Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

Работа допущена к защите
директор ВШИСиСТ
_____ В.М. Ицыксон
«_____» _____ 2022 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

РАЗРАБОТКА УПРАВЛЯЮЩЕГО УСТРОЙСТВА КАЛИБРАТОРА СУБНАНОСЕКУНДНОЙ СИНХРОНИЗАЦИИ

по направлению 03.09.01 «Информатика и вычислительная техника»
по образовательной программе
03.09.01_01 «Вычислительные машины, комплексы, системы и сети»

Выполнил
студент гр. 3530901/80101 <*подпись*> Д.В. Пешков

Руководитель
к.т.н.,
доцент, <*подпись*> А.А. Лавров

Консультант
по нормоконтролю <*подпись*> А.Г. Новопашенный

Санкт-Петербург
2022

САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО

Институт компьютерных наук и технологий

УТВЕРЖДАЮ
директор ВШИСиСТ
_____ В.М. Ицыксон
«_____» _____ 2022г.

ЗАДАНИЕ

на выполнение выпускной квалификационной работы

студенту Пешкову Даниилу Валерьевичу гр. 3530901/80101

1. Тема работы: Разработка управляющего устройства калибратора субнаносекундной синхронизации.
2. Срок сдачи студентом законченной работы: 23.06.2022.
3. Исходные данные по работе: Техническая документация на систему субнаносекундной синхронизации «White Rabbit Project».
4. Содержание работы (перечень подлежащих разработке вопросов):
   4.1. Изучить техническую документацию по процедуре калибровки узлов сети White Rabbit.
   4.2. Изучить техническую документацию на используемые в калибраторе электронные узлы.
   4.3. Изучить принцип работы стробоскопического осциллографа.
   4.4. Портировать используемые открытые модули под целевую ПЛИС.
   4.5. Реализовать модули для проведения измерений при помощи стробирования.
   4.6. Спроектировать систему на кристалле и интегрировать в неё используемые аппаратные модули.
   4.7. Написать драйвера для используемой периферии.
   4.8. Протестировать и отладить полученное управляющее устройство на опытном образце калибратора, при необходимости осуществить доработку.
5. Перечень графического материала (с указанием обязательных чертежей): Отсутствует.

6. Консультанты по работе:
   6.1. А.Г. Новопашенный (нормоконтроль).
7. Дата выдачи задания: 06.05.2022.

Руководитель ВКР _____ А.А. Лавров

Задание принял к исполнению 06.05.2022

Студент _____ Д.В. Пешков

РЕФЕРАТ

На 35 с., 31 рисунок, 0 таблиц, 2 приложения

КЛЮЧЕВЫЕ СЛОВА: КАЛИБРУЮЩЕЕ УСТРОЙСТВО, ВЫСОКОТОЧНАЯ СИНХРОНИЗАЦИЯ, СТРОБИРОВАНИЕ. .

Тема выпускной квалификационной работы: «Разработка управляющего устройства калибратора субнаносекундной синхронизации».

В работе рассмотрен подход к калибровке синхронизации распределённых систем потоковой обработки данных. Рассмотрен способ снятия осциллограмм сигналов при помощи стробирования. Разработано устройство для автоматизация процесса калибровки для средств высокоточной синхронизации распределенных систем потоковой обработки данных

ABSTRACT

35 pages, 31 figures, 0 tables, 2 appendices

KEYWORDS: CALIBRATION DEVICE, HIGH-PRECISION SYNCHRONIZATION, STROBING, WHITE RABBIT.

The subject of the graduate qualification work is «Title of the thesis».

The paper considers an approach to calibration of synchronization of distributed streaming data processing systems. A method of removing waveforms of signals using strobing is considered. A device has been developed to automate the calibration process for means of high-precision synchronization of distributed streaming data processing systems.

СОДЕРЖАНИЕ

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

КА	–	Конечный автомат
ОУ	–	Объект управления
ПЛИС	–	Программируемая логическая интегральная схема
СнК	–	Система на кристалле
УУ	–	Управляющее устройство
ФАПЧ	–	Фазовая автоподстройка частоты
BSRAM	–	Burst Static Random Access Memory
DAC	–	Digital to analog converter
FIFO	–	first in, first out
GPIO	–	General-purpose input/output
ISA	–	Instruction Set Architecture
LUT	–	Look-Up table
NTP	–	Networking Time Protocol
PLL	–	Phase-Locked Loop
PPS	–	Pulse Per Second
PTP	–	Precision Time Protocol
RAM	–	Random Access Memory
ROM	–	Read-only memory
SFP	–	Small Form-factor Pluggable
UART	–	Universal Asynchronous Receiver-Transmitter
WR	–	White Rabbit
WR PTP	–	White Rabbit Precision Time Protocol

ВВЕДЕНИЕ

Для надёжного функционирования распределённых систем потоковой обработки данных, работающих в режиме реального времени, на исполняемые в них операции накладываются строгие временные ограничения. Многочисленные узлы таких систем могут находиться друг от друга на значительных расстояниях, что приводит к длительным (и не всегда одинаковым) задержкам при передаче сигналов между ними. Возникающие задержки приводят к рассинхронизации работы устройств систем, что влечёт за собой возникновение потенциально некорректных результатов выполнения операций.

С целью согласования всех устройств и обеспечения общего представления времени во всей сети применяются системы синхронизации часов. Такие системы гарантируют, что часы всех устройств сети отсчитывают время с одинаковой скоростью (выдают одинаковые показания в каждый момент времени).

Для передачи информации при синхронизации могут использоваться различные протоколы. Наибольшее распространение получили следующие два: NTP (Network Time Protocol) и PTP (Precision Time Protocol). Протокол NTP способен обеспечивать точность синхронизации времени до одной миллисекунды, а протокол PTP – до десяти миллисекунд.

Протоколы NTP и PTP не подходят для случая, когда необходима синхронизация с субнаносекундной точностью. Например, такая высокая точность требуется в распределённых системах, применяемых в экспериментах физики высоких энергий. Там они используются для потоковой обработки информации, поступающей с детекторов и ускорителей частиц.

Системы субнаносекундной синхронизации применяются в Большом Адронном Коллайдере, расположенным в ЦЕРН, в Швейцарии. Они также планируются к применению в строящемся комплексе «NICA» Объединённого института ядерных исследований (ОИЯИ) в Дубне. Другим приложением субнаносекундной синхронизации являются системы радиочастотного позиционирования, использующие технологию сверхширокополосной связи (UWB) и алгоритм позиционирования TDoA (Time Difference of Arrival).

Основные положения работы описаны в [1].

Цель работы: разработка управляющего устройства для устройства, выполняющего калибровку систем субнаносекундной синхронизации.

Решаемые в данной работе задачи: портирование и интеграция существующих открытых модулей в систему на кристалле, реализация модулей для проведения измерений по принципу стробоскопического осциллографа, написание драйверов для используемой периферии, написания управляющей программы для процессорного ядра, тестирование и отладка.

Разработка аппаратной части проекта выполнена на языке System Verilog, управляющей программы – на языке C. Для компиляции используется компилятор gcc под архитектуру RV32IM.

ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. White Rabbit

White Rabbit – система синхронизации часов. Разработана при сотрудничестве множества институтов и компаний. Изначально проект был начат для улучшения текущей системы синхронизации в Церне (Большой адронный коллайдер). Предполагалось использование для физических экспериментов, однако в процессе было создано обобщенное решение, которое нашло своё применение в различных сферах.

Характеристики:

- Субнаносекундная точность
- Большое количество синхронизируемых узлов
- Расстояния в десятки километров
- Канал передачи — 1 Gbps Ethernet
- Открытый исходный код

Достоинствами White Rabbit являются полностью открытый исходный код и аппаратура, а также использование существующих стандартов (Ethernet, PTP и т. д.).

1.2. Калибровка

Синхронизация в сети White Rabbit выполняется по протоколу WR PTP (White Rabbit Precison Time Protocol) — модифицированному протоколу PTP. (IEEE 1588). Однако для достижения субнаносекундной точности необходима дополнительная калибровка. Её принцип и математическое обоснование приведены в [6].

Внутри каждого устройства возникают постоянные задержки на приёме и передаче (Δ_{TXM},Δ_{RXM},Δ_{TXS},Δ_{RXS}) (рис.1.1), которые являются результатом задержек в SFP (Small Form-factor Pluggable) модуле, в электрических цепях и электронных компонентах. Так же присутствуют дополнительные задержки ε_M и ε_S, возникающие при восстановлении символов из входного потока данных.

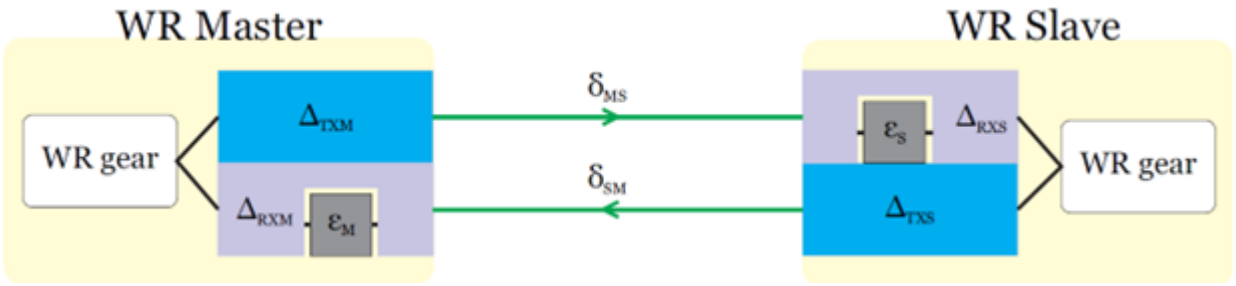


Рис.1.1. Модель соединения между двумя устройствами

Эти задержки калибруются отдельно на каждом устройстве и не являются предметом рассмотрения в данной работе.

Суммарная задержка распространения сигнала от ведущего устройства к ведомому устройству и обратно считается по формуле:

$$delay_{MM} = \Delta_{TXM} + \Delta_{RXS} + \varepsilon_S + \Delta_{TXS} + \Delta_{RXM} + \varepsilon_M + \delta_{MS} + \delta_{SM} \quad (1.1)$$

Обмен данными между двумя устройствами происходит по одной линии оптоволокна, работающей в полнодуплексном режиме. Для передачи в одну и другую сторону используется свет с разной длиной волны, из-за чего задержки δ_{MS} и δ_{SM} не равны – возникает асимметричность. Из-за асимметричности устройства не могут самостоятельно определить задержки δ_{MS} и δ_{SM}, а только суммарную задержку.

Определение коэффициента асимметричности оптоволокна позволит протоколу White Rabbit PTP обеспечить требуемую точность синхронизации устройств сети, так как будет учтена неоднородность задержек.

Коэффициент асимметричности определяется, как:

$$\alpha = \frac{\delta_{MS} - \delta_{SM}}{\delta_{SM}} \quad (1.2)$$

Измеряется коэффициент при помощи дополнительного короткого оптоволокна, с известной задержкой (рис.1.2).

Два устройства подключаются калибруемым оптоволокном (δ_2) и дополнительным (δ_1) отдельно, далее два устройства синхронизируются. После этого, при помощи осциллографа, измеряется разность фаз синхросигналов 1-PPS (Pulse Per Second)($skew_{PPS}$), генерируемых ведущим и ведомым устройством.

$$skew_{PPS} = t_{PPS_S} - t_{PPS_M} \quad (1.3)$$

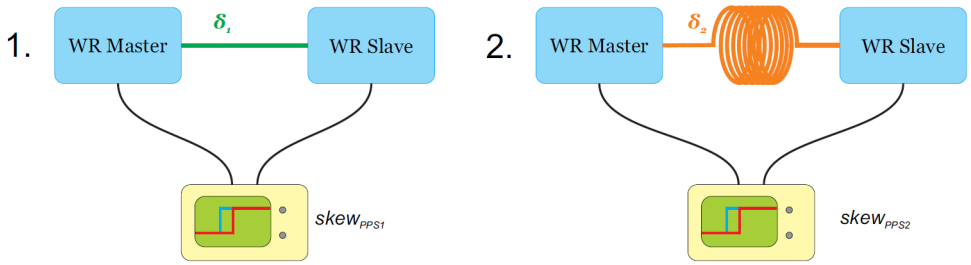


Рис.1.2. Измерение асимметричности про помощи дополнительного оптоволокна

Подставив измеренные значения в формулу 1.4 можно вычислить искомый коэффициент асимметричности.

$$\alpha = \frac{2\,(skew_{PPS2} - skew_{PPS1})}{\frac{1}{2}\delta_2 - (skew_{PPS2} - skew_{PPS1})} \tag{1.4}$$

Вышеописанный метод подходит для случаев, когда линия оптоволокна ещё не установлена. В иных случаях применяется немного изменённый метод с подключением петли от выхода 1-PPS одного устройства к другому (рис.1.3). Измеряется расхождение фаз при передаче сигнала PPS от одного устройства и от другого. Полученные значения усредняются (1.5).

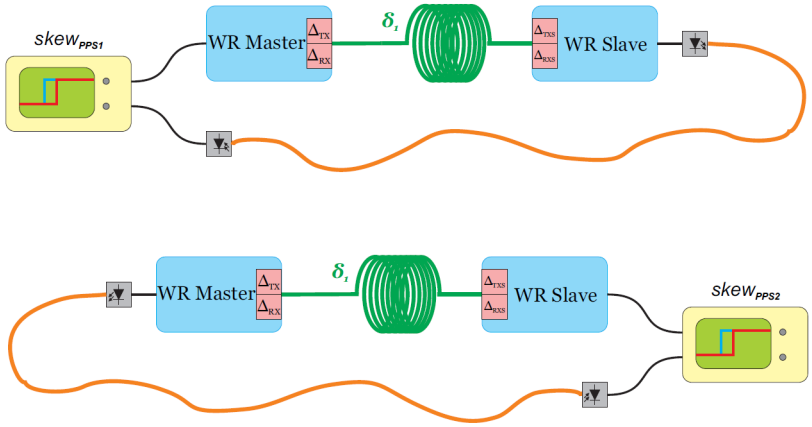


Рис.1.3. Измерение асимметричности установленной линии

$$skew_{PPS} = \frac{1}{2}\,(skew_{PPS1} + skew_{PPS2}) \tag{1.5}$$

Далее это значение может быть использовано для вычисления коэффициента по формуле 1.4. Однако для данного метода так же требуется короткое оптоволокно с известной задержкой.

1.3. Актуальность

Для процесса калибровки предполагается использовать устройство, способное измерять отрезки времени меньше, чем 1 нс, чтобы обеспечить субнаносекундную точность, т.е. иметь частоту дискретизации 1 GSa. Если применить правило «пятикратного превышения частоты дискретизации», то используемый осциллограф должен иметь частоту дискретизации выше 5 GSa.

Цена осциллографов с такими характеристиками крайне высока. Актуальность данной работы заключается в том, что предлагается разработать относительно бюджетное устройство, работающее по принципу стробоскопического осциллографа.

Сигналы PPS от калибруемых устройств являются периодическими. Это позволяет применить для их обнаружения, захвата и анализа устройство, работающее по принципу стробоскопического осциллографа.

1.4. Стробоскопический осциллограф

Стробоскопические осциллографы предназначены для обнаружения, захвата и анализа периодических сигналов. Принцип работы стробоскопического осциллографа проиллюстрирован на рис.1.4.

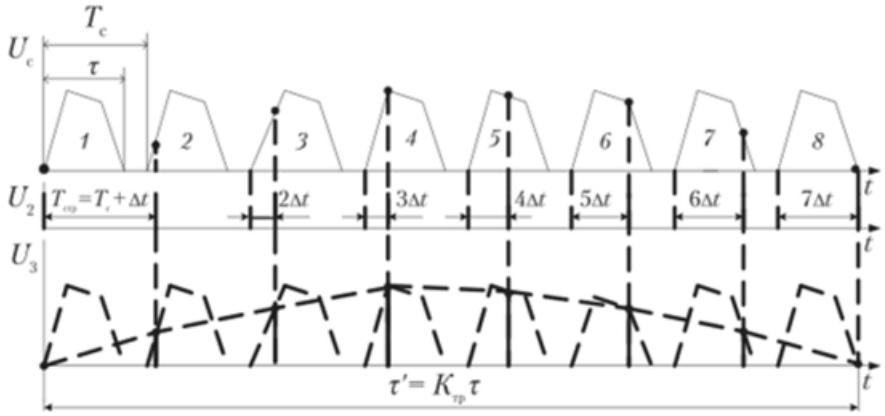


Рис.1.4. Принцип работы стробоскопического осциллографа

Условные обозначения:

U_c – исследуемый периодический сигнал

T_c – период исследуемого сигнала

τ – длительность исмпульса исследуемого сигнала

Δt – шаг считывания исследуемого сигнала

U_2 – стробы осциллографа

U_3 – снятая осциллограмма сигнала

Таким образом, для снятия очередной точки изменяется смещение Δt. Частота дискретизации определяется минимальным смещением, которое может задать осциллограф.

Осциллограмма исследуемого сигнала снимается следующим образом:

1. Осциллограф захватывает группу выборок измеряемого сигнала U_c с периодом T
2. Осциллограф смещает точку запуска на величину Δt и захватывает следующий набор выборок
3. Процесс повторяется, в результате чего строится осциллограмма сигнала

Описанный принцип действия стробоскопических осциллографов обеспечивает высокую чувствительность и широкую полосу пропускания этих приборов. Ключевое значение для работы стробоскопического осциллографа имеет шаг сдвига точки захвата сигнала. Частота дискретизации при этом становится не столь несущественна. Большой объём памяти также не требуется, так требуется захватывать и обрабатывать малое число выборок.

ГЛАВА 2. АППАРАТНАЯ ПЛАТФОРМА И АЛГОРИТМ ПРОВЕДЕНИЯ ИЗМЕРЕНИЙ

Управляющее устройство реализовано на базе ПЛИС, так как есть необходимость разрабатывать специфические аппаратные модули.

На основании требований к количеству логических ячеек в ПЛИС и необходимой обвязке студентом Владиславом Михайловским была разработана структурная схема устройства (рис.2.2) и печатная плата (рис.2.1).

В качестве аппаратной платформы УУ (управляющее устройство) выступает ПЛИС серии LittleBee китайского производителя Gowin Semiconductor – GW1N-UV9QN88C6/I5.

Данная ПЛИС имеет следующие характеристики:

- количество логических блоков LUT4 – 8640 шт.
- количество триггеров – 6480 шт.
- объём FLASH – 408 Кбит
- количество блоков BSRAM – 26 шт.
- объём блоков SRAM – 468 Кбит
- количество блоков PLL (ФАПЧ) – 2 шт.
- напряжение питания ядра – 1.8–3.3 В
- количество доступных для пользователя выходов I/O – 77
- среди них дифференциальных пар – 19

К УУ подключаются два измерительных канала, которыми необходимо управлять для проведения измерений. ПЛИС тактируется высокостабильным тактовым сигналом с частотой 125 МГц. Связь с устройством верхнего уровня осуществляется через FTDI чип FT2232H[5].

Перед разработкой УУ необходимо определить ОУ (объект управления), которым предстоит управлять.

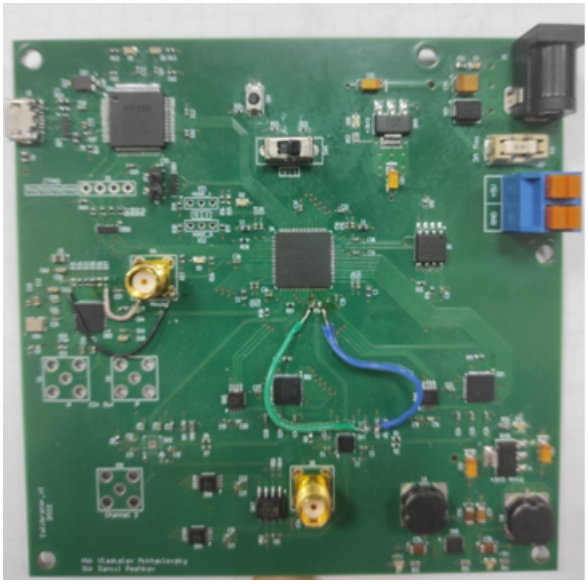


Рис.2.1. Печатная плата устройства

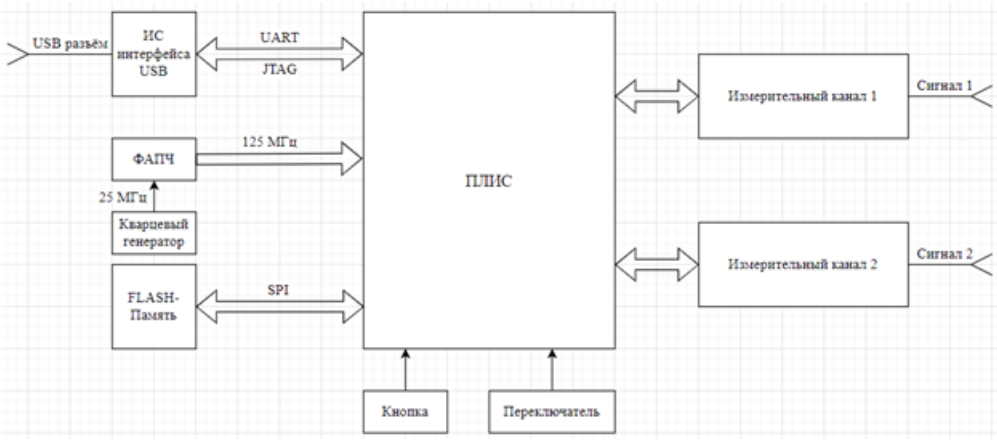


Рис.2.2. Структурная схема устройства

2.1. Измерительный канал

На рис.2.3 приведена функциональная схема одного из измерительных каналов.

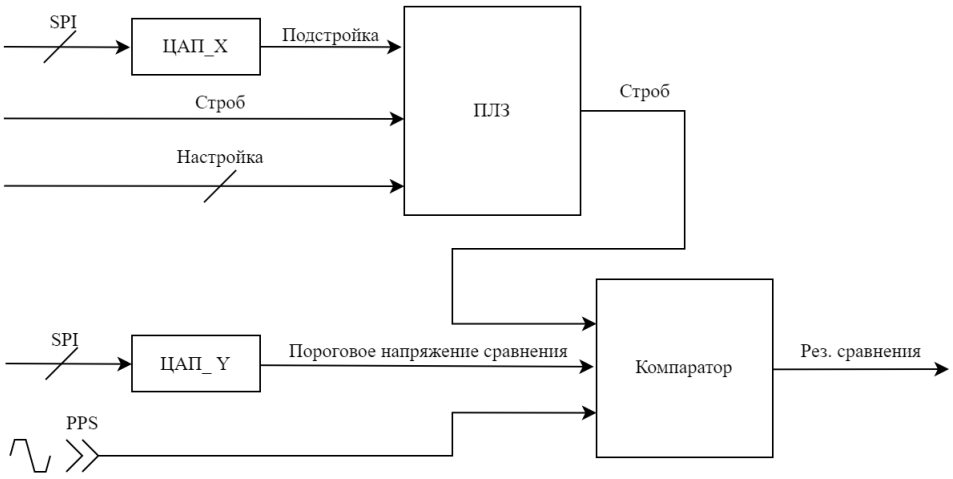


Рис.2.3. Функциональная схема измерительного канала

SPI (Serial Peripheral Interface) — интерфейс для задания напряжения на ЦАП_X/Y

Подстройка/настройка — сигналы управления ПЛЗ. Задают величину, на которую происходит задержка сигнала строба

Строб — сигнал, при получении которого компаратор фиксирует текущее значение своего выхода и удерживает его

ЦАП_X — Цифро-Аналоговый Преобразователь. Используется для задания напряжения, управляющего подстройкой задержки ПЛЗ

ЦАП_Y – Цифро-Аналоговый Преобразователь. Используется для задания величины порогового напряжения сравнения для компаратора

ПЛЗ — программируемая линия задержки. Используется для изменения задержки распространения строба на заданную величину

Компаратор — устройство, производящее сравнение входного Сигнала PPS и пороговым значением сравнения

Пороговое напряжение сравнения — задаваемое ПЛИС значение напряжения, с которым происходит сравнение

PPS — входной синхросигнал от калибруемого устройства

2.2. Алгоритм проведения измерений

Разрабатываемое устройство, помимо узкоспециализированного применения для измерения разности фаз синхросигналов калибруемых устройств, также может работать в режиме стробоскопического осциллографа общего назначения.

2.2.1. Режим измерения разности фаз

Для определения разности фаз сначала находится фаза синхросигнала опорного устройства (то, от которого тактируется калибратор). Для этого на ЦАП_У устанавливается пороговое напряжение, характерное для используемого стандарта ввода-вывода измеряемого сигнала. Затем, увеличивая значение задержки, находится такое, при котором на выходе компаратора на втором канале будет логическая единица. Для нахождения разности фаз ($skew_{PPS}$) (рис.2.4) необходимо умножить установленный код на линии задержки на шаг изменения задержки.

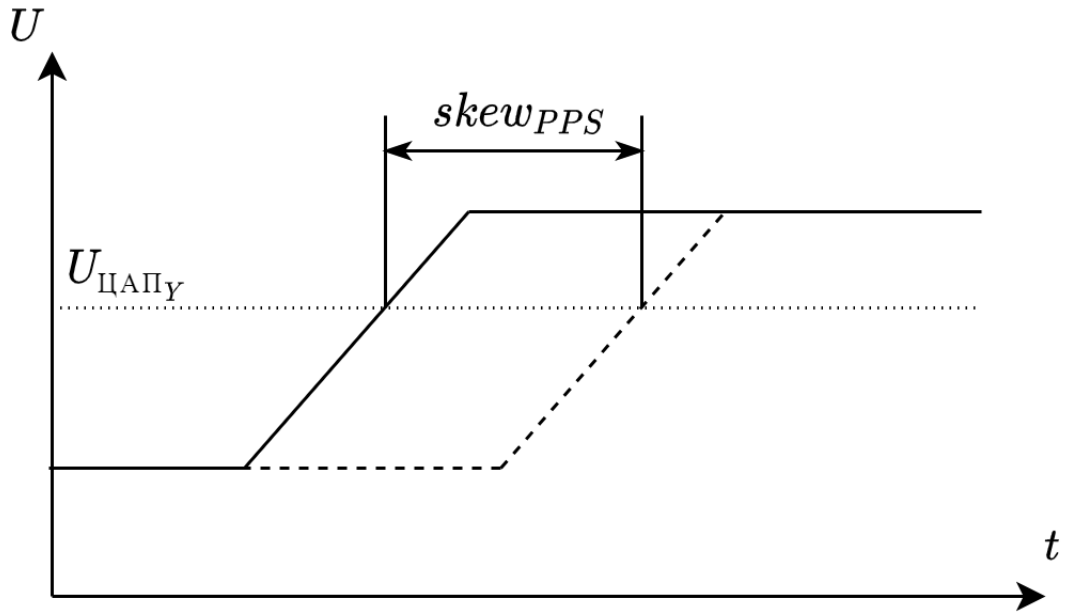


Рис.2.4. Измерение разности фаз

2.2.2. Режим стробоскопического осциллографа

Перед началом проведения измерения необходимо определить частоту измеряемого сигнала, и начать генерировать стробы с той же частотой. Стробы блокируют текущий выход компаратора, и на его выходе остаётся результат сравнения в момент прихода строба. При помощи изменения задержки можно сдвигать момент прихода строба, тем самым сдвигая измеряемую точку по оси OX.

Для поиска значения напряжения в измеряемой точке необходимо изменять пороговое напряжение сравнения на компараторе. Наблюдая изменения выхода компаратора, в зависимости от значения порогового напряжения, можно судить о том, найдено ли значение в измеряемой точке.

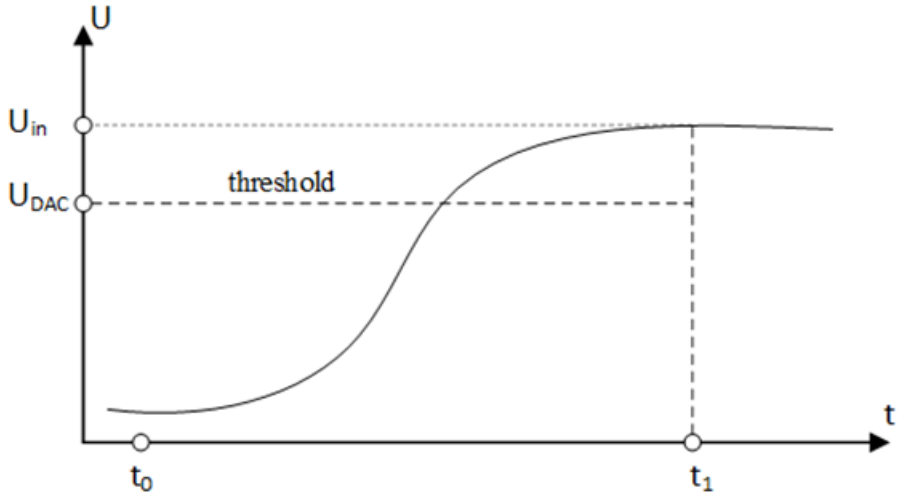


Рис.2.5. Снятие осциллограммы сигнала

На рис.2.5 приведён пример измерения. t_0 – точка от которой измеряется сигнал (момент прихода строба), t_1 – измеряемая точка (момент прихода задержанного строба на компаратор). Временной отрезок $[t_0 , t_1]$ настраивается линией задержки. Если в момент защёлкивания на выходе компаратора логическая единица, значит измеряемое напряжение выше, чем пороговое. Перед приходом следующего строба пороговое напряжение повышается. Так продолжается до тех пор, пока на выходе компаратора не будет логического нуля. В таком случае мы считаем, что измеряемая точка имеет значения напряжение равное пороговому.

Затем увеличивается значение задержки и начинается поиск значения напряжения в следующей точке.

2.3. Пороговое напряжение сравнения

Для задания порогового напряжения используется ЦАП AD5662 от Analog Devices. AD5662 – 16-битный ЦАП с гарантированной точностью 12 бит.

С опорным напряжением 3.3 В, шаг изменения напряжения – $\frac{3.3}{2^{16}} \approx 50$ мкВ (если не учитывать нелинейность задания напряжения). Для задания напряжения используется интерфейс SPI (рис.2.6).

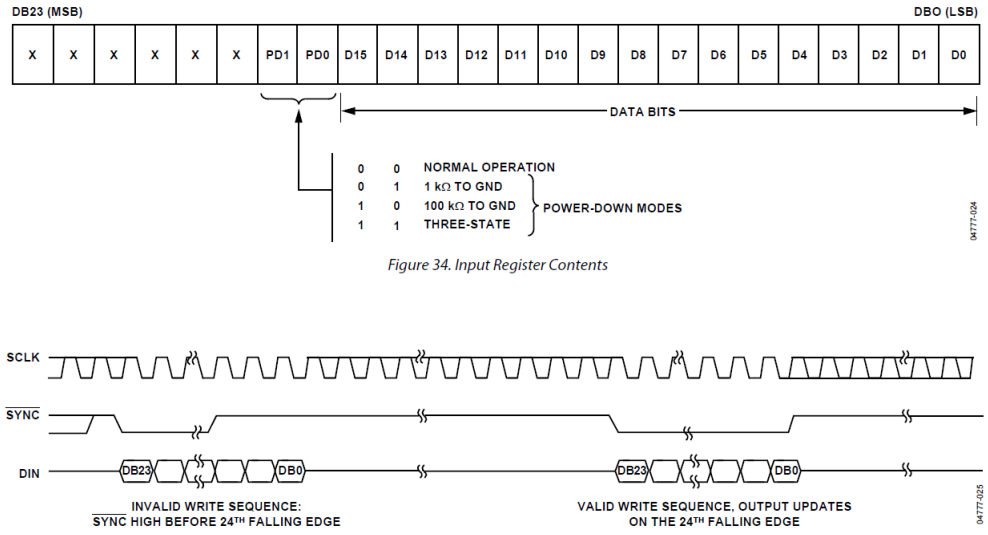


Figure 34. Input Register Contents

Рис.2.6. Интерфейс взаимодействия с AD5662

При напряжении питания 3.3 В максимальная частота сигнала SCLK – 20 МГц. Максимальное время установки выходного напряжения – 10 мкс (обычно 8 мкс). Описание используемого ЦАП приведено в документации[3].

2.4. Линия задержки

Для задержки строба используется программируемая линия задержки SY100EP196V пр-ва Microchip. Задержка задаётся 10-битным цифровым кодом с шагом ≈ 10 пс (рис.2.7). Так же есть аналоговый вход FTUNE, которым УУ может точнее точнее подстраивать задержку в диапазоне 30 пс при помощи ЦАП_X(рис.2.8) (на данный момент не используется).

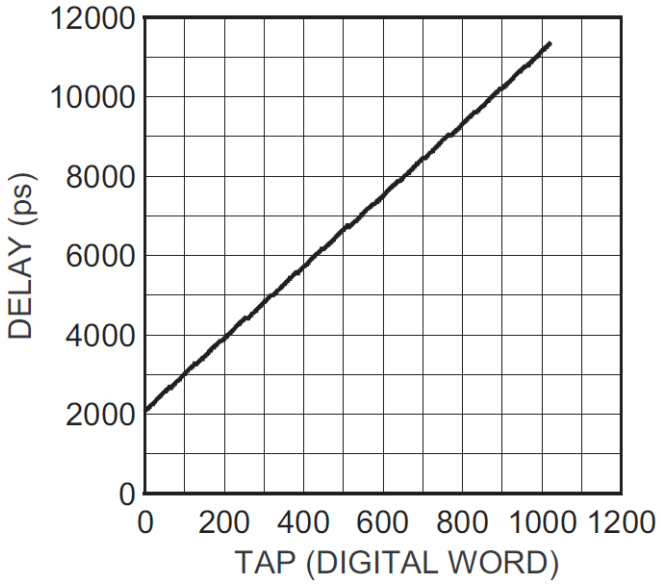


Рис.2.7. Задержка, задаваемая цифровым входом

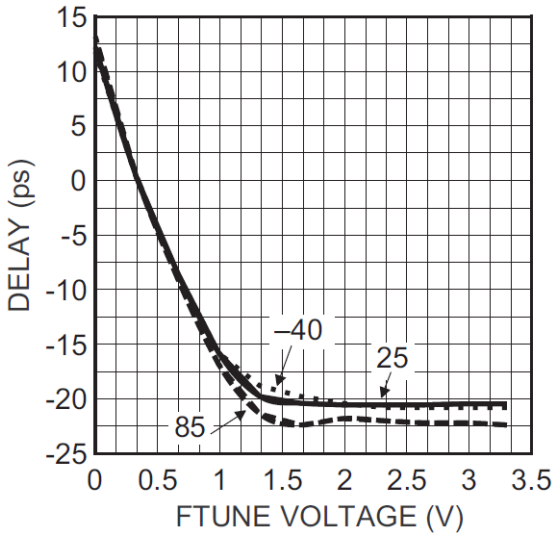


Рис.2.8. Подстройка задержки через аналоговый вход FTUNE

Описание используемой программируемой линии задержки приведено в документации[12].

2.5. Компаратор

Для сравнения используется компаратор ADCMP582 от Analog Devices. Основным параметром, который необходимо учесть является t_{PL} (рис.2.9).

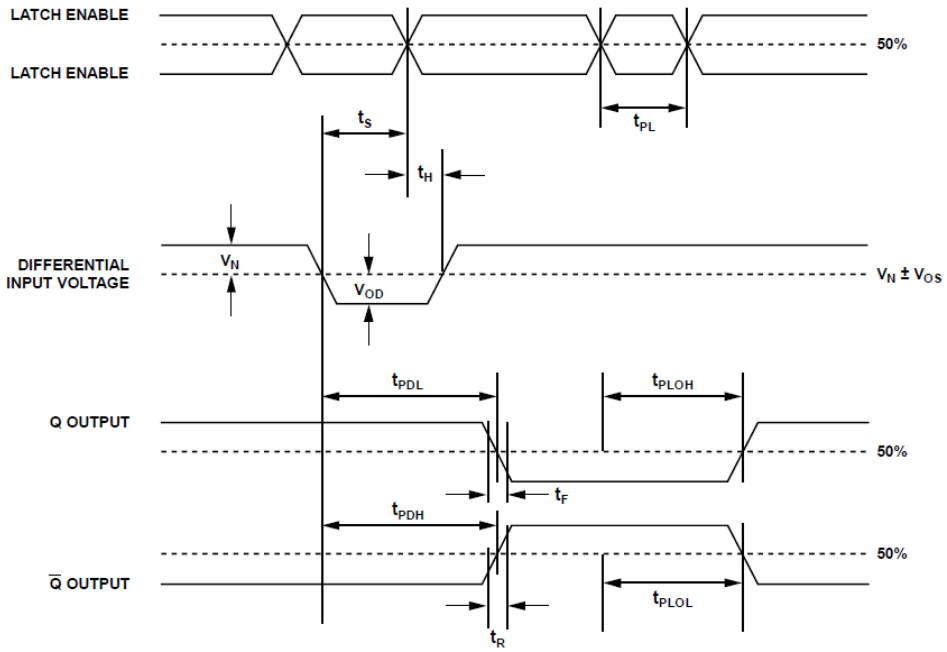


Рис.2.9. Временная диаграмма ADCMP582

t_{PL} – минимальное время, в течение которого вход защёлки (Latch Enable) должен быть высоким, чтобы результат сравнения попал на выход компаратора.

Описание использованного компаратора приведено в документации[2].

ГЛАВА 3. РАЗРАБОТКА ПРОГРАММНО-АППАРАТНОЙ ЧАСТИ УПРАВЛЯЮЩЕГО УСТРОЙСТВА

Все исходные коды аппаратных описаний находятся в директории */rtl/*.

3.1. Описание верхнего уровня

Описание верхнего уровня находится в файле */rtl/calsoc_top.sv*.

Управляющее устройство реализовано в виде СнК (Система на кристалле) (рис.3.1) на базе открытого процессорного ядра *PicoRV32*, основанного на открытой архитектуре RISC-V.

Все периферийные модули подключаются к ядру через шину *Wishbone*. Арбитраж на шине выполняет открытый модуль *wbxbar*.

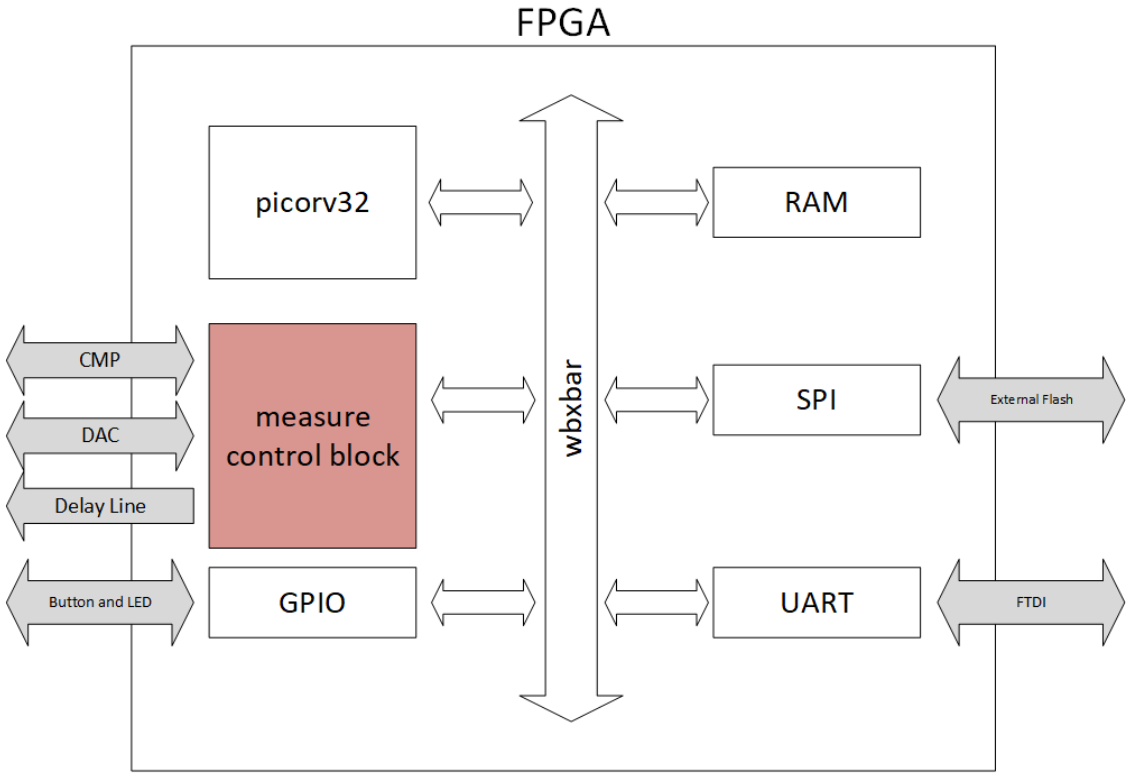


Рис.3.1. Структурная схема СнК

Все периферийные устройства разделяют между собой общее адресное пространство.

0x00000000 – 0x00FFFFFF – RAM

0x01000000 – 0x01FFFFFF – ROM загрузичка

0x02000000 – 0x02FFFFFF – GPIO
0x03000000 – 0x03FFFFFF – UART1
0x04000000 – 0x04FFFFFF – память программы
0x05000000 – 0x05FFFFFF – измерительный модуль

3.2. Измерительный модуль

Измерительный модуль выдаёт все необходимые управляющие сигналы для проведения измерений и логически разделён на несколько компонентов:

- *stb_gen* – модуль, измеряющий частоту сигнала и генерирующий стробы
- *ch_measure_ctl* – модуль, непосредственно управляющий измерением одного канала
- *spi_master* – модуль, реализующий взаимодействие с AD5662
- *sc_fifo* – FIFO, накапливающее измеренные значения

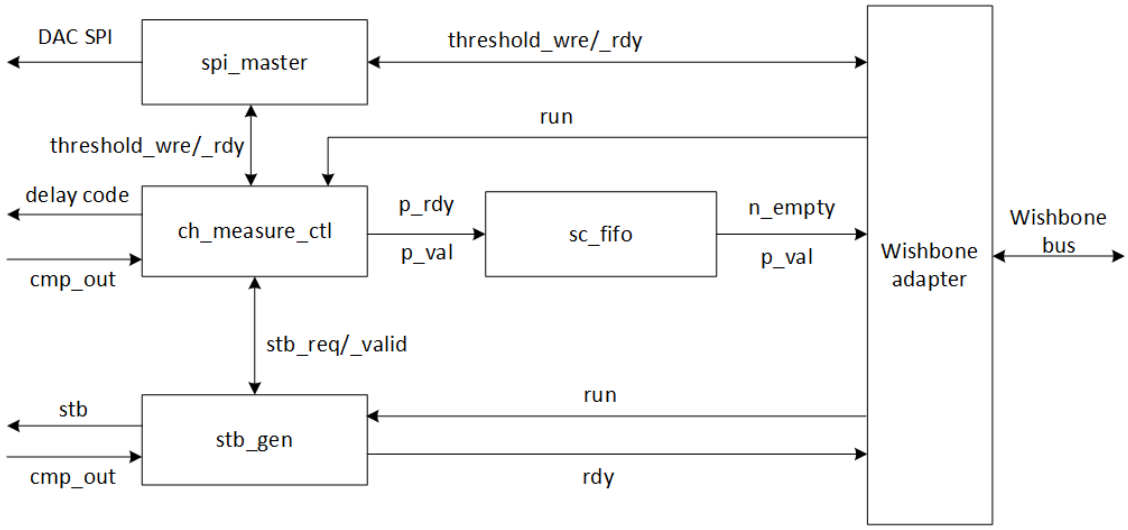


Рис.3.2. Структурная схема измерительного модуля (*measure_unit*)

Все описанные ранее модули подключены в модуле верхнего уровня *measure_unit* (рис.3.2). В нём же реализована вся логика управлением проведением измерений через шину Wishbone.

Далее будет отдельно рассмотрен каждый модуль.

3.2.1. Модуль stb_gen

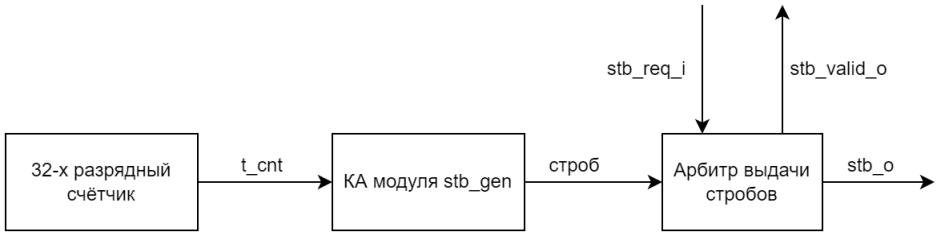


Рис.3.3. Структурная схема блока генерации стробов (*stb_gen*)

Для определения частоты измеряемого сигнала необходимо измерить время между двумя фронтами. Для измерения используется 32-разрядный счётчик, один отсчёт которого равняется 8 нс (125 МГц).

Для измерения сигнала с периодом равным одной секунде необходима разрядность 27 бит, однако для возможности измерять сигналы с меньшей частотой и кратности степени двойки разрядность увеличена до 32 бит.

Реализация «в лоб» не может работать стабильно на целевой ПЛИС из-за задержек на цепочке переносов в 32-разрядном сумматоре. Для корректной работы на такой частоте необходимо конвейеризировать счётчик (рис.3.4).

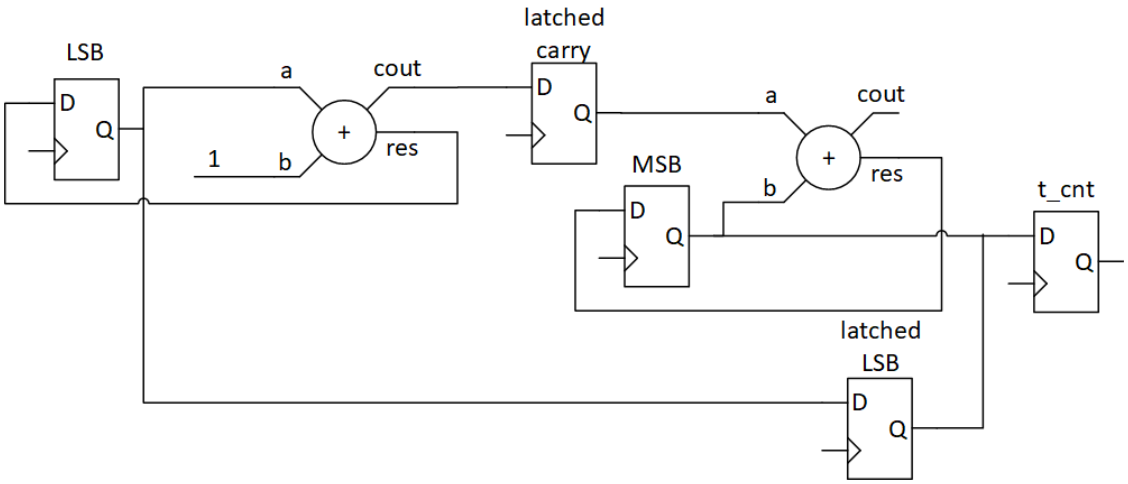


Рис.3.4. Конвейерный 32-х разрядный счётчик

К младшим байтам (регистр *LSB*) каждый такт прибавляется 1, текущее значение младших байт защёлкивается в регистре *latched LSB*. При переполнении, перенос защёлкивается в регистре *latched carry*. К старшим байтам (регистр *MSB*) прибавляется сохранённый перенос из-за младших байт. Выход счётчика – комбинация значений регистров *MSB* и *latched LSB*.

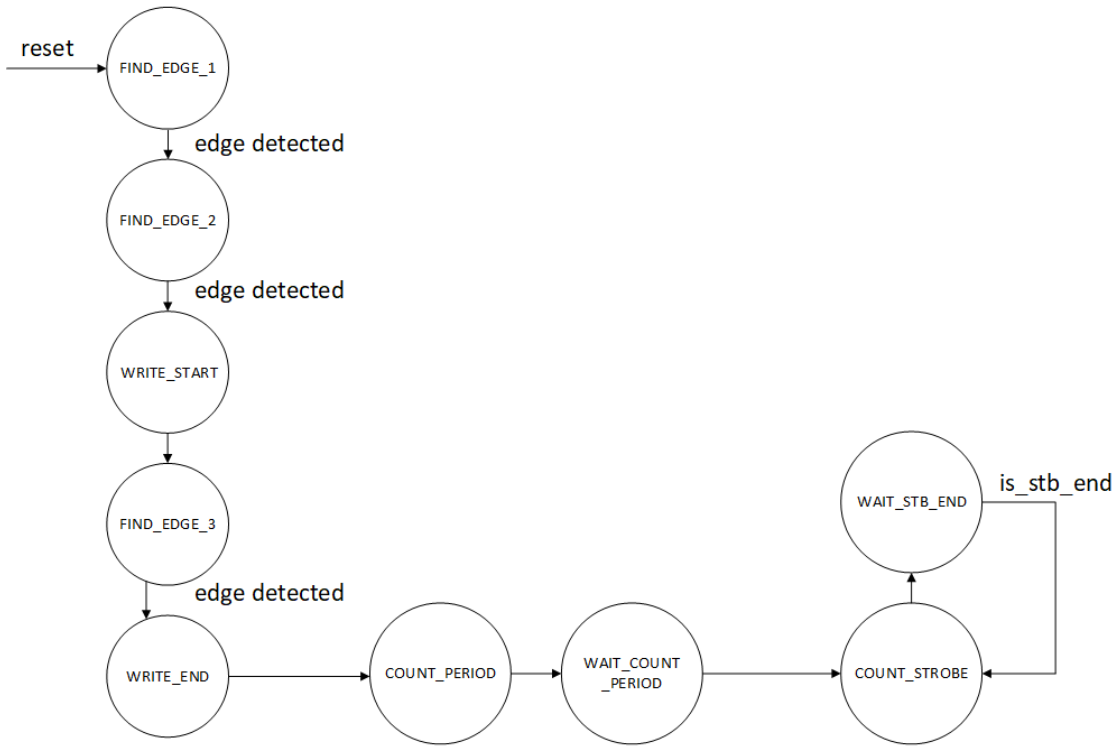


Рис.3.5. Конечный автомат модуля *stb_gen*

Нас рис.3.5 представлен конечный автомат модуля *stb_gen*. Вся логика максимально упрощена, так как, если на одной регистровой передаче будет много комбинаторной логики, при синтезе под целевую ПЛИС не получится выдержать требуемые t_{setup} и t_{hold}.

FIND_EDGE_1 – состояние после сброса, ожидание первого фронта на входе *sig_i*

FIND_EDGE_2 – ожидание второго фронта на входе *sig_i*

WRITE_START – запись текущего значения *t_cnt* в *t_start*

FIND_EDGE_3 – ожидание третьего фронта на входе *sig_i*

WRITE_END – запись текущего значения *t_cnt* в *t_end*

COUNT_PERIOD – вычисление периода сигнала (*t_end* – *t_start*)

WAIT_COUNT_PERIOD – задержка на 1 такт

COUNT_STROBE – начало расчёта времени начала и конца отрицательного импульса на выходе строба

WAIT_STB_END – ожидание конца строба и переход к расчёту новой

При записи 1 в регистр управлением данным модулем происходит его сброс, и начинается определение частоты сигнала.

Помимо 32-х разрядного счётчика, конвейеризация была необходима для всех операций с 32-х битными числами. В модуле *two_cycle_32_adder* реализован двухтактный сумматор, однако использованный подход идентичен тому, что используется в счётчике.

Отдельного упоминания стоит оптимизация операции проверки на равенство двух чисел. Для этого необходимо сравнить попарно все биты чисел, что выполняется параллельно и не должно увеличивать максимальный путь, однако при синтезе получалась цепочка с последовательным сравнением всех разрядов (рис.3.6).

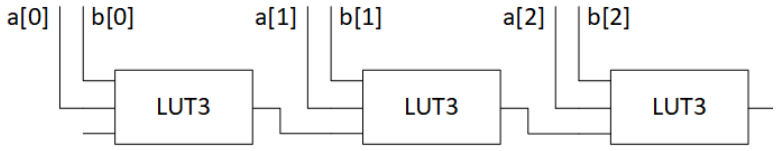


Рис.3.6. Реализация сравнения при синтезе

Для исправления этого пришлось явно заменить операцию сравнения на исключающее ИЛИ и свёртку по ИЛИ-НЕ.

Выходом строба управляет арбитр, который держит компаратор в защёлкнутом состоянии. По приходе запроса на строб (сигнал *stb_req_i*) (рис.3.3) арбитр пропускает один строб и выставляет логическую единицу на выходе *stb_valid_o*.

Для тестирования модуля были написаны тестбенчи *tb_stb_gen.sv* и *tb2_stb_gen.sv*. На рис.3.7 и рис.3.8 представлены результаты моделирования.

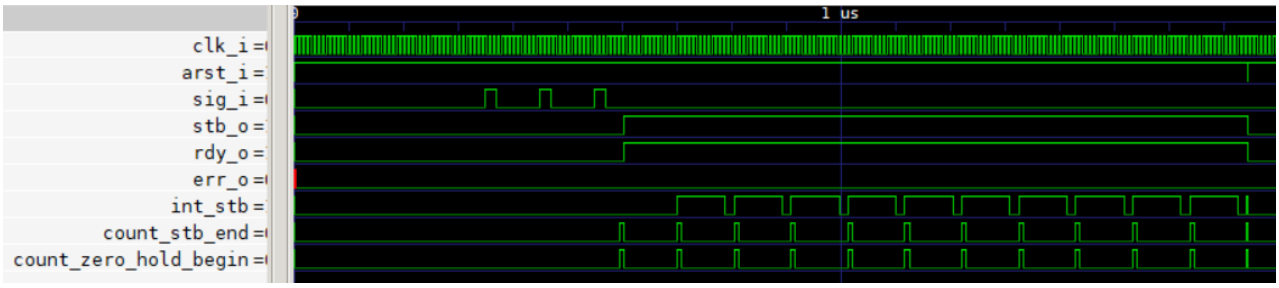


Рис.3.7. Пример определения частоты входного сигнала

```
iverilog -o test_stb_gen.vvp -g2005-sv tb_stb_gen.sv
./stb_gen.sv:108: vvp.tgt sorry: Case unique/unique0 qualities are ignored.
vvp -n test_stb_gen.vvp
VCD info: dumpfile dump.vcd opened for output.
measured signal  T=         100 ns
generated strobe T=         104 ns
err=          4 ns      clk T=   8 ns
counted period=       104

measured signal  T=       20000 ns
generated strobe T=       20000 ns
err=          0 ns      clk T=   8 ns
counted period=     20000

measured signal  T=      200000 ns
generated strobe T=      200000 ns
err=          0 ns      clk T=   8 ns
counted period=    200000

measured signal  T=     1333333 ns
generated strobe T=     1333336 ns
err=          3 ns      clk T=   8 ns
counted period=   1333336

OK!
tb_stb_gen.sv:69: $finish called at 20196276 (1ns)
```

Рис.3.8. Вывод в консоль при симуляции

Видно, что сигналы с периодом, кратным 8 нс измеряются корректно, при измерении других – ошибка меньше 8 нс.

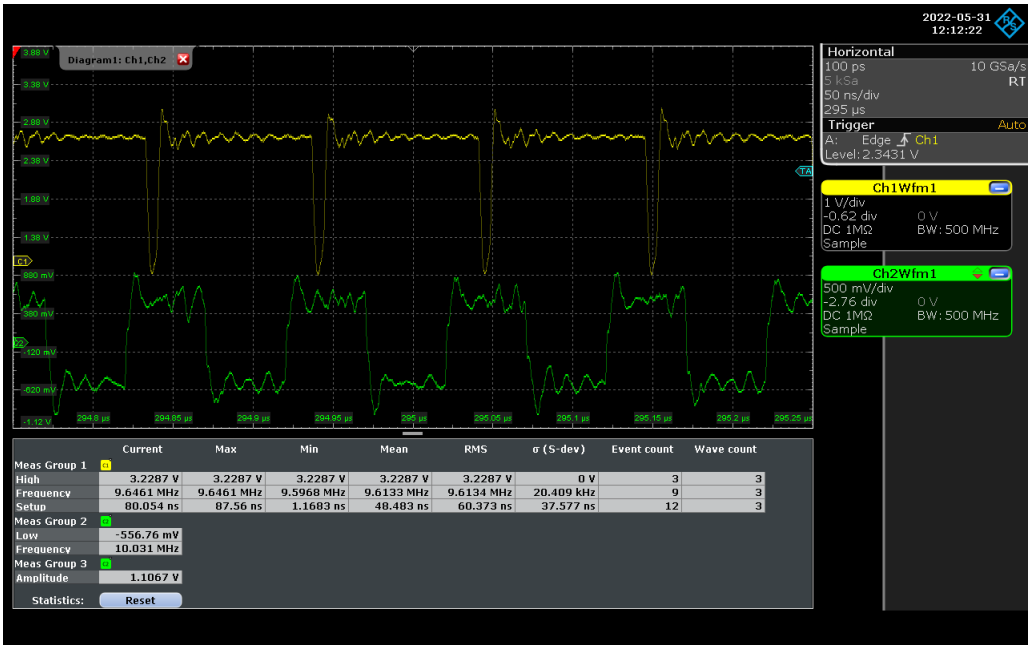


Рис.3.9. Генерация стробов для 10 Мгц

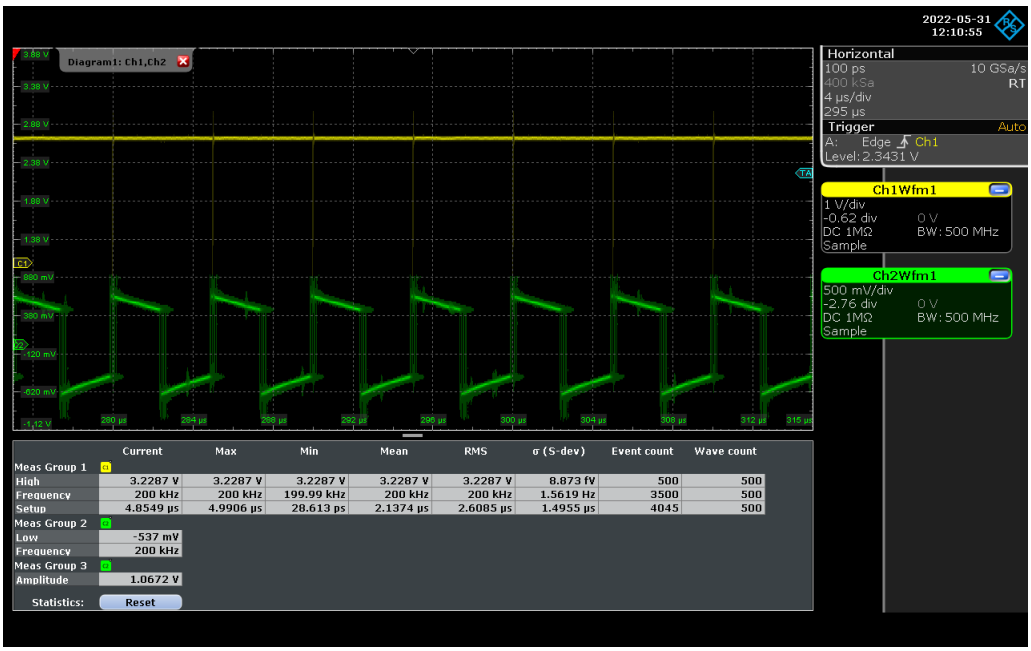


Рис.3.10. Генерация стробов для 200 кГц

На рис.3.9 и рис.3.10 приведены осциллограммы генерации стробов для 10 МГц и 200 кГц. Видно, что точно определить не кратную частоту невозможно – при генерации получаются стробы с частотой 9.6 Мгц, а не 10 Мгц.

3.2.2. Модуль ch_measure_ctl

Данный модуль управляет измерением одного из каналов – выставляет необходимое пороговое напряжение и задержку, а затем выставляет запрос на строб. После прохода строба, в зависимости от текущего и предыдущего выхода компаратора, принимается решение о следующем значении задержки и порогового напряжения.

Для ускорения снятия осциллограммы применяется предсказание направления изменения сигнала. Для этого, после нахождения очередной точки, изменяется только задержка, после чего на основании выхода компаратора принимается решение – измеряемый сигнал нарастает или уменьшается.

Для запроса генерации строба используются сигналы *stb_req_o* и *stb_valid_i*, которые подключены к модулю *stb_gen*.

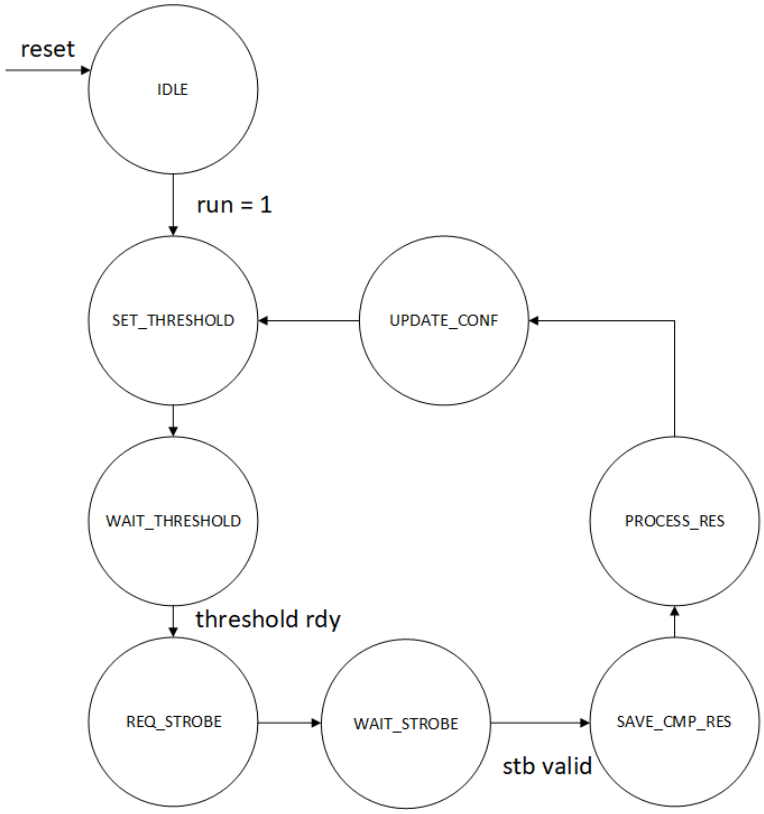


Рис.3.11. Конечный автомат модуля *ch_measure_ctl*

На рис.3.11 представлен конечный автомат описываемого модуля.

IDLE – состояние покоя

SET_THRESHOLD – отправление запроса на установку порогового напряжения (*threshold_wre_o* = 1)

WAIT_THRESHOLD – ожидание установки порогового напряжения (*threshold_rdy_i* == 1)

REQ_STROBE – установка запроса на стробу (*stb_req_o* = 1)

WAIT_STROBE – ожидание прохода строба (*stb_valid_i* == 1)

SAVE_CMP_RES – обновление текущего и предыдущего выхода компаратора

PROCESS_RES – принятие решения о статусе поиска точки (определение направления поиска или поиск точки) и о выборе направления поиска

UPDATE_CONF – обновление регистров с текущей задержкой и пороговым напряжением на основании принятого ранее решения

В состоянии PROCESS_RES, в зависимости от выхода компаратора переключаются два других конечных автомата, отвечающих за статус поиска точки и направления поиска.

На рис.3.12 приведён пример снятия осциллограммы сигнала.

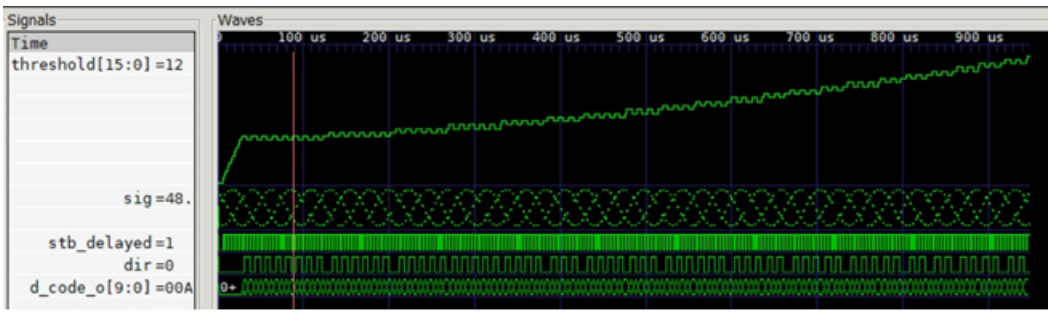
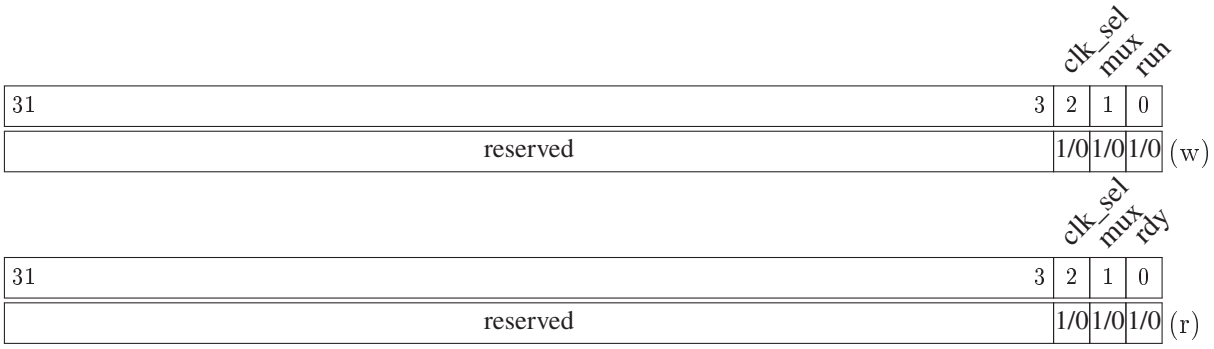


Рис.3.12. Пример снятия осциллограммы с угадыванием направления изменения сигнала

3.2.3. Регистры управления измерительным блоком

Управление измерительным модулем осуществляется через регистры, размещённые в адресном пространстве процессора (memory-mapped) на шине Wishbone[14] (рис.3.13, рис.3.14, рис.3.15, рис.3.16, рис.3.17).

Register 3.1: STB_GEN_CTL



Register 3.2: STB_GEN_PERIOD

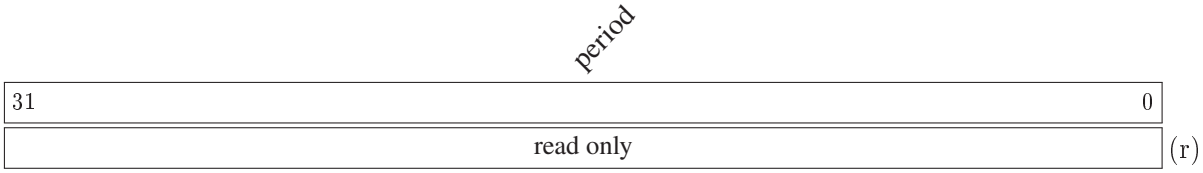


Рис.3.13. Регистры для управления модулем *stb_gen*

run – запись 1 начинает измерение частоты входного сигнала

mux – выбор канала для измерения (0 – первый канал, 1 – второй)

clk_sel – выбор источника тактового сигнала для генерации стробов (0 – внутренний, 1 – внешний)

rdy – 1 при окончании измерения

period – измеренное значение периода (кол-во отсчётов счётчика)

Register 3.3: CH_CTL_DELTA_REG

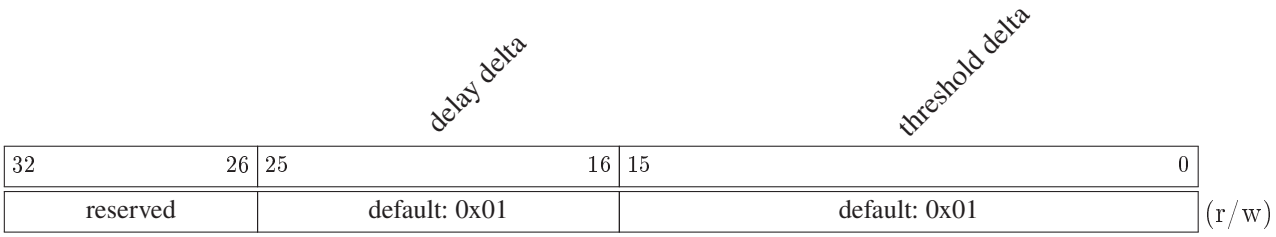


Рис.3.14. Регистр для задания параметров измерения

threshold delta – шаг изменения порогового напряжения (разрешение по оси OY)
delay delta – шаг изменения задержки (разрешение по оси OX)

Register 3.4: W_THRESHOLD

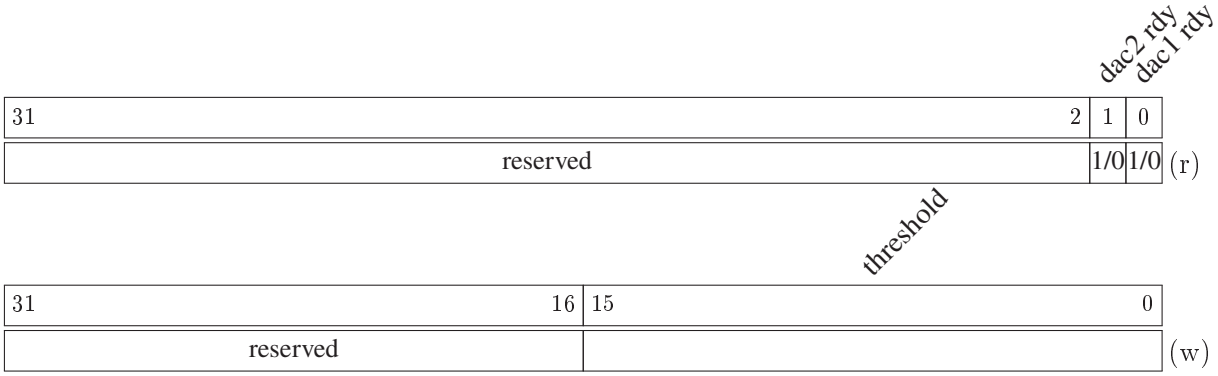


Рис.3.15. Регистр для установки порогового напряжения

dac1/2 rdy – ЦАП на первом/втором канале установил указанное напряжение
threshold – запись в это поле устанавливает заданное напряжение на оба канала

Register 3.5: MU_CTL_1/2

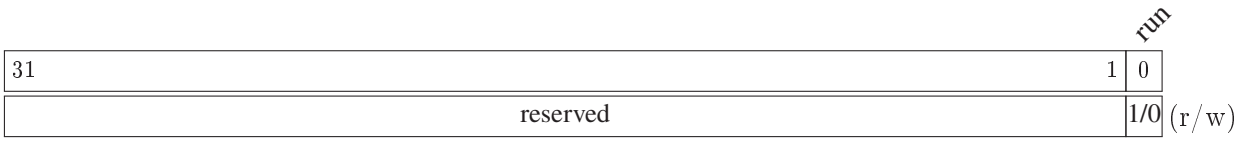


Рис.3.16. Регистр для управление измерением канала

run – запись 1 начинает измерение

Register 3.6: MU_CH_1/2_VAL

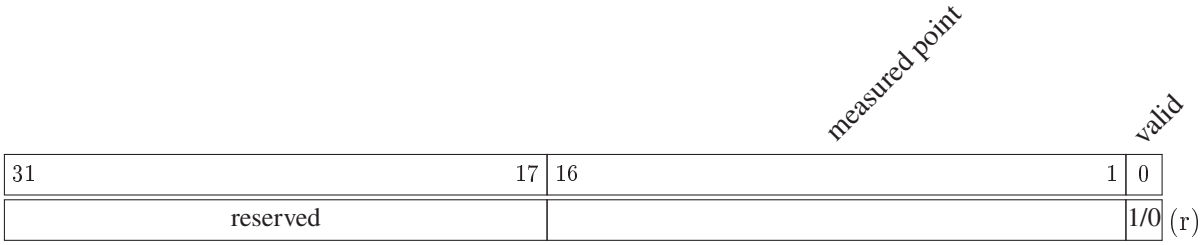


Рис.3.17. Регистр для чтения измеренных значений

measured point – значение измеренной точки
valid – 1 если FIFO не пустое и прочитано измеренное значение

3.3. Программная часть управляющего устройства

Программная часть состоит из двух компонентов: загрузчика (*/bootloader/*) и управляющей программы (*/firmware/*).

Загрузчик необходим для обновления прошивки на FLASH памяти внутри калибратора. При запуске устройства загрузчик проверяет включен ли переключатель на печатной плате калибратора, если нет, то исполнение передаётся управляющей программе.

Драйвера для используемой периферии находятся в директории (*/firmware/src/dev/*). В заголовочном файле *dev.h* определены базовые адреса подключенной периферии.

Все измерения проводятся аппаратно, процессорное ядро используется для подачи команд на проведение измерений и взаимодействия с устройством верхнего уровня, а так же отладки аппаратных модулей. В дальнейшем планируется расширение программной части в связи с добавлением интеграции White Rabbit.

Результат выводится через последовательный порт в консоль подключенного к калибратору ПК.

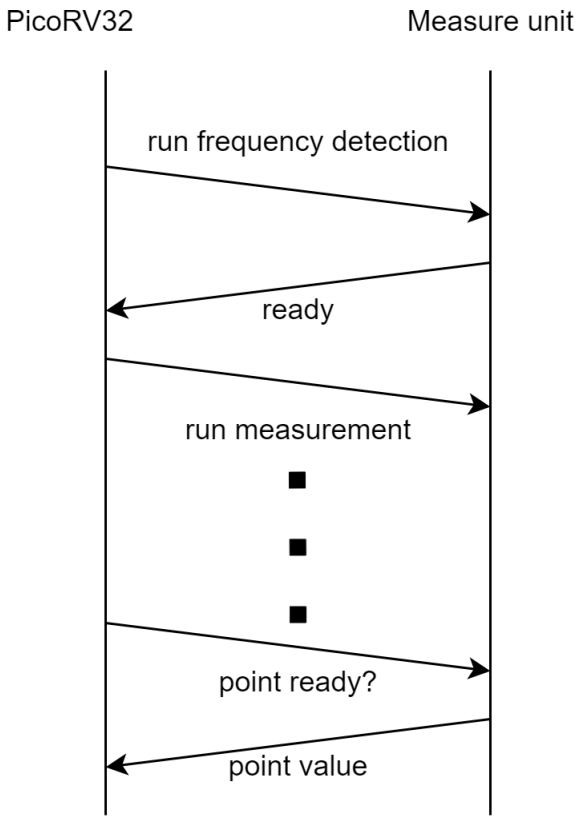


Рис.3.18. Описание взаимодействия процессорного ядра и измерительного модуля

На рис.3.18 изображена схема взаимодействия управляющей программы и измерительного модуля. На начальном этапе запускается определение частоты, после этого подаётся команда на проведение снятия осциллограммы. По готовности, измеренные точки считываются ядром.

ЗАКЛЮЧЕНИЕ

В результате работы рассмотрен метод калибровки узлов сети White Rabbit и разработано управляющее устройство для калибратора субнаносекундной синхронизации. Поставленные задачи: портирование под целевую ПЛИС и интеграция в СнК открытых модулей, создание модулей для проведения измерений при помощи стробирования, написание и отладка встраиваемого программного обеспечения – выполнены в полном объёме.

В процессе работы были освоены:

- Маршрут проектирования на ПЛИС пр-ва Gowin Semiconductor
- Открытая шина для СнК Wishbone и разработка ведомых устройств для неё
- Применение открытого синтезируемого ядра на базе архитектуры RISC-V – PicoRV32
- Проектирование и реализация СнК на основе открытых модулей
- Конфигурация компилятора и компоновщика под используемую ISA и СнК

На данный момент калибратор находится на этапе отладки, планируется вторая ревизия платы с исправлением ошибок, найденных в ходе отладки. В планах на дальнейшую разработку имеется доработка прошивки, связанная с:

- Улучшением метода определения частоты измеряемого сигнала и генерации строб
- Доработкой алгоритма и реализации снятия осциллограммы
- Добавлением интеграции в инфраструктуру White Rabbit
- Доработкой режима стробоскопического осциллографа общего назначения

Также планируется проведение процедуры калибровки разработанного устройства, что позволит повысить точность проводимых измерений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Пешков Д.В., Михайловский В.С. Калибрующее устройство для средств высокоточной синхронизации распределённых систем потоковой обработки данных // Экономика и Индустрия 5.0 в условиях новой реальности (ИНПРОМ-2022): сборник трудов Всероссийской научно-практической конференции с зарубежным участием 28-30 апреля 2022. - СПб.: ПОЛИТЕХ-ПРЕСС, 2022. - С. 680-684.

2. ADCMP582BCPZ Datasheet [Электронный ресурс] // Analog Devices URL: https://www.analog.com/media/en/technical-documentation/data-sheets/adcmp580_581_582.pdf (дата обращения: 16.06.2022).

3. AD5662BRMZ Datasheet [Электронный ресурс] // Analog Devices URL: https://www.analog.com/media/en/technical-documentation/data-sheets/ad5662.pdf (дата обращения: 16.06.2022).

4. Another Wishbone (or even AXI-lite) Controlled UART [Электронный ресурс] // Github URL: https://github.com/ZipCPU/wbuart32 (дата обращения: 16.06.2022).

5. FT2232HL Datasheet [Электронный ресурс] // FTDI Chip URL: https://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT2232H.pdf (дата обращения: 16.06.2022).

6. Grzegorz Daniluk. White Rabbit calibration procedure [Электронный ресурс] // CERN BE-CO-HT URL: https://www.ohwr.org/project/white-rabbit/uploads/76cdbdbadccc9d6c54d5caf246550fbf/WR_Calibration-v1.1-20151109.pdf (дата обращения: 16.06.2022).

7. GW1N series of FPGA Products Data Sheet DS100-2.7.1E [Электронный ресурс] // GOWIN Semiconductor Corp. URL: https://gowinsemi.com/upload/database_doc/1752/document/6296db1b68af5.pdf (дата обращения: 16.06.2022).

8. GowinSynthesis User Guide [Электронный ресурс] // GOWIN Semiconductor Corp. URL: http://cdn.gowinsemi.com.cn/SUG550E.pdf (дата обращения: 16.06.2022).

9. Gowin FPGA Primitive User Guide [Электронный ресурс] // GOWIN Semiconductor Corp. URL: https://www.gowinsemi.com/upload/database_doc/39/document/5bfcff2ce0b72.pdf (дата обращения: 16.06.2022).

10. OpenCores Free and Open Source gateware IP cores [Электронный ресурс] // OpenCores URL: https://opencores.org/ (дата обращения: 16.06.2022).

11. PicoRV32 - A Size-Optimized RISC-V CPU [Электронный ресурс] // Github URL: https://github.com/YosysHQ/picorv32 (дата обращения: 16.06.2022).

12. SY100EP196V Datasheet [Электронный ресурс] // Microchip URL: https://

ww1.microchip.com/downloads/aemDocuments/documents/TCG/ProductDocuments/DataSheets/SY100EP196V-3.3V-5V-2.5GHz-Programmable-Delay-Chip-with-Fine-Tune-C pdf (дата обращения: 16.06.2022).
13. WB2AXIP: Bus interconnects, bridges, and other components [Электронный ресурс] // Github URL: https://github.com/ZipCPU/wb2axip (дата обращения: 16.06.2022).
14. WISHBONE System-on-Chip (SoC)Interconnection Architecture for Portable IP Cores [Электронный ресурс] // OpenCores URL: https://cdn.opencores.org/downloads/wbspec_b4.pdf (дата обращения: 16.06.2022).

Приложение 1

Описание файлов проекта

Все исходные коды находятся в репозитории на Github https://github.com/daniilpeshkov/calsoc

./rtl/ – директория с исходными кодами аппаратных описаний
./firmware/ – директория с исходными кодами управляющей программы
./bootloader/ – директория с исходными кодами загрузчика
./syn/ – директория с файлами для синтеза (назначения выводов ПЛИС, временные ограничения, конфигурации втроенного логического анализатора)
./boot.py – скрипт для прошивки калибратора
./calsoc.gprj – файл для открытия проекта в Gowin EDA

Приложение 2

Исходные коды измерительного модуля

Листинг П2.1. *stb_gen.sv*

```
module stb_gen #(
  parameter OFFSET = 20,
  parameter T_CNT_WIDTH = 32
) (
  input wire clk_i,
  input wire arst_i,

  input wire sig_i,

  output logic err_o,
  output logic rdy_o,
  output logic stb_o,
  output logic [T_CNT_WIDTH-1:0] stb_period_o,

  input  logic stb_req_i,
  output logic stb_valid_o,
  output logic debug_stb_o
);
  // localparam T_CNT_WIDTH       = 32;
  localparam ZERO_HOLD_CYCLES   = 2;

  logic int_stb = 0;
  assign debug_stb_o = int_stb;

  logic sig_synced;

  //stb req interface

  logic stb_oe; // stb_oe == 1 blocks strobe generation

  assign stb_o = (rdy_o ? int_stb | stb_oe : int_stb);

  assign stb_valid_o = stb_oe & rdy_o;

  logic prev_stb_req;

  always_ff @(posedge clk_i) prev_stb_req <= stb_req_i;

```

```
40  logic req_posedge;
41  assign req_posedge = ~prev_stb_req & stb_req_i;
42
43  logic is_zero_hold_start;
44  logic is_stb_end;
45
46  always_ff @(posedge clk_i, negedge arst_i) begin
47    if (~arst_i) begin
48      stb_oe = 0;
49    end else begin
50      casex ({req_posedge, is_stb_end})
51        2'bx1:  stb_oe <= 1;
52        2'b1x:  stb_oe <= 0;
53      endcase
54    end
55  end
56
57  sync_ff #(
58    .WIDTH (1),
59    .STAGES(2)
60  ) sig_i_sync_ff_inst (
61    .clk_i (clk_i),
62    .data_i(sig_i),
63    .data_o(sig_synced)
64  );
65
66  logic prev_sig; //edge detect
67
68  always_ff @(posedge clk_i) begin
69    prev_sig <= sig_synced;
70  end
71
72  logic sig_posedge;
73  assign sig_posedge = sig_synced & ~prev_sig;
74
75  typedef enum logic[8:0] {
76    FIND_EDGE_1      = 9'b000000001,
77    FIND_EDGE_2      = 9'b000000010,
78    WRITE_START      = 9'b000000100,
79    FIND_EDGE_3      = 9'b000001000,
80    WRITE_END      = 9'b000010000,
81    COUNT_PERIOD     = 9'b000100000,
82    WAIT_COUNT_PERIOD    = 9'b001000000,
83    COUNT_STROBE     = 9'b010000000,
84    WAIT_STB_END     = 9'b100000000
```

```
85  } stb_gen_state;
86
87  stb_gen_state state = FIND_EDGE_1;
88
89  logic [T_CNT_WIDTH-1 : 0] t_cnt /* synthesis syn_keep=1
       syn_preserve=1 syn_ramstyle="registers" */;
90  logic [T_CNT_WIDTH-1 : 0] t_start;
91  logic [T_CNT_WIDTH-1 : 0] t_end;
92  stb_gen_state next_state;
93
94
95  logic count_zero_hold_begin, zero_hold_begin_valid;
96  logic count_stb_end, stb_end_valid;
97
98  assign count_zero_hold_begin = (state == COUNT_STROBE ? 1 :
       0);
99  assign count_stb_end = (state == COUNT_STROBE ? 1 : 0);
100
101 always_comb begin
102   unique case (state) /* sythesis parallel_case*/
103     FIND_EDGE_1:      if (sig_posedge) next_state =
          FIND_EDGE_2;
104               else next_state = state;
105     FIND_EDGE_2:      if (sig_posedge) next_state =
          WRITE_START;
106               else next_state = state;
107     WRITE_START:      next_state = FIND_EDGE_3;
108     FIND_EDGE_3:      if (sig_posedge) next_state =
          WRITE_END;
109               else next_state = state;
110     WRITE_END:        next_state = COUNT_PERIOD;
111     COUNT_PERIOD:     next_state = WAIT_COUNT_PERIOD;
112     WAIT_COUNT_PERIOD:    next_state = COUNT_STROBE;
113     COUNT_STROBE:     next_state = WAIT_STB_END;
114     WAIT_STB_END:     if (is_stb_end) next_state =
          COUNT_STROBE;
115               else next_state = state;
116     default:        next_state = state;
117   endcase
118 end
119
120 always_ff @(posedge clk_i, negedge arst_i) begin
121   if (~arst_i) begin
122     state = FIND_EDGE_1;
123   end else begin
```

```
124       state <= next_state;
125     end
126   end
127
128   logic [T_CNT_WIDTH-1:0] adder_zero_hold_res;
129   logic [T_CNT_WIDTH-1:0] adder_stb_end_res;
130
131   always_ff @(posedge clk_i) begin
132     stb_period_o <= (state == COUNT_PERIOD ? t_end - t_start :
          stb_period_o);
133   end
134
135   localparam MAGIC_CONST = 3;
136
137   logic [T_CNT_WIDTH-1:0] period_minus_zero_hold;
138   always_ff @(posedge clk_i) period_minus_zero_hold <=
       stb_period_o - (ZERO_HOLD_CYCLES+MAGIC_CONST); //magic
       constat due to computation pipeline
139
140   two_cycle_32_adder adder_zero_hold_begin (
141     .clk_i  (clk_i),
142     .a_i  (t_cnt),
143     .b_i  (period_minus_zero_hold),
144     .valid_i(count_zero_hold_begin),
145     .valid_o(zero_hold_begin_valid),
146     .res_o  (adder_zero_hold_res)
147   );
148
149   two_cycle_32_adder adder_stb_end (
150     .clk_i  (clk_i),
151     .a_i  (t_cnt),
152     .b_i  (stb_period_o - MAGIC_CONST), //magic constat due to
          computation pipeline
153     .valid_i(count_stb_end),
154     .valid_o(stb_end_valid),
155     .res_o  (adder_stb_end_res)
156   );
157
158   logic [T_CNT_WIDTH-1:0] latched_zero_hold_res;
159   logic [T_CNT_WIDTH-1:0] latched_stb_end_res;
160
161   always_ff @(posedge clk_i) latched_zero_hold_res <= (
       zero_hold_begin_valid ? adder_zero_hold_res :
       latched_zero_hold_res);
```

```
162  always_ff @(posedge clk_i) latched_stb_end_res <= (
       stb_end_valid ? adder_stb_end_res : latched_stb_end_res);
163
164
165  logic is_zero_hold_start_lo;
166  logic is_zero_hold_start_hi;
167  logic is_stb_end_lo;
168  logic is_stb_end_hi;
169
170  always_ff @(posedge clk_i) is_zero_hold_start_lo <=  ~|(
       t_cnt[15:0] ^ latched_zero_hold_res[15:0]);  //t_cnt ==
       latched_zero_hold_res;
171  always_ff @(posedge clk_i) is_zero_hold_start_hi <=  ~|(
       t_cnt[31:16] ^ latched_zero_hold_res[31:16]);  //t_cnt ==
        latched_zero_hold_res;
172  always_ff @(posedge clk_i) is_zero_hold_start <=
       is_zero_hold_start_lo & is_zero_hold_start_hi;
173
174  always_ff @(posedge clk_i) is_stb_end_lo <=  ~|(t_cnt[15:0]
       ^ latched_stb_end_res[15:0]);  //t_cnt ==
       latched_zero_hold_res;
175  always_ff @(posedge clk_i) is_stb_end_hi <=  ~|(t_cnt[31:16]
        ^ latched_stb_end_res[31:16]);  //t_cnt ==
       latched_zero_hold_res;
176  always_ff @(posedge clk_i) is_stb_end <= is_stb_end_hi &
       is_stb_end_lo;
177
178  always_ff @(posedge clk_i, negedge arst_i) begin
179    if (~arst_i) begin
180      int_stb = 0;
181    end else begin
182      case (1)
183        is_zero_hold_start: int_stb <= 0;
184        is_stb_end:     int_stb <= 1;
185        default:      int_stb <= int_stb;
186      endcase
187    end
188  end
189
190  always_ff @(posedge clk_i, negedge arst_i) begin
191    if (~arst_i) rdy_o = 0;
192    else rdy_o <= (state == COUNT_STROBE ? 1 : rdy_o);
193  end
194
195  always_ff @(posedge clk_i) begin
```

```
196     err_o <= (state == FIND_EDGE_1 ? 0 : err_o);
197   end
198
199   always_ff @(posedge clk_i) t_end <= (state == WRITE_END ?
        t_cnt : t_end);
200
201   always_ff @(posedge clk_i) t_start <= (state == WRITE_START
        ? t_cnt : t_start);
202
203   // pipelined counter
204
205   logic [T_CNT_WIDTH/2-1 : 0] high_bytes = 0       /* synthesis
        syn_keep=1 syn_preserve=1 syn_ramstyle="registers" */;
206   logic [T_CNT_WIDTH/2-1 : 0] latched_low_bytes = 0   /*
        synthesis syn_keep=1 syn_preserve=1 syn_ramstyle="
        registers" */;
207   logic [T_CNT_WIDTH/2-1 : 0] low_bytes = 0       /* synthesis
        syn_keep=1 syn_preserve=1 syn_ramstyle="registers" */;
208   logic [T_CNT_WIDTH/2-1 : 0] low_bytes_plus_1;
209   logic carry;
210
211   assign {carry, low_bytes_plus_1} = low_bytes + 1;
212
213   //incrementing low bytes
214   always_ff @(posedge clk_i, negedge arst_i)
215     if (~arst_i) low_bytes = 0;
216     else low_bytes <= low_bytes_plus_1;
217
218   //latching low bytes for 1 cycle
219   always_ff @(posedge clk_i, negedge arst_i)
220     if (~arst_i) latched_low_bytes = 0;
221     else latched_low_bytes <= low_bytes;
222
223   logic latched_carry = 0;
224
225   //latching carry
226   always_ff @(posedge clk_i, negedge arst_i)
227     if (~arst_i) latched_carry = 0;
228     else latched_carry <= carry;
229
230   //adding latched carry to high bytes
231   always_ff @(posedge clk_i, negedge arst_i)
232     if (~arst_i) high_bytes = 0;
233     else high_bytes <= high_bytes + latched_carry;
234
```

```
235   //seting t_cnt
236   always_ff @(posedge clk_i, negedge arst_i)
237     if (~arst_i) t_cnt = 0;
238     else t_cnt <= {high_bytes, latched_low_bytes};
239
240 endmodule
```

Листинг П2.2. *two_cycle_32_adder.sv*

```
2  module two_cycle_32_adder (
3      input logic clk_i,
4      input logic [31:0] a_i,
5      input logic [31:0] b_i,
6      input logic valid_i,
7
8      output logic valid_o,
9      output logic [31:0] res_o
10 );
11
12     logic [31:0] a, b;
13
14     always_ff @(posedge clk_i) begin
15         a <= (valid_i ? a_i : a);
16         b <= (valid_i ? b_i : b);
17     end
18
19     enum logic[2:0] {
20         IDLE            = 3'b001,
21         FIRST_STAGE     = 3'b010,
22         SECOND_STAGE    = 3'b100
23     } state = IDLE, next_state;
24
25     always_ff @(posedge clk_i) state <= next_state;
26
27     logic [1:0] valid;
28     integer i;
29     always_ff @(posedge clk_i) begin
30         valid[0] <= valid_i;
31         for (i = 1; i <= 1; i = i +1 ) valid[i] <= valid[i-1];
32         valid_o <= valid[1];
33     end
34
35
36     always_comb begin
```

```
37          next_state = state;
38          case (state)
39              IDLE: if (valid_i) next_state = FIRST_STAGE;
40              FIRST_STAGE: next_state = SECOND_STAGE;
41              SECOND_STAGE: next_state = IDLE;
42          endcase
43      end
44
45      logic carry;
46
47      always_ff @(posedge clk_i) begin
48          case (state)
49              FIRST_STAGE: {carry, res_o[15:0]} <= a[15:0] + b
                    [15:0];
50              default: {carry, res_o[15:0]} <= {carry, res_o
                    [15:0]};
51          endcase
52          case (state)
53              SECOND_STAGE: res_o[31:16] <= a[31:16] + b[31:16]
                    + carry;
54              default: res_o[31:16] <= res_o[31:16];
55          endcase
56      end
57
58  endmodule
```

Листинг П2.3. *sc_fifo.sv*

```
2   module sc_fifo #(
3       parameter LGFLEN = 10,
4       parameter WIDTH = 32
5   ) (
6       input logic     clk_i,
7       input logic     arstn_i,
8
9       input logic [WIDTH-1:0] data_i,
10      input logic             wre_i,
11
12      output logic [WIDTH-1:0] data_o,
13      input  logic             re_i,
14      output logic             n_empty_o
15  );
16
17      logic [WIDTH-1:0] cyc_buf [int'($pow(2, LGFLEN)-1):0];
```

```
18
19      logic [LGFLEN-1:0] r_addr, w_addr;
20
21      assign n_empty_o = (r_addr != w_addr);
22      assign data_o = cyc_buf[r_addr];
23
24      always_ff @(posedge clk_i/*, negedge arstn_i*/) begin :
           w_addr_ff
25          if (~arstn_i) begin
26              w_addr = 0;
27          end else begin
28              if (wre_i) begin
29                  w_addr <= w_addr + 1;
30              end
31          end
32      end
33
34      always_ff @(posedge clk_i/*, negedge arstn_i*/) begin :
           r_addr_ff
35          if (~arstn_i) begin
36              r_addr = 0;
37          end else begin
38              if (re_i & n_empty_o) begin
39                  r_addr <= r_addr + 1;
40              end
41          end
42      end
43
44      always_ff @(posedge clk_i) begin
45          if (wre_i) begin
46              cyc_buf[w_addr] <= data_i;
47          end
48      end
49
50  endmodule
```

Листинг П2.4. *ch_measure_ctl.sv*

```
2   module ch_measure_ctl #(
3     parameter DEFAULT_THRESHOLD_DELTA = 1,
4     parameter DEFAULT_D_CODE_DELTA = 1
5   ) (
6     input logic clk_i,
7     input logic arst_i,
```

```
8
9     //stb request interface
10    output logic stb_req_o,
11    input  logic stb_valid_i,
12
13    //CMP input
14    input logic cmp_out_i,
15
16    //control threshold and delay delta
17    input logic [15:0] threshold_delta_i,
18    input logic [9:0] d_code_delta_i,
19
20    //dac (threshold) output
21    output logic [15:0] threshold_o,
22    output logic threshold_wre_o,
23    input logic threshold_rdy_i,
24
25    //delay line
26    output logic [9:0] d_code_o,
27
28    //ctl
29    input logic run_i,
30
31    //TODO output measured value
32    output logic point_rdy_o
33  );
34
35    enum logic[7:0] {
36      IDLE             = 8'b00000001,
37      SET_THRESHOLD      = 8'b00000010,
38      WAIT_THRESHOLD     = 8'b00000100,
39      REQ_STROBE       = 8'b00001000,
40      WAIT_STROBE      = 8'b00010000,
41      SAVE_CMP_RES     = 8'b00100000,
42      PROCESS_RES      = 8'b01000000,
43      UPDATE_CONF      = 8'b10000000
44    } ctl_state, next_ctl_state;
45
46    enum logic [1:0] {
47      DIR_UP     = 2'b01,
48      DIR_DOWN   = 2'b10
49    } threshold_dir;
50
51    enum logic [1:0] {
52      FIND_DIR   = 2'b01,
```

```
53      FIND_POINT  = 2'b10
54    } point_state, next_point_state;
55
56    logic cur_cmp_out, prev_cmp_out;
57
58    always_comb begin : next_ctl_state_comb
59      if (~run_i) begin
60        next_ctl_state = IDLE;
61      end else begin
62        case (ctl_state)
63          IDLE:          next_ctl_state = SET_THRESHOLD;
64          SET_THRESHOLD:     next_ctl_state = WAIT_THRESHOLD;
65          WAIT_THRESHOLD:    if (threshold_rdy_i) next_ctl_state
              = REQ_STROBE;
66                  else next_ctl_state = ctl_state;
67          REQ_STROBE:      next_ctl_state = WAIT_STROBE;
68          WAIT_STROBE:     if (stb_valid_i) next_ctl_state =
              SAVE_CMP_RES;
69                  else next_ctl_state = ctl_state;
70
71          SAVE_CMP_RES:    next_ctl_state = PROCESS_RES;
72          PROCESS_RES:     next_ctl_state = UPDATE_CONF;
73          UPDATE_CONF:     next_ctl_state = SET_THRESHOLD;
74
75        endcase
76      end
77    end
78
79    always_comb begin : next_point_state_comb
80      if (ctl_state == PROCESS_RES) begin
81        case (point_state) /* synthesis full_case */
82          FIND_POINT: if (cur_cmp_out != prev_cmp_out)
              next_point_state = FIND_DIR;
83                else next_point_state = point_state;
84          FIND_DIR: next_point_state = FIND_POINT;
85        endcase
86      end else begin
87        next_point_state = point_state;
88      end
89    end
90
91    always_ff @(posedge clk_i, negedge arst_i) begin
92      if (~arst_i) begin
93        cur_cmp_out = 1;
94      end else if (ctl_state == SAVE_CMP_RES) begin
```

```
95        cur_cmp_out <= cmp_out_i;
96        prev_cmp_out <= cur_cmp_out;
97      end
98    end
99
100   always_ff @(posedge clk_i, negedge arst_i) begin :
          ctl_state_ff
101     if (~arst_i) begin
102       ctl_state = IDLE;
103     end else begin
104       ctl_state <= next_ctl_state;
105     end
106   end
107
108   logic [15:0] next_threshold;
109   logic [9:0]  next_d_code;
110
111   always_comb begin : next_threshold_comb
112     case (ctl_state)
113       IDLE:       next_threshold = 0;
114       UPDATE_CONF:  if (point_state == FIND_POINT) begin
115                   case (threshold_dir) /* synthesis full_case */
116                     DIR_UP:   next_threshold = threshold_o +
                              threshold_delta_i;
117                     DIR_DOWN: next_threshold = threshold_o -
                              threshold_delta_i;
118                   endcase
119                 end else begin
120                   next_threshold = threshold_o;
121                 end
122       default:    next_threshold = threshold_o;
123     endcase
124   end
125
126   always_comb begin : next_d_code_comb
127     case (ctl_state)
128       IDLE:       next_d_code = 0;
129       UPDATE_CONF:  if (point_state == FIND_DIR) begin
130                     next_d_code = d_code_o + d_code_delta_i;
131                 end else begin
132                   next_d_code = d_code_o;
133                 end
134       default:    next_d_code = d_code_o;
135     endcase
136   end
```

```
137
138   always_ff @(posedge clk_i, negedge arst_i) begin :
          threshold_ff
139     if (~arst_i) begin
140       threshold_o = 0;
141     end else begin
142       threshold_o <= next_threshold;
143     end
144   end
145
146   always_ff @(posedge clk_i, negedge arst_i) begin :
          threshold_wre_ff
147     if (~arst_i) begin
148       threshold_wre_o = 0;
149     end else begin
150       case (ctl_state)
151         SET_THRESHOLD:  threshold_wre_o <= 1;
152         default:        threshold_wre_o <= 0;
153       endcase
154     end
155   end
156
157   always_ff @(posedge clk_i, negedge arst_i) begin :
          stb_req_ff
158     if (~arst_i) begin
159       stb_req_o = 0;
160     end else begin
161       case (ctl_state)
162         REQ_STROBE: stb_req_o <= 1;
163         default:  stb_req_o <= 0;
164       endcase
165     end
166   end
167
168   always_ff @(posedge clk_i, negedge arst_i) begin :
          threshold_dir_ff
169     if (~arst_i) begin
170       threshold_dir = DIR_UP;
171     end else begin
172       if (ctl_state == PROCESS_RES & point_state == FIND_DIR)
              begin
173         case ({cur_cmp_out, prev_cmp_out})
174           2'b10:    threshold_dir <= DIR_DOWN;
175           2'b01:    threshold_dir <= DIR_UP;
```

```
176           default:  threshold_dir <= (threshold_dir ==
                 DIR_DOWN ? DIR_UP : DIR_DOWN);
177         endcase
178       end
179     end
180   end
181
182   always_ff @(posedge clk_i, negedge arst_i) begin :
         point_state_ff
183     if (~arst_i) begin
184       point_state = FIND_POINT;
185     end else begin
186       point_state <= next_point_state;
187     end
188   end
189
190   always_ff @(posedge clk_i, negedge arst_i) begin : d_code_ff
191     if (~arst_i) begin
192       d_code_o = 0;
193     end else begin
194       d_code_o <= next_d_code;
195     end
196   end
197
198   always_ff @(posedge clk_i, negedge arst_i) begin :
         point_rdy_ff
199     if (~arst_i) begin
200       point_rdy_o = 0;
201     end else begin
202       if (ctl_state == PROCESS_RES & cur_cmp_out !=
             prev_cmp_out) point_rdy_o <= 1;
203       else point_rdy_o <= 0;
204     end
205   end
206
207 endmodule
```