри ПЛИС.

В качестве ПЛИС была выбрана отладочная плата Digilent Nexys2 основанная на ПЛИС Xilinx Spartan-3. Кроме ПЛИС на отладочной плате расположены порты для питания платы и программирования ПЛИС, а также большое количесвто входных и выходных соединений, кнопок, переключателей и индикаторов, что позволяет сделать сбор спектра более удобным.

Однако, данная плата может работать только с дискретными сигналами. Сигнал, поступающий с детектора, имеет резкий фронт и экспоненциальный спад (рис. 2.3), амплитуда которого постоянно непрерывно меняется, и подавать сразу на вход в ПЛИС его нельзя. Поэтому этот сигнал предварительно подается на вход другой платы, на которой расположен быстрый аналоговый компаратор с управляемым (в ручную) порогом. На этой плате этот сигнал усиливается до определенного уровня и проходит через компаратор, который на выходе генерирует сигнал в форме цифрового импульса, длина которого равна ширине входного сигнала на заданой поргом компаратора высоте. Затем этот сигнал идет на вход в ПЛИС. Чтобы исключить ненужные срабатывания компаратора, сигнал с ФЭУ подается на вход спектрометрического усилителя, который фильтрует и немного растягивает исходный аналоговый сигнал.

В ходе работы был собран макет компактного спектрометра (рис. 2.4). В качестве детектирующей среды был выбран кристалл NaI, т.к. он обладает высоким световыходом, хорошим энергетическим разрешением

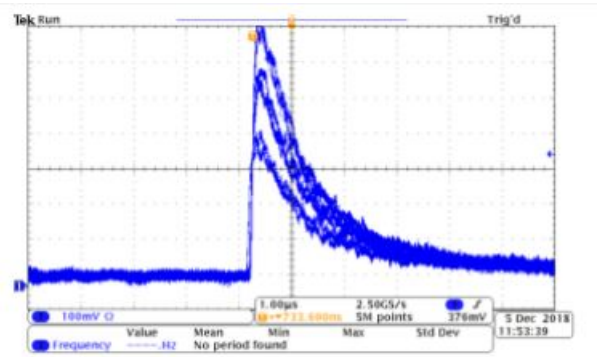

Рис. 2.3: Выходной сигнал ФЭУ.

и достаточно длиным временем высвечивания (250 нс), что снижает требования к быстродействию схемы TDC.

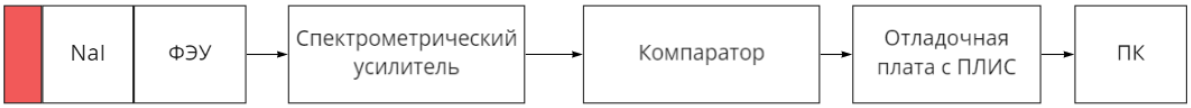

Рис. 2.4: Блок-схема макета спектрометра на основе TDC.

2.2 Алгоритм TDC и сбор спектра внутри ПЛИС

Работа TDC организована следующим образом. Сигнал с выхода компаратора попадает на вход в ПЛИС. Измерение длительности импульса осуществляется с помощью тактового сигнала ПЛИС (рис. 2.1). Для этого внутри ПЛИС организован счетчик. В исходном состоянии в нем выставлено значение 0. Как только на его вход начинает подаваться непрерывный сигнал, соответствующий логической единице, он начинает отсчитывать количество тактовых сигналов. Когда входной сигнал возвращается в положение логического нуля, счетчик передает полученное значение количества тактовых сигналов в модуль формирования спектра, а затем обнуляется.

Массив-спектр внутри ПЛИС представляет собой массив переменных 16-битной разрядности. Размер элементов массива можно менять в зависимости от загрузки детектора. Номер элемента массива соответствует

числу, измеренному с помощью счетчика тактовых сигналов. Когда счетчик отправляет в этот блок число, к соответствующему элементу массива прибавляется 1. Размер массива может меняться с помощью глобальной переменной, и на данный момент в нем содержится 200 элементов. Данное число соответствует числу дискретных уровней временного спектра, измеряемых с помощью TDC и определяет максимальную длину регистрируемого импульса - 4 мкс (20 нс X 200 = 4 мкс). Таким образом набирается распределение числа импульсов по их длительности.

Сейчас ширина канала спектрометра ограничена длиной тактового сигнала ПЛИС и составляет 20 нс. В дальнейшем возможно увеличение разрешения спектрометра путем уменьшения длины тактового сигнала, но для этого необходим заменить ПЛИС на более быстродействующую.

Для приема данных в компьютере через интерефейс RS-232 была написана программа на языке Python. Данная программа формирует массив-спектр уже в компьютере, заполняет его полученными с UART данными и записывает в специальный файл. Данный файл используется для дальнейшей обработки и визуализации спектра с помощью программного пакета Origin Lab.

3 Проверка работы алгоритмов сбора спектра

3.1 Подтверждение гипотезы о связи амплитуды и длительности сигнала

Для поиска функции, связывающей амплитуду и длительность сигнала, на вход спектрометрического усилителя подавался сигнал с генератора импульсов (рис. 3.1). Считаем стабильность амплитуды сигнала с генератора абсолютной. С его помощью создавался модельный спектр, в котором распределение сигналов по амплитуде было равномерным, а их форма была максимально приближена к форме выходного сигнала ФЭУ. Распределение набиралось до тех пор, пока счетчик не регистрировал 3000 событий. Порог компаратора подбирался таким образом, чтобы минимальный порог соответствовал минимальному порогу на уровне шумов ФЭУ и составил 30 мВ.

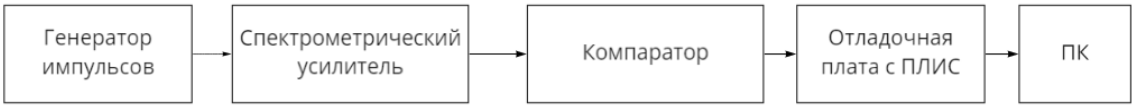

Рис. 3.1: Блок-схема макета для измерения калибровочного спектра.

Такая процедура повторялась 50 раз для сигналов с амплитудами в диапазоне от 30 до 500 мВ с шагом 5 мВ в диапазоне от 50 до 160 мВ, шагом 10 мВ в диапазоне от 160 до 400 мВ и с шагом 20 мВ в диапазоне от 400 до 500 мВ. Так как амплитудный спектр для одинаковых сигналов представляет собой единичный пик, его можно аппроксимировать функцией Гаусса, что было проделано для всех спектров (рис. 3.2) . С помощью такой аппроксимации были найдены точные номера каналов для вершин пиков.

После этого строилась зависимость амплитуды сигнала от номера

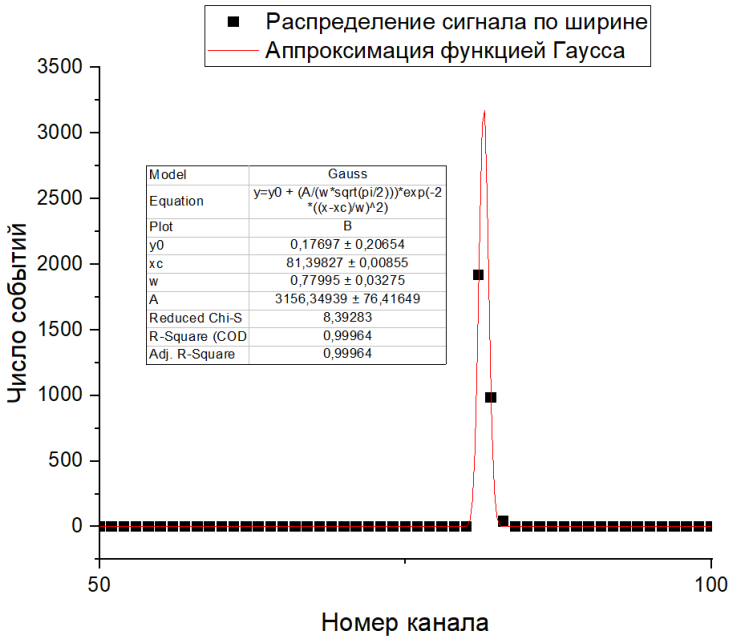

Рис. 3.2: Распределение сигналов по ширине при амплитуде сигнала 130 мВ.

канала (рис. 3.3), которая аппроксимировалась функцией ab^x. Был получен следующий результат:

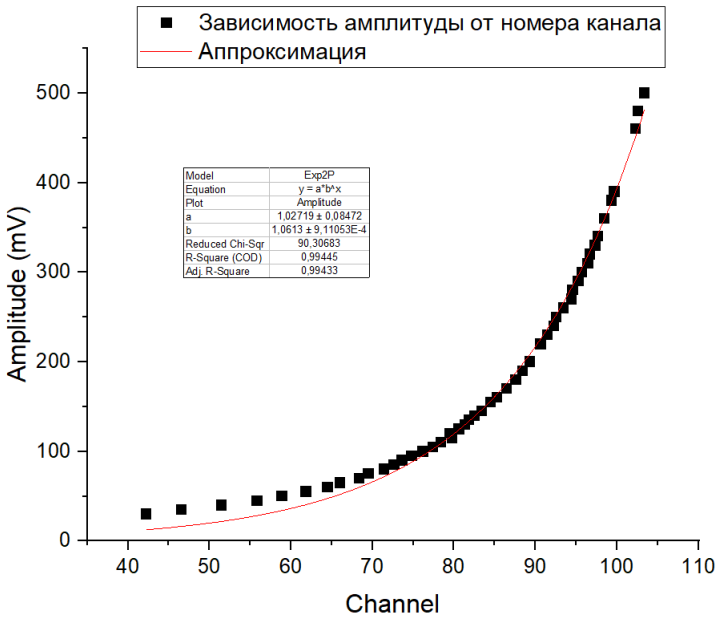

Рис. 3.3: Зависимость амплитуды сигнала от номера временного канала.

$$A = 1,03 * 1,061^T \tag{3.1}$$

где A - амплитуда сигнала в мВ, T - номер канала. Функция, связывающая длительность сигнала и его амплитуду, получилось гладкой, поэтому возможно однозначное восстановление амплитудного распределения из спектра TDC.

3.2 Проверка спектрометра с помощью измерения спектра радиоактивного исчтоника

Для проверки провильности методики TDC использовались два параллельных канала измерения спектра радиоактивного источника (рис. 3.4). Первый канал - это измерение спектра с помощью разработанных алгоритмов в ПЛИС. Второй - с помощью АЦП CAEN DT5780 (14 разрядов, 250 МГц частота дискретизации).

На вход в спектрометрического усилителя подавался сигнал с ФЭУ. В качестве источника был взят ^{60}Co. ^{60}Co - это радиоактивный изотоп, основным продуктом распада которого является γ-излучение. Взаимодействие γ-кванта в сцинтиллятор NaI(Tl) приводит к образованию стандартного по форме сигнала с резким фронтом и экспоненциальным спадом, амплитуда которого зависит от выделенной в сцинтилляторе энергии. Это позволяет применять полученную выше аппроксимацию (3.1) для восстановления амплитудного спектра γ-квантов.

Спектр набирался в течение 15 минут. Уровень компаратора подбирался таким образом, чтобы скорость счета с помощью TDC и АЦП была одинаковой. Полученное распределение сигналов по длительности представлено на рис. 3.5.

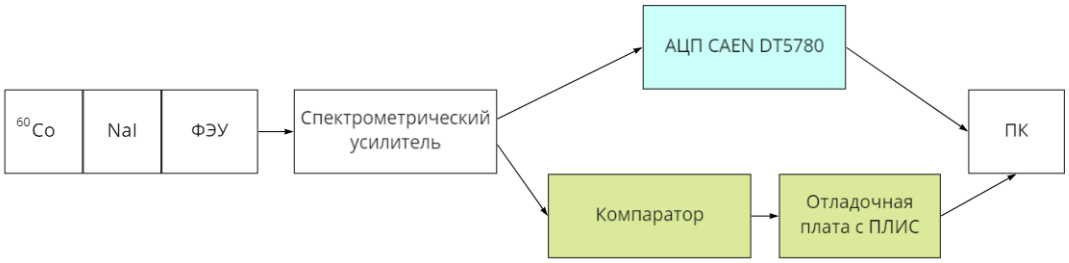

Рис. 3.4: Блок-схема макета для измерения спектра ^{60}Co.

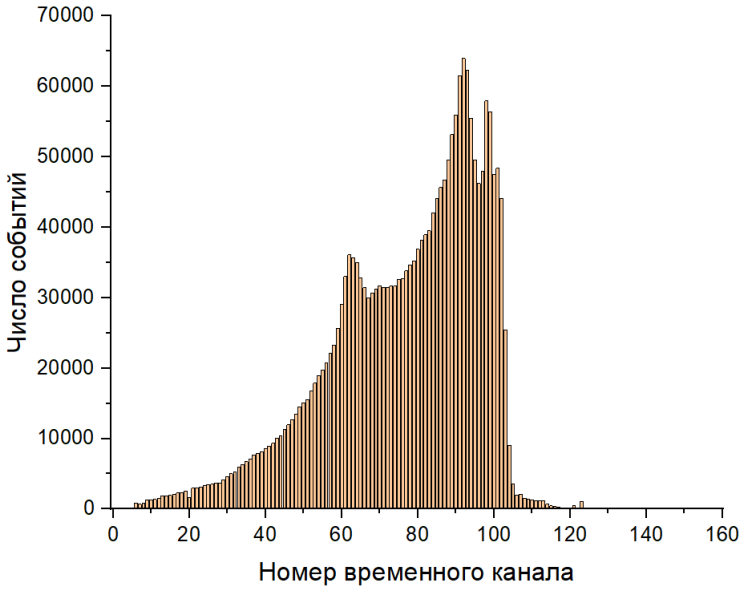

Рис. 3.5: Распределение сигналов по длительности, измеренное с помощью TDC.

Полученные с помощью TDC данные обрабатывались следующим образом. Временное распределение переводилось в амплитудное с помощью полученной ранее функции (3.1) следующим образом. Для границ каждого временного промежутка вычислялась соответствующая им амплитуда, таким образом строились границы для амплитудных промежутков. Полученный амплитудный спектр имеет вид, представленный на рис.3.6.

Затем полученное распределение сравнивалось с референсным спектром, измеренным с помощью АЦП (рис. 3.7). Видно, что ширина канала в полученном амплитудном распределении имеет неравномерный характер, она увеличивается с ростом амплитуды. Это приводит к снижению разрешающей способности при увеличении амплитуды сигнала и усложняет обработку спектра.

Пики 1 и 2 в референсном амплитудном спектре соответствуют пикам полного поглощения, поэтому удобно оценивать правильность работы метода TDC, используя именно их. Чтобы убедиться, что пики в референсном спектре и в распределении, полученном с помощью TDC, соответствуют друг другу, было вычислено количество

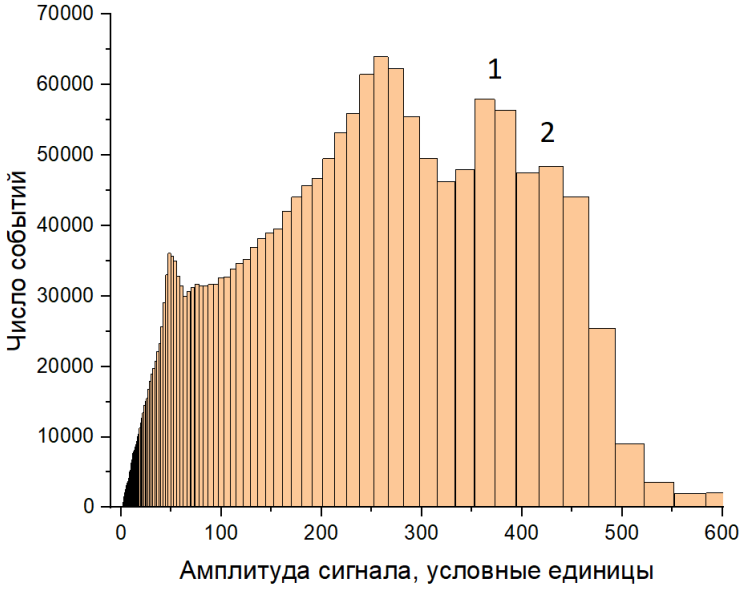

Рис. 3.6: Восстановленный спектр ^{60}Co, измеренный с помощью TDC (Распределение сигналов по амплитуде). Пики 1 и 2 соответствуют пикам полного поглощения ^{60}Co с энергиями 1173 и 1333 кэВ.

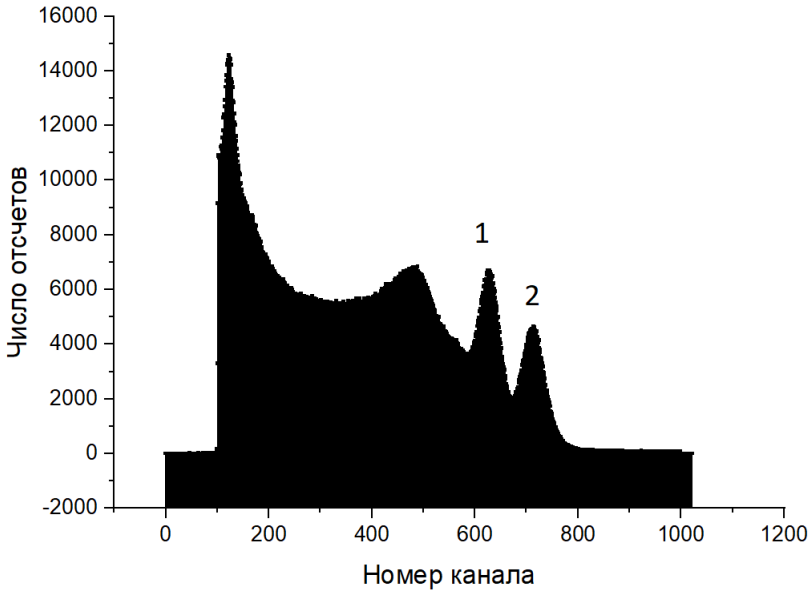

Рис. 3.7: Спектр ^{60}Co, измеренный с помощью АЦП CAEN DT5780. Пики 1 и 2 соответствуют пикам полного поглощения ^{60}Co с энергиями 1173 и 1333 кэВ

зарегистрированных событий в каждом пике. Результат представлен в таблице 3.1.

Получается, что число зарегистрированных событий в пиках

Таблица 3.1: Сравнение числа зарегистрированных событий в пиках с помощью АЦП и TDC

Пик	Число событий в референсном спектре	Число событий TDC
1	227422	255965
2	165239	174565

одинаково с некоторой погрешностью для спектров, измеренных с помощью АЦП и TDC. Это позволяет сделать вывод о том, что распределение по длительности сигнала получено корректно и алгоритм TDC работает правильно.

Далее проводилась оценка разрешающей способности спектрометра на основе TDC. Для этого оценивалось, каким образом разрешающая способность по амплитуде зависит от амплитуды. Из зависимости (3.1) имеем:

$$A = 1,03 * 1,061^T = 1,03e^{(ln1,061)T} = 1,03e^{0,059T} = f(T); \qquad (3.2)$$

Для приращения ΔA можно записать:

$$\Delta A = f'(T)\Delta T = \frac{1,03e^{0,059T}}{0,059}\Delta T = \frac{A}{0,059}\Delta T; \qquad (3.3)$$

Так как ΔT постоянно и равно 20 нс, из 3.3 видно, что с ростом A растет ширина амплитудного канала. Это ведет за собой уменьшение разрешающей способности с ростом амплитуды. Это хорошо видно на рис. 3.6.

Так как пики 1 и 2 являются пиками полного поглощения ^{60}Co, с их помощью можно численно вычислить разрешающую способность TDC и сравнить ее с разрешающей способностью АЦП. Для этого проводилась аппроксимация пиков 1 и 2 функциями Гаусса для спектра, измеренного с помощью TDC, с помощью программного пакета Origin Lab.

Чтобы правильно сравнить разрешающую способность спектрометра на основе TDC с классическим, необходимо, чтобы в обоих спектрах было одинаковое число дискретных уровней. Для этого для спектра, измеренного

с помощью АЦП, было уменьшено количесвто каналов с 1024 до 200. Это осуществлялось путем сложения числа событий в соседних каналах, и полученное число сопоставлялось более широкому каналу. После этого процедура аппроксимации пиков функцией Гаусса была произведена и для классического АЦП.

Результаты показаны на рис. 3.8 и 3.9.

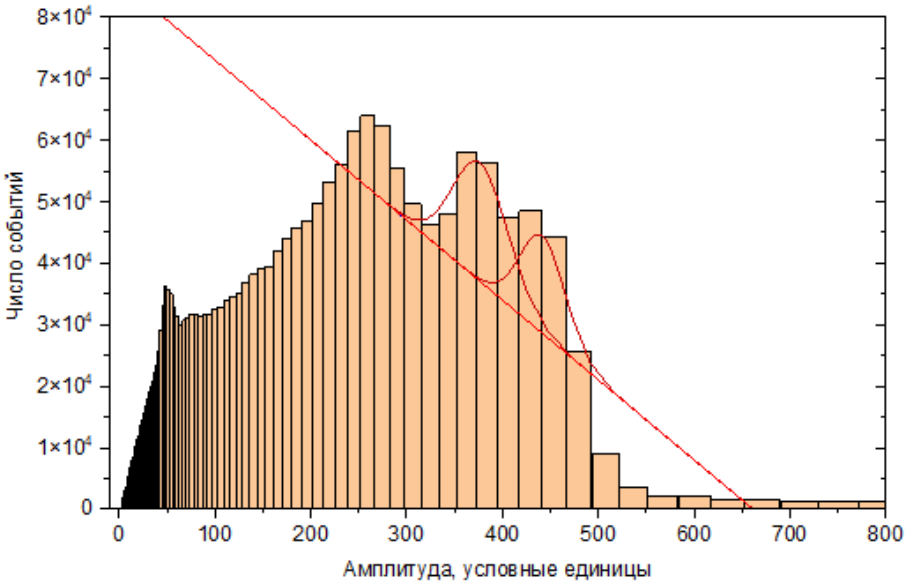

Рис. 3.8: Аппроксимация пиков 1 и 2 функциями Гаусса для распределения, измеренного с помощью TDC

С помощью проведенной аппроксимации были найдены ширины пиков на их полувысоте, и после этого была вычислена разрешающая способность по формуле:

$$R_A = \frac{\Delta A}{A} * 100\%.$$

где R_A - разрешающая способность, выраженная в процентах, ΔA - ширина пика на полувысоте, A - координата вершины пика. Полученные результаты представлены в таблице 3.2.

Получается, что разрешающая способность в амплитудном распределении TDC в области пиков 1 и 2 в 3 раза хуже, чем у АЦП. Но так как счет событий у АЦП и TDC сходится с некоторой погрешностью (таблица 3.1), предложенную архитектуру спетрометра можно использовать в дозиметрии.

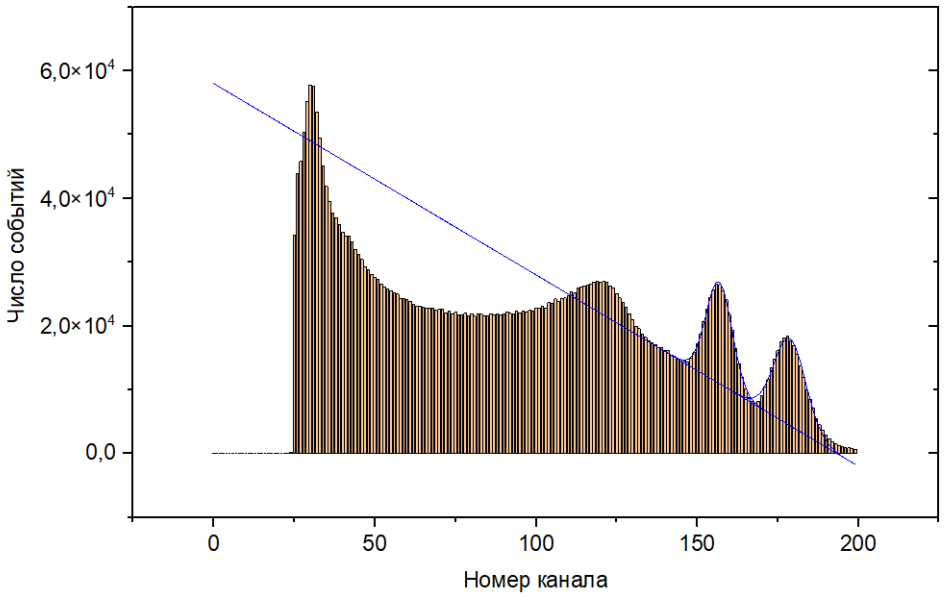

Рис. 3.9: Аппроксимация пиков 1 и 2 функциями Гаусса для распределения, измеренного с помощью АЦП

Таблица 3.2: Сравнение разрешающей сспособности для спектрометров на основе АЦП и TDC

Пик	Ширина пика для спектра на основе АЦП	Ширина пика для спектра на основе TDC	Разрешающая способность спектрометра на основе АЦП	Разрешающая способность спектрометра на основе TDC
1	9,47 каналов	67,96 условных единиц	6%	18%
2	11,55 каналов	55,85 условных единиц	6,4%	12,6%

При рассчете эквивалентной дозы необходимо учитывать энергию ионизирующего излучения [10]. В большинстве случаев это делается с помощью модельных спектров, а сам дозиметр работает только в счетном режиме. Однако, оценивая с помощью TDC энергию регистрируемых частиц, можно повышать точность измерения эквивалентной дозы. Также малые размеры предлагаемой архитектуры позволяют разработать носимый дозиметр для личного использования.

4 Схема совпадений

Так как конструкцию спектрометра можно уменьшать не только за счет уменьшения электронных компонентов, но и за счет уменьшения размеров детектирующей аппаратуры, возможен переход от архитектуры спектрометра, использующей ФЭУ, к более компактной. Для этого можно заменить ФЭУ на 2 кремниевых фотоумножителя (Silicon Photomultiplier, SiPM) [11].

SiPM представляет собой матрицу лавинных диодов, на которую подается напряжение, близкое к напряжению пробоя (напряжение смещения). Фотоны сцинтилляции, попадающие на SiPM, вызывают лавинную ионизацию, сигнал которой в дальнейшем обрабатывается. Такой режим работы приводит к высокой вероятности спонатнного срабатывания SiPM в отсутствие фотонов сцинтилляции (темновые шумы SiPM). Поэтому для снижения темновых шумов можно использовать два SiPM, регистрирующих фотоны сцинтилляции одновременно с одного сцинтилляционного кристалла. Подавая их выходные сигналы на вход схемы совпадений, можно почти полностью исключить темновые шумы SiPM. Для реализации этого механизма в работу архитектуру компактного спектрометра на основе ПЛИС была встроена схема совпадений.

Работа схемы совпадений организована следующим образом. На два входа в ПЛИС поступает сигнал с разных детекторов, регистрирующих излучение в одном и том же кристалле сцинтиллятора. Также на отдельный вход в ПЛИС через интерфейс RS-232 подается ширина окна совпадений, которая выбирается в зависимости от длительности регистрируемых импульсов сцинтилляции.

Сигналы с каждого из SiPM (s_1 и s_2, см. рис. 4.1) подаются на отдельные счетчики. Параллельно эти сигналы подаются на соответствующие формирователи окна. Этот модуль, при появлении на

его входе сигнала, на выходе генерирует сигнал заданной длины, равной ширине окна совпадений (рис. 4.2). Затем длинный сигнал с одного канала и короткий сигнал с другого логически умножаются. Получаемые таким образом два сигнала логически складываются. Такая схема позволяет искать совпадения относительно обоих каналов одновременно.

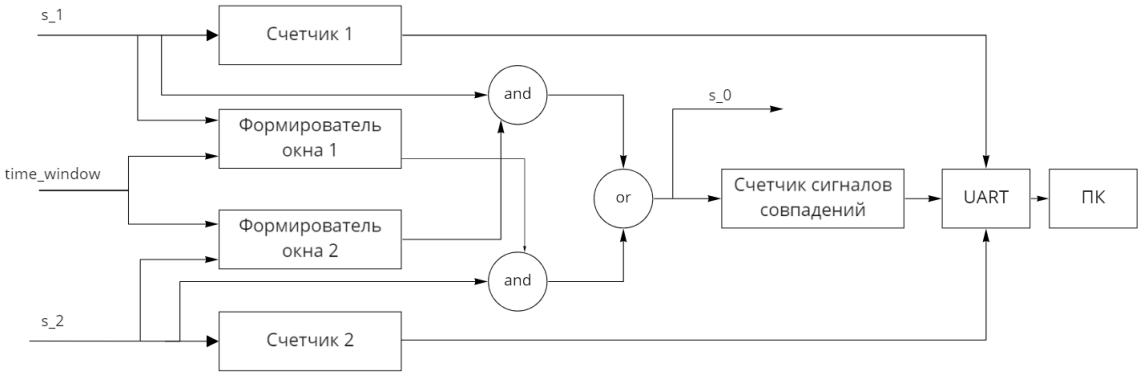

Рис. 4.1: Блок-схема работы схемы совпадений.

Полученный сигнал s_0 поступает на вход отдельного счетчика сигналов совпадений (рис. 4.1). Также этот сигнал подается на выход из ПЛИС. Он может использоваться в качестве триггера для внешнего АЦП для работы с внешним классическим спектрометрическим трактом. Значения всех трех счетчиков каждые 10 секунд передаются с помощью интерфейса RS-232 в компьютер.

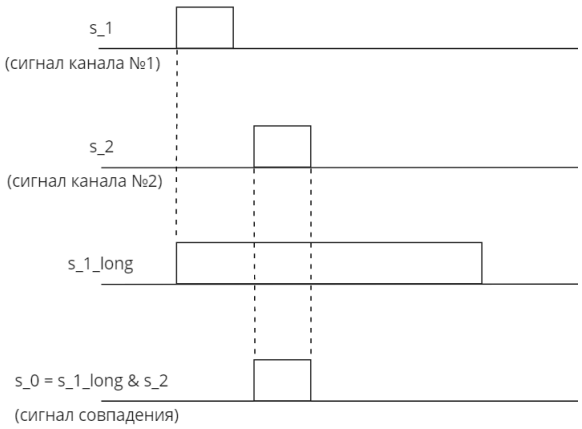

Рис. 4.2: Алгоритм работы схемы совпадений.

Счетчик 1 и Счетчик 2 (рис. 4.1), встроенные в схему совпадений, позволяют контролировать уровни темновго шума каждого из SiPM, таким образом контролируя правильность выбора напряжения смещения. Счетчик сигналов совпадений показывает число сцинтилляционных фотонов, регистрируемых двумя SiPM в рамках окна совпадений. Таким образом, его значение показывает количество зарегистрированных в сцинтилляторе частиц, и мы получаем схему простейшего дозиметра.

Реализация описанных выше алгоритмов TDC и схемы совпадений в одной ПЛИС с использованием SiPM для регистрации сцинтилляции позволит создать спектрометр с еще более компактными размерами и малым энергопотреблением.

ЗАКЛЮЧЕНИЕ

В результате работы была предложена архитектура построения спектрометра-дозиметра на основе методов, позволяющих перейти от измерения амплитуды сигналов к измерению их длительности. Был разработан код для ПЛИС, реализующий алгоритм TDC и схему совпадений. Для проверки кода был собран макет спектрометра, с помощью которого был измерен спектр радиоактивного источника ^{60}Co. Для обработки данных, поступающих с макета спектрометра в компьютер, была также написана программа. Обработка полученных спектров производилась с помощью программного пакета Origin Lab.

На данный момент точность измерения длительности сигнала составляет 20 нс. Разрешающая способность по амплитуде падает с ростом амплитуды. Счет событий в каналах спектрометра на основе TDC выполняется корректно, что позволяет приблизительно оценить энергию регистрируемых частиц. Другие преимущества данной архитектуры - это малые размеры и энергопотребление по сравнению с классическими спектрометрическими трактами.

Эти свойства позволяют использовать предложенную архитектуру спектрометра в качестве компактного спектрометра-дозиметра, который, по сравнению с обычными дозиметрами, позволяет точнее измерять эквивалентную дозу излучения за счет дополнительной оценки энергии излучения. Точная оценка эквивалентной дозы крайне важна для контроля безопасности персонала на радиационно-опасных объектах.

В программу ПЛИС встроена схема совпадений, что позволит перейти к конструкции, использующей два SiPM «на совпедение» вместо ФЭУ. Это позволит уменьшить размеры и энергопотребление спектромера.

СПИСОК ЛИТЕРАТУРЫ

1. АЦП. Учебное пособие к выполнению лабораторной работы по дисциплине "Микропроцессорная техника"и курсового проекта по дисциплине "Электроника и микроэлектроника"для студентов ФТФ специальности 200600. - Томский политехнический университет. - 2004

2. Аналого-цифровые и цифроаналоговые преобразователи. Методические указания к выполнению лабораторной работы по дисциплине "Электротехника, электроника и схемотехника"для студентов специальности 090301. - Юго-Западный государственный университет. - 2016.

3. Xishan Sun et al. Energy and Timing Measurement with Time-Based Detector Readout for PET Applications: Principle and Validation with Discrete Circuit Components // Nucl Instrum Methods Phys Res A. 2011 June 11; 641(1): 128–135.

4. William Lemaire et al. Embedded time of arrival estimation for digital silicon photomultipliers with in-pixel TDCs // Nuclear Inst. and Methods in Physics Research, A 959 (2020) 163538

5. M. Patka et al. Multichannel FPGA based MVT system for high precision time (20 ps RMS) and charge measurement // arXiv:1707.03565v1 [physics.ins-det] 12 Jul 2017

6. Crispin Williams et al. Particle identification using time of flight // Journal of Physics G: Nuclear and Particle Physiscs. 2012. vol. 39.

7. Yu Liang et al. Design and performance of a TDC ASIC for the upgrade of the ATLAS Monitored Drift Tube detector // Nuclear Instruments and Methods in Physics Research A, Volume 939, 21 September 2019, p 10-15

8. Fan Huan-Huan et al. TOT Measurement Implemented in FPGA TDC //

9. F. Gonnella et al. Time over threshold in the presence of noise // arXiv:1412.1743v1 [physics.ins-det] 4 Dec 2014

10. Нормы радиационной безопасности НРБ-99/2009: СанПиН 2.6.1.2523-09; утв. постановлением Главного государственного санитарного врача Российской Федерации от 7 июля 2009 года N 47

11. Кондрашов А. А. Компактный детектор тяжелых заряженных частиц на основе сцинтиллятора ZnSe // Бакалаврская работа, Московский Государственный Университет имени М. В, Ломоносова, 2020

ПРИЛОЖЕНИЕ 1. ПРОГРАММА, НАПИСАННАЯ НА ЯЗЫКЕ VHDL, РЕАЛИЗУЮЩАЯ TDC И СХЕМУ СОВПАДЕНИЙ НА ПЛИС

Модуль-счетчик входных сигналов.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity counter is
  generic(
    byte_size : integer := 24
  );
  Port (
    sin  : in STD_LOGIC;

    sout : out std_logic_vector(byte_size - 1 downto 0);

    inf  : in std_logic;
    bl   : in std_logic;

    clk  : in std_logic;
    rst  : in std_logic
  );
end counter;
```

```
architecture Behavioral of counter is

  signal cnt    : std_logic_vector(byte_size - 1 downto 0)
    ;

begin

  sout <= cnt;

  sync : process (clk)
  begin

  if rising_edge(clk) then
    if (rst = '1') or (inf = '1') then
      cnt <= (others => '0');
    else
      if (sin = '1') and (bl = '0') then
        cnt <= cnt + 1;
      end if;
    end if;

  end if;
  end process;

end Behavioral;
```

Модуль, генерирующий единичный сигнал при наличии на его входе сигнала. Используется для корректного подсчета сигналов.

```
entity my_signal is
    Port (
    sin : in STD_LOGIC;
    sout : out std_logic;
```

```
    clk : in std_logic;
    rst : in std_logic
    );
end my_signal;

architecture Behavioral of my_signal is

signal d : std_logic;
signal s : std_logic_vector (1 downto 0);

begin
  sout <= d;

  sync : process(clk)
  begin

    if rising_edge(clk) then
      if rst = '1' then
        s <= (others => '0');
        d <= '0';
      else
        s(1) <= sin;
        s(0) <= s(1);

        d <= '0';

        if s(0) = '0' and s(1) = '1' then
          d <= '1';

        end if;

      end if;
```

```
    end if;
  end process;

end Behavioral;
```

Модуль-формирователь сигнала, ддина которого равна длине временного окна, внутри которого ищется совпадение между двумя сигналами.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity buff is
    Port (
    sin : in STD_LOGIC;
    sout : out std_logic;
    fin  : in std_logic;
    fout : out std_logic;

    time_window : in std_logic_vector (7 downto 0);

    clk : in std_logic;
    rst : in std_logic
    );
end buff;

architecture Behavioral of buff is

signal clk_window : integer;
```

```
  type astate is (idle, count);

  type reg_type is record
    state : astate;
    i     : integer;
    s_buff : std_logic;
    f_buff : std_logic;

  end record;

  signal r, rin : reg_type;

begin

  sout <= r.s_buff;
  fout <= r.f_buff;

  clk_window <= to_integer(unsigned(time_window));

  async_1 : process(r, sin, fin, clk_window)
  variable v : reg_type;
  begin

    v := r;

    case v.state is

      when idle =>


        if (sin = '1') and (fin = '0') then
          v.state := count;
          v.f_buff := '1';
```

```
        v.s_buff := '1';
        v.i := 0;
      else
        v.f_buff := '0';
        v.s_buff := '0';
      end if;


    when count =>
      if v.i < clk_window then
        v.s_buff := '1';
        v.f_buff := '1';
        v.i := v.i + 1;

      else
        v.i := 0;
        v.f_buff := '0';
        v.s_buff := '0';
        v.state  := idle;
      end if;
  end case;

  rin <= v;
end process;

sync : process (clk)
begin
  if rising_edge(clk) then
    if rst = '1' then
      r.i <= 0;
      r.s_buff <= '0';
      r.state  <= idle;
      r.f_buff <= '0';
```

```
      else
        r <= rin;
      end if;
    end if;
  end process;

end Behavioral;
```

Модуль, реализующий измерение длительности входного сигнала.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity tdc_long is
    generic(
    constant wid : integer := 50000;
    constant disc : integer := 7
  );
    Port (
    s : in STD_LOGIC;
    s_con : in std_logic;
    wide : out std_logic_vector (disc downto 0);
    bl : out std_logic;
    time_window : in std_logic_vector(7 downto 0);
    valid : out std_logic;

    trg   : in std_logic;

    clk : in std_logic;
    rst : in std_logic
```

```
  );
end tdc_long;

architecture Behavioral of tdc_long is

type state is (idle, mistake, count);
type mtype is record
  state : state;
  valid : std_logic;
  m     : integer;
  t     : std_logic_vector (disc downto 0);
end record;

signal r, rin : mtype;

begin

  wide <= r.t;
  valid <= r.valid;

  async : process (s, r)
  variable v : mtype;
  begin

    v := r;

    case v.state is
      when idle =>
        v.m := 0;
        v.t := (others => '0');
        v.valid := '0';
        if (s = '1') and (s_con = '1') then
          v.state := mistake;
```

```
      end if;

    when mistake =>

      v.m := v.m + 1;
      v.t := v.t + 1;

      if (v.m > 5) then
        if (s = '1') then
          v.state := count;
        else
          v.state := idle;
        end if;
      end if;

    when count =>
      if (s = '1') then
        v.t := v.t + 1;
      else
        v.valid := '1';
        v.state := idle;
      end if;
  end case;
  rin <= v;
end process;

sync : process (clk)
begin

  if rising_edge(clk) then
    if rst = '1' then
      r.state <= idle;
      r.t <= (others => '0');
```

```
        r.m <= 0;
        r.valid <= '0';
      else
        r <= rin;
      end if;
    end if;
  end process;

end Behavioral;
```

Модуль, реализующий схему совпадений. Он нужен для корректного разведения сигналов на входы в вышеупомянутое модули.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

use IEEE.NUMERIC_STD.ALL;

entity coinsidence is

  generic(

    constant freq         : integer := 100;
    constant cnt_size     : integer := 24;
    constant disc : integer := 7

  );
    Port (
    s1 : in STD_LOGIC;
    s2 : in std_logic;
    s3 : in std_logic;
```

```
s0 : out std_logic;

s1_cnt : out std_logic_vector(cnt_size - 1 downto 0);
s2_cnt : out std_logic_vector(cnt_size - 1 downto 0);
s0_cnt : out std_logic_vector(cnt_size - 1 downto 0);

tx_o   : out std_logic;

time_window : in std_logic_vector (7 downto 0);
inf : in std_logic;

clk : in std_logic;
rst : in std_logic;

tdc1 : out std_logic_vector(disc downto 0);
tdc2 : out std_logic_vector(disc downto 0);

valid1 : out std_logic;
valid2 : out std_logic;

valid_o : out std_logic;

zan1 : out std_logic;
zan2 : out std_logic;

s1_long_out : out std_logic;
s2_long_out : out std_logic;
f1_out : out std_logic;
f2_out : out std_logic;
s1_m_out : out std_logic;
s2_m_out : out std_logic

);
```

```
end coinsidence;

architecture Behavioral of coinsidence is

component my_signal is
    Port (
    sin : in STD_LOGIC;
    sout : out std_logic;

    clk : in std_logic;
    rst : in std_logic
    );
end component;

component tdc_long is

  generic(
    constant wid : integer := 50000;
    constant disc : integer := 7
  );
    Port (
    s : in STD_LOGIC;
    s_con : in std_logic;
    wide : out std_logic_vector (disc downto 0);
    bl : out std_logic;
    time_window : in std_logic_vector (7 downto 0);
    valid : out std_logic;
    trg   : in std_logic;

    clk : in std_logic;
    rst : in std_logic
  );
```

```
end component;

component counter is
  generic(
    byte_size : integer := 24
  );
  Port (
    sin  : in STD_LOGIC;
    sout : out std_logic_vector(byte_size - 1 downto 0);

    inf  : in std_logic;
    bl   : in std_logic;

    clk  : in std_logic;
    rst  : in std_logic
  );
end component;

component buff is
    Port (
    sin : in STD_LOGIC;
    sout : out std_logic;
    fin  : in std_logic;
    fout : out std_logic;

    time_window : in std_logic_vector (7 downto 0);

    clk : in std_logic;
    rst : in std_logic
    );
end component;

signal s1_m : std_logic;
```

```
signal s2_m : std_logic;
signal f1, f2 : std_logic;
signal s0_m : std_logic;
signal s1_long, s2_long : std_logic;
signal bl1 : std_logic;
signal bl2 : std_logic;
signal bl3 : std_logic;
signal bl0 : std_logic;

type state is (idle, buf, end1, end2, end3, end4, end5,
   end6, waiting1, waiting2, waiting3, waiting4, waiting5
   , waiting6, send1, send2, send3, send4, waiting7,
   waiting8, waiting9, waiting10);
type regtype is record
  tdc_buf : std_logic_vector(15 downto 0);
  state   : state;
  s       : std_logic_vector(7 downto 0);
  t       : integer;
  ch      : integer;
  valid   : std_logic;
  valid_o : std_logic;
  zan     : std_logic;
end record;

signal r1, r2, rin1, rin2 : regtype;

signal busy_buf : std_logic;

signal valid1_buf : std_logic;
signal valid2_buf : std_logic;
signal valid3_buf : std_logic;

signal tdc1_buf : std_logic_vector(disc downto 0);
```

```
signal tdc2_buf : std_logic_vector(disc downto 0);
signal tdc3_buf : std_logic_vector(disc downto 0);

signal valid_tdc : std_logic;
signal s_tdc : std_logic_vector(7 downto 0);

signal ch : integer;

begin

  valid1 <= valid1_buf;
  valid2 <= valid2_buf;

  valid_o <= r1.valid_o or r2.valid_o;

  tdc1 <= tdc1_buf + tdc2_buf + tdc3_buf;
  tdc2 <= tdc2_buf;

  s0 <= s0_m;

  s1_long_out <= s1_long;
  s2_long_out <= s2_long;
  s1_m_out <= s1_m;
  s2_m_out <= s2_m;
  f1_out <= f1;
  f2_out <= f2;

  zan1 <= r1.zan;
  zan2 <= r2.zan;


  valid_tdc <= r1.valid or r2.valid;
```

```
port_map_1 : my_signal
port map (
  sin => s1,
  sout => s1_m,
  clk => clk,
  rst => rst
);

port_map_2 : my_signal
port map (
  sin => s2,
  sout => s2_m,
  clk => clk,
  rst => rst
);

port_map_11 : tdc_long
generic map (
  wid => 50000
)
port map(
  s => s1,
  s_con => '1',
  wide => tdc1_buf,
  bl => bl1,
  clk => clk,
  rst => rst,
  time_window => time_window,
  valid => valid1_buf,
  trg => s0_m
);
```

```
port_map_12 : tdc_long
generic map (
  wid => 50000
)
port map(
  s => s2,
  s_con => s1,
  wide => tdc2_buf,
  bl => bl2,
  clk => clk,
  rst => rst,
  time_window => time_window,
  valid => valid2_buf,
  trg => s0_m
);

port_map_13 : tdc_long
generic map (
  wid => 50000
)
port map(
  s => s3,
  s_con => s2,
  wide => tdc3_buf,
  bl => bl3,
  clk => clk,
  rst => rst,
  time_window => time_window,
  valid => valid3_buf,
  trg => s0_m
);
```

```
port_map_3 : counter
generic map(
  byte_size => 24
)
port map (
  sin => s1_m,
  sout => s1_cnt,
  clk => clk,
  rst => rst,
  inf => inf,
  bl => bl1
);

port_map_4 : counter
generic map(
  byte_size => 24
)
port map (
  sin => s2_m,
  sout => s2_cnt,
  clk => clk,
  rst => rst,
  inf => inf,
  bl => bl2
);

port_map_5 : buff
port map (
  sin => s1_m,
  sout => s1_long,
  fin => f2,
  fout => f1,
  time_window => time_window,
```

```
    clk => clk,
    rst => rst
);

port_map_6 : buff
port map (
    sin => s2_m,
    sout => s2_long,
    fin => f1,
    fout => f2,
    time_window => time_window,
    clk => clk,
    rst => rst
);

bl0 <= bl1 or bl2;
-- s0_m <= ((s1_m and s2_long) or (s2_m and s1_long) or
    (s1_m and s2_m)) and (not bl0);
s0_m <= '1';

port_map_7 : counter
generic map(
    byte_size => 24
)
port map (
    sin => s0_m,
    sout => s0_cnt,
    clk => clk,
    rst => rst,
    inf => inf,
    bl => '0'
);
```

```
end Behavioral;
```

Модуль верхнего уровня. Он реализует общение программы с компьютером. Здесь же происходит сбор спектра.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity connection is

  generic(

    constant cnt : integer := 24;
    constant d   : integer := 1;
    constant n   : integer := 300;
    constant disc : integer := 7

  );

    Port (
    dat_i : in STD_LOGIC;
    dat_o : out std_logic;

    zan1  : out std_logic;
    zan2  : out std_logic;

    s1    : in std_logic;
    s2    : in std_logic;
    s3    : in std_logic;
```

```
    clk    : in std_logic;

    inn_out : out std_logic;

    S0     : OUT std_logic;
    sign_1 : out std_logic;
    sign_2 : out std_logic;

    s1_long_out : out std_logic;
    s2_long_out : out std_logic;
    f1_out : out std_logic;
    f2_out : out std_logic;
    s1_m_out : out std_logic;
    s2_m_out : out std_logic;

    clk_out : out std_logic;

    btn : in std_logic;

    rst : in std_logic


    );
end connection;

architecture Behavioral of connection is

component coinsidence is

  generic(

    constant freq         : integer := 100;
```

```
  constant cnt_size    : integer := 24;
  constant disc : integer := 7

);

  Port (
  s1 : in STD_LOGIC;
      s2 : in STD_LOGIC;
      s3 : in std_logic;
      s0 : out STD_LOGIC;
      clk : in std_logic;
      rst : in std_logic;

      inf : in std_logic;

      tx_o : out std_logic;

      time_window : in std_logic_vector (7 downto 0);

      tdc1 : out std_logic_vector(disc downto 0);
  tdc2 : out std_logic_vector(disc downto 0);
  valid1 : out std_logic;
  valid2 : out std_logic;

  valid_o : out std_logic;
  zan1 : out std_logic;
  zan2 : out std_logic;



      s1_cnt : out std_logic_vector (cnt_size - 1
        downto 0);
      s2_cnt : out std_logic_vector (cnt_size - 1
        downto 0);
```

```
        s0_cnt : out std_logic_vector (cnt_size - 1
          downto 0);
        s1_long_out : out std_logic;
    s2_long_out : out std_logic;
    f1_out : out std_logic;
    f2_out : out std_logic;
    s1_m_out : out std_logic;
    s2_m_out : out std_logic
        );
end component;

component uart_rx is

  generic(
    frequency : integer := 100;
    baudrate : integer := 19200);


    Port ( d_i   : in STD_LOGIC;
           rst   : in STD_LOGIC;
           clk   : in STD_LOGIC;
           valid : out std_logic;
           d_o   : out STD_LOGIC_VECTOR (7 downto 0));

end component;

component uart_tx is

  generic (
    FREQUENCY : integer := 50; -- Ref. clock frequency in
       mHz
    BAUDRATE  : integer := 19200; -- Target baudrate
    BYTE_SIZE : integer := 8 -- Data packet size
```

```
    );

    Port ( Clk_i      : in  STD_LOGIC;
           Rst_i      : in  STD_LOGIC;
           Tx_o       : out STD_LOGIC;
           Tx_busy_o  : out STD_LOGIC;
           Tx_load_i  : in  STD_LOGIC;
           Tx_data_i  : in  STD_LOGIC_vector(BYTE_SIZE - 1
               downto 0);
           led : out std_logic;
           led2 : out std_logic;
           led3 : out std_logic);
end component;

component buff is
    Port (
    sin : in STD_LOGIC;
    sout : out std_logic;

    time_window : in std_logic_vector (7 downto 0);

    clk : in std_logic;
    rst : in std_logic
    );
end component;

component spi_slave
    port (
        clk      : in  std_logic;
        SCK      : in  std_logic;
        MOSI     : in  std_logic;
        SSEL     : in  std_logic;
        D_TX     : in  std_logic_vector(31 downto 0);
```

```
        MISO     : out std_logic;
        ALE      : out std_logic;
        DLE      : out std_logic;
        ADR_RX   : out std_logic_vector(7 downto 0);
        D_RX     : out std_logic_vector(31 downto 0)
    );
end component;

signal time_window_buff : std_logic_vector(7 downto 0) ;
signal t : std_logic_vector(7 downto 0) := "00000110";

type spectrum is array (0 to n) of std_logic_vector (15
   downto 0);
--signal sp_0 : spectrum := (others => (others => '0'));
--signal t_i  : integer := 0;
type astate is (idle, end1, wait1, send11, send12, send13
   , send21, send22, send23, send01, send02, send03,
   send_n,
        wait11, wait12, wait13, wait21, wait22, wait23,
           wait01, wait02, wait03, wait0);
constant info_window : integer := 500000000;
type regtype is record
  state : astate;
  s     : std_logic_vector (7 downto 0);
  valid : std_logic;
  k     : integer range 0 to info_window + 1;
  t     : integer range 0 to 101;
  inf   : std_logic;
  close : std_logic;
  s1_cnt_r : std_logic_vector (cnt - 1 downto 0);
  s2_cnt_r : std_logic_vector (cnt - 1 downto 0);
  s0_cnt_r : std_logic_vector (cnt - 1 downto 0);
  busy_o : std_logic;
```

```
end record;

type send_spectrum is (idle, send_sp_0, send_sp_1,
   send_sp_2, send_sp_3, send_sp_4, send_sp_5, send_sp_6,
    send_sp_7, send_sp_8, send_sp_9, send_sp_n,
              wait_sp_0, wait_sp_1, wait_sp_2, wait_sp_3,
                 wait_sp_4, wait_sp_5, wait_sp_6,
                 wait_sp_7, wait_sp_8, wait_sp_9,
                 wait_sp_n, waiting_buf);

type send_sp_reg is record
  state : send_spectrum;
  s : std_logic_vector (7 downto 0);
  valid : std_logic;
  i : integer range 0 to n + 1;
  t : integer range 0 to 13;
  busy_o : std_logic;
  sp : spectrum;
end record;

signal g, gin : send_sp_reg;

signal busy_buf      : std_logic;

type sp_state is (idle, search);
type get_sp is record
  state : sp_state;
  i : integer range 0 to n+1;
  sp : spectrum;
  tdc_int : integer;
end record;

signal sp_r, sp_rin : get_sp;
```

```
signal r, rin : regtype;

signal z : integer;
signal inn : std_logic;

type arraym is array(2 downto 0) of std_logic_vector (31
  downto 0);

signal s0_buff : std_logic;
signal s1_cnt_buff : std_logic_vector (cnt - 1 downto 0);
signal s2_cnt_buff : std_logic_vector (cnt - 1 downto 0);
signal s0_cnt_buff : std_logic_vector (cnt - 1 downto 0);

signal valid_buff  : std_logic;

signal rst_buf : std_logic;
signal rst_clk : integer := 0;

signal sign_1_buf : std_logic;
signal sign_2_buf : std_logic;
signal sign_3_buf : std_logic;

signal clk_i : integer := 0;
signal clk_sign : std_logic;

signal d_i_b : std_logic;

signal massiv : arraym;
signal spi_adr_rx_w : std_logic_vector(7 downto 0);
signal spi_adr : integer;
signal mas_i   : integer range 0 to 2;
signal massiv_buf : std_logic_vector (31 downto 0);
```

```
signal tdc1_buf : std_logic_vector(disc downto 0);
signal tdc2_buf : std_logic_vector(disc downto 0);

signal tdc1_int : integer;
signal tdc2_int : integer;

signal dout_tdc1 : std_logic;
signal dout_tdc2 : std_logic;

signal valid1_buf : std_logic;
signal valid2_buf : std_logic;

signal busy1 : std_logic;
signal busy2 : std_logic;
signal dout_cnt : std_logic;

signal inf_buf : std_logic;
signal busy_cnt : std_logic;
signal busy1_in : std_logic;
signal busy2_in : std_logic;
signal busy_cnt_in : std_logic;

signal dout : std_logic;

signal tx_o_buf : std_logic;
signal tx_o_true : std_logic;

signal zan1_buf : std_logic;
signal zan2_buf : std_logic;

signal valid_o_buf : std_logic;
```

```
type state_tdc is (idle, send1, send2, send3, send4,
   send5, send6, send7, send8, send9, send10, send11,
   wait1, wait2, wait3, wait4, wait5, wait6, wait7, wait8
   , wait9, wait10, waiting);
type reg_tdc is record
  state : state_tdc;
  t : integer range 0 to 13;
  valid : std_logic;
  s : std_logic_vector(7 downto 0);
  j : std_logic;
  tdc_mem : std_logic_vector(disc downto 0);
end record;

signal q1, q2, qin1, qin2: reg_tdc;

signal busy : std_logic;
signal busy_tdc1 : std_logic;
signal busy_tdc2 : std_logic;
signal tx_o_tdc1 : std_logic := '1';
signal tx_o_tdc2 : std_logic := '1';

begin

  sign_1_buf <= s1;
  sign_1 <= s1;
  sign_2_buf <= s2;
  sign_3_buf <= s3;
  sign_2 <= s2;
  d_i_b  <= not dat_i;
  clk_out <= clk;
  rst_buf <= rst;
```

```
map_rx : uart_rx

  generic map (
    frequency => 50,
    baudrate  => 19200
  )
  port map (
    d_i   => d_i_b,
    d_o   => time_window_buff,
    rst   => rst_buf,
    valid => valid_buff,
    clk   => clk
  );

map_coinsidence : coinsidence

  generic map (
    freq => 50,
    cnt_size => 24,
    disc     => disc
  )
  port map (
    s1 => sign_1_buf,
    s2 => sign_2_buf,
    s3 => sign_3_buf,
    s0 => s0,

    inf => inf_buf,

    time_window => t,

    s1_cnt => s1_cnt_buff,
    s2_cnt => s2_cnt_buff,
```

```
      s0_cnt => s0_cnt_buff,

      tx_o => tx_o_buf,

      tdc1 => tdc1_buf,
      tdc2 => tdc2_buf,

      zan1 => zan1_buf,
      zan2 => zan2_buf,

      valid1 => valid1_buf,
      valid2 => valid2_buf,

      valid_o => valid_o_buf,

      clk => clk,
      rst => rst_buf,

      s1_long_out => s1_long_out,
      s2_long_out => s2_long_out,
      f1_out => f1_out,
      f2_out => f2_out,
      s1_m_out => s1_m_out,
      s2_m_out => s2_m_out
    );

inf_buf <= r.inf;

dat_o <= tx_o_true and tx_o_tdc1 and tx_o_tdc2;

zan1 <= zan1_buf;
zan2 <= zan2_buf;
```

```
map_tx : uart_tx

  generic map (
    FREQUENCY => 50,
    BAUDRATE  => 19200,
    BYTE_SIZE => 8
  )
  port map (
    Clk_i => clk,
    Rst_i => rst_buf,
    Tx_o  => tx_o_true,
    Tx_busy_o => busy_buf,
    Tx_load_i => r.valid,
    Tx_data_i => r.s,

    led => open,
    led2 => open,
    led3 => open
  );

tdc1_tx : uart_tx
  generic map (
    FREQUENCY => 50,
    BAUDRATE  => 19200,
    BYTE_SIZE => 8
  )
  port map (
    Clk_i => clk,
    Rst_i => rst_buf,
    Tx_o  => tx_o_tdc1,
    Tx_busy_o => busy_tdc1,
    Tx_load_i => g.valid,
    Tx_data_i => g.s,
```

```
      led => open,
      led2 => open,
      led3 => open
  );

--  tdc2_tx : uart_tx
--    generic map (
--      FREQUENCY => 50,
--      BAUDRATE  => 19200,
--      BYTE_SIZE => 8
--    )
--    port map (
--      Clk_i => clk,
--      Rst_i => rst,
--      Tx_o  => tx_o_tdc2,
--      Tx_busy_o => busy_tdc2,
--      Tx_load_i => q2.valid,
--      Tx_data_i => q2.s,
--
--      led => open,
--      led2 => open,
--      led3 => open
--    );

  busy <= busy_buf or busy_tdc1 or busy_tdc2;

  inn_out <= inn;

  in_signal : process (clk)
  begin
    if rising_edge(clk) then
      if z < 50000333 then
```

```
      z <= z + 1;
      if (z = 1) or (z = 2) or (z = 10) or (z = 11)
        then
        inn <= '1';
      else
        inn <= '0';
      end if;
    else
      z <= 0;
    end if;
  end if;
end process;

get_spectrum : process (valid1_buf, sp_r)
variable v : get_sp;
begin

--  if valid1_buf = '1' then
--    t_i <= to_integer(unsigned(tdc1_buf));
--    sp_0(t_i) <= sp_0(t_i) + 1;
--  end if;

  v := sp_r;
  case v.state is
  when idle =>
    v.i := 0;
    if valid1_buf = '1' then
      v.tdc_int := to_integer(unsigned(tdc1_buf));
      v.state := search;
    end if;

  when search =>
    if v.i < n + 1 then
```

```
        if (v.i <= v.tdc_int) and (v.i + 1 > v.tdc_int)
          then
          v.sp(v.i) := v.sp(v.i) + 1;
          v.i := 0;
          v.state := idle;
        end if;
        v.i := v.i + 1;
      else
        v.i := 0;
        v.state := idle;

      end if;
    end case;
    sp_rin <= v;
end process;

send_sp : process (g, btn, busy, r.busy_o)
variable v : send_sp_reg;
begin

    v := g;
    v.valid := '0';

    case v.state is

      when idle =>
        v.busy_o := '0';
        if btn = '1' then
          v.sp := sp_r.sp;
          v.state := waiting_buf;

        end if;
```

```
when waiting_buf =>
  if (r.busy_o = '1') then --and (v.t < 12) then
    --v.state := wait2;

  else
    if (v.t < 12) then
      v.t := v.t + 1;
    else
      v.state := wait_sp_0;
      v.busy_o := '1';
      v.t := 0;
    end if;
  end if;

when wait_sp_0=>

  if (busy = '1') then --and (v.t < 12) then
    --v.state := wait2;

  else
    if (v.t < 12) then
      v.t := v.t + 1;
    else
      v.state := send_sp_0;
      v.t := 0;
    end if;
  end if;



when send_sp_0=>

  v.s := "00010001";
  v.valid := '1';
```

```
    v.state := wait_sp_1;

  when wait_sp_1=>

    if (busy = '1') then --and (v.t < 12) then
      --v.state := wait2;

    else
      if (v.t < 12) then
        v.t := v.t + 1;
      else
        v.state := send_sp_1;
        v.t := 0;
      end if;
    end if;

  when send_sp_1=>
    v.s := v.sp(v.i)(15 downto 8);
    v.valid := '1';


    v.state := wait_sp_2;

  when wait_sp_2=>

    if (busy = '1') then --and (v.t < 12) then
      --v.state := wait2;

    else
      if (v.t < 12) then
        v.t := v.t + 1;
```

```
      else
        v.state := send_sp_2;
        v.t := 0;
      end if;
    end if;

  when send_sp_2=>
    v.s := v.sp(v.i)(7 downto 0);
    v.valid := '1';

    v.state := wait_sp_3;

  when wait_sp_3=>

    if (busy = '1') then --and (v.t < 12) then
      --v.state := wait2;

    else
      if (v.t < 12) then
        v.t := v.t + 1;
      else
        v.state := send_sp_3;
        v.t := 0;
      end if;
    end if;

  when send_sp_3=>
    v.s := std_logic_vector(to_unsigned(v.i, 8));
    v.valid := '1';



    v.state := wait_sp_4;
```

```
when wait_sp_4=>

  if (busy = '1') then --and (v.t < 12) then
    --v.state := wait2;

  else
    if (v.t < 12) then
      v.t := v.t + 1;
    else
      v.state := send_sp_4;
      v.t := 0;
    end if;
  end if;

when send_sp_4=>
  v.s := "00000000";
  v.valid := '1';


  v.state := wait_sp_5;

when wait_sp_5=>

  if (busy = '1') then --and (v.t < 12) then
    --v.state := wait2;

  else
    if (v.t < 12) then
      v.t := v.t + 1;
    else
      v.state := send_sp_5;
      v.t := 0;
    end if;
```

```
    end if;

  when send_sp_5=>
    v.s := "00000000";
    v.valid := '1';


    v.state := wait_sp_6;

  when wait_sp_6=>

    if (busy = '1') then --and (v.t < 12) then
      --v.state := wait2;

    else
      if (v.t < 12) then
        v.t := v.t + 1;
      else
        v.state := send_sp_6;
        v.t := 0;
      end if;
    end if;

  when send_sp_6=>
    v.s := "00000000";
    v.valid := '1';


    v.state := wait_sp_7;

  when wait_sp_7=>

    if (busy = '1') then --and (v.t < 12) then
```

```
    --v.state := wait2;

  else
    if (v.t < 12) then
      v.t := v.t + 1;
    else
      v.state := send_sp_7;
      v.t := 0;
    end if;
  end if;

when send_sp_7=>
  v.s := "00000000";
  v.valid := '1';


  v.state := wait_sp_8;

when wait_sp_8=>

  if (busy = '1') then --and (v.t < 12) then
    --v.state := wait2;

  else
    if (v.t < 12) then
      v.t := v.t + 1;
    else
      v.state := send_sp_8;
      v.t := 0;
    end if;
  end if;

when send_sp_8=>
```

```
    v.s := "00000000";
    v.valid := '1';


    v.state := wait_sp_9;

  when wait_sp_9=>

    if (busy = '1') then --and (v.t < 12) then
      --v.state := wait2;

    else
      if (v.t < 12) then
        v.t := v.t + 1;
      else
        v.state := send_sp_9;
        v.t := 0;
      end if;
    end if;

  when send_sp_9=>
    v.s := "00000000";
    v.valid := '1';
    v.state := wait_sp_n;

  when wait_sp_n=>

    if (busy = '1') then --and (v.t < 12) then
      --v.state := wait2;

    else
      if (v.t < 12) then
        v.t := v.t + 1;
```

```
        else
          v.state := send_sp_n;
          v.t := 0;
        end if;
      end if;

    when send_sp_n=>
      v.s := "00001010";
      v.valid := '1';

      v.i := v.i + 1;
      if v.i < n + 1 then
        v.state := wait_sp_0;
      else
        v.state := idle;
        v.i := 0;
      end if;

    end case;

  gin <= v;

end process;

async : process(busy_buf, r, s1_cnt_buff, s2_cnt_buff,
  s0_cnt_buff, g.busy_o)
variable v : regtype;
begin

  v := r;
  v.valid := '0';

  case v.state is
```

```
when idle =>
    v.close := '0';
    v.inf := '0';
    v.busy_o := '0';

    if v.k > info_window then
    --if btn = '1' then
      v.s1_cnt_r := s1_cnt_buff;
      v.s2_cnt_r := s2_cnt_buff;
      v.s0_cnt_r := s0_cnt_buff;



    --   if busy_in = '0' then
        v.state := wait0;
        v.inf := '1';
        v.k := 0;
        v.close := '1';
      --v.busy_o := '1';
      --else
        --v.state := waiting;
      --end if;

    else
      v.k := v.k + 1;
      v.close := '0';
      v.inf := '0';

    end if;

when wait0 =>

  if (g.busy_o = '1') then --and (v.t < 12) then
```

```
    --v.state := wait2;

  else
    if (v.t < 12) then
      v.t := v.t + 1;
    else
      v.state := end1;
      v.t := 0;
    end if;
  end if;

when end1 =>
  v.inf := '0';
  v.busy_o := '1';
  v.s := "00010011";
  v.valid := '1';


  v.state := wait1;

when wait1 =>

  if (busy = '1') then --and (v.t < 12) then
    --v.state := wait2;

  else
    if (v.t < 12) then
      v.t := v.t + 1;
    else
      v.state := send11;
      v.t := 0;
    end if;
  end if;
```

```
when send11 =>

  v.s := v.s1_cnt_r(23 downto 16);
  v.valid := '1';
  v.state := wait11;

when wait11 =>

  if (busy = '1') then --and  then
    --v.state := send1;

  else
    if (v.t < 12) then
      v.t := v.t + 1;
    else
      v.state := send12;
      v.t := 0;
    end if;
  end if;

when send12 =>
  v.s := v.s1_cnt_r(15 downto 8);
  v.valid := '1';
  v.state := wait12;

when wait12 =>

  if (busy = '1') then --and  then
    --v.state := send1;

  else
    if (v.t < 12) then
```

```
        v.t := v.t + 1;
      else
        v.state := send13;
        v.t := 0;
      end if;
    end if;

  when send13 =>
    v.s := v.s1_cnt_r(7 downto 0);
    v.valid := '1';
    v.state := wait13;

  when wait13 =>

    if (busy = '1') then --and  then
      --v.state := send1;

    else
      if (v.t < 12) then
        v.t := v.t + 1;
      else
        v.state := send21;
        v.t := 0;
      end if;
    end if;

  when send21 =>
    v.s := v.s2_cnt_r(23 downto 16);
    v.valid := '1';
    v.state := wait21;

  when wait21 =>
```

```
    if (busy = '1') then --and (v.t < 12) then
      --v.state := wait2;

    else
      if (v.t < 12) then
        v.t := v.t + 1;
      else
        v.state := send22;
        v.t := 0;
      end if;
    end if;

  when send22 =>
    v.s := v.s2_cnt_r(15 downto 8);
    v.valid := '1';
    v.state := wait22;

  when wait22 =>

    if (busy = '1') then --and (v.t < 12) then
      --v.state := wait2;

    else
      if (v.t < 12) then
        v.t := v.t + 1;
      else
        v.state := send23;
        v.t := 0;
      end if;
    end if;

  when send23 =>
    v.s := v.s2_cnt_r(7 downto 0);
```

```
    v.valid := '1';
    v.state := wait23;

  when wait23 =>

    if (busy = '1') then --and (v.t < 12) then
      --v.state := wait2;

    else
      if (v.t < 12) then
        v.t := v.t + 1;
      else
        v.state := send01;
        v.t := 0;
      end if;
    end if;

  when send01 =>
    v.s := v.s0_cnt_r(23 downto 16);
    v.valid := '1';

    v.state := wait01;

  when wait01 =>

    if (busy = '1') then --and (v.t < 12) then
      --v.state := wait2;

    else
      if (v.t < 12) then
        v.t := v.t + 1;
      else
        v.state := send02;
```

```
        v.t := 0;
      end if;
    end if;

  when send02 =>
    v.s := v.s0_cnt_r(15 downto 8);
    v.valid := '1';

    v.state := wait02;

  when wait02 =>

    if (busy = '1') then --and (v.t < 12) then
      --v.state := wait2;

    else
      if (v.t < 12) then
        v.t := v.t + 1;
      else
        v.state := send03;
        v.t := 0;
      end if;
    end if;

  when send03 =>
    v.s := v.s0_cnt_r(7 downto 0);
    v.valid := '1';



    v.state := wait03;

  when wait03 =>
```

```
        if (busy = '1') then --and (v.t < 12) then
          --v.state := wait2;

        else
          if (v.t < 12) then
            v.t := v.t + 1;
          else
            v.state := send_n;
            v.t := 0;
          end if;
        end if;

      when send_n=>
        v.s := "00001010";
        v.valid := '1';



        v.state := idle;

    end case;
    rin <= v;
end process;

sync : process (clk, rst)
begin
    if rising_edge(clk) then
      if rst = '1' then
        r.state <= idle;
        r.s     <= (others => '0');
        r.valid <= '0';
        r.k <= 0;
        r.t <= 0;
        r.inf <= '0';
```

```
    r.s1_cnt_r <= (others => '0');
    r.s2_cnt_r <= (others => '0');
    r.s0_cnt_r <= (others => '0');
    r.busy_o <= '0';

    g.state <= idle;
    g.s <= (others => '0');
    g.valid <= '0';
    g.sp <= (others => (others => '0'));
    g.busy_o <= '0';

    q1.t <= 0;
    q1.state <= idle;
    q1.valid <= '0';
    q1.tdc_mem <= (others => '0');
    q1.s <= (others => '0');
    q1.j <= '0';

    q2.t <= 0;
    q2.state <= idle;
    q2.valid <= '0';
    q2.tdc_mem <= (others => '0');
    q2.s <= (others => '0');
    q2.j <= '0';

    sp_r.i <= 0;
    sp_r.tdc_int <= 0;
    sp_r.sp <= (others => (others => '0'));

  else
    r    <= rin;
    q1   <= qin1;
    q2   <= qin2;
```

```
            sp_r <= sp_rin;
              g <= gin;
          end if;
        end if;
      end process;

end Behavioral;
```