Моделирование алгоритмов

Сеть ССИ представляет собой синхронную сеть, соответственно достаточно точно может быть имитирована набором синхронных ФИФО, имеющих различные тактовые частоты для чтения и записи. Для наглядности процесса вставки, удаления служебных символов и работоспособности алгоритма компенсации эффекта проскальзывания, в модельной синхронной сети установлю неточность частоты тактовых генераторов в 10% от номинала в 1МГц.

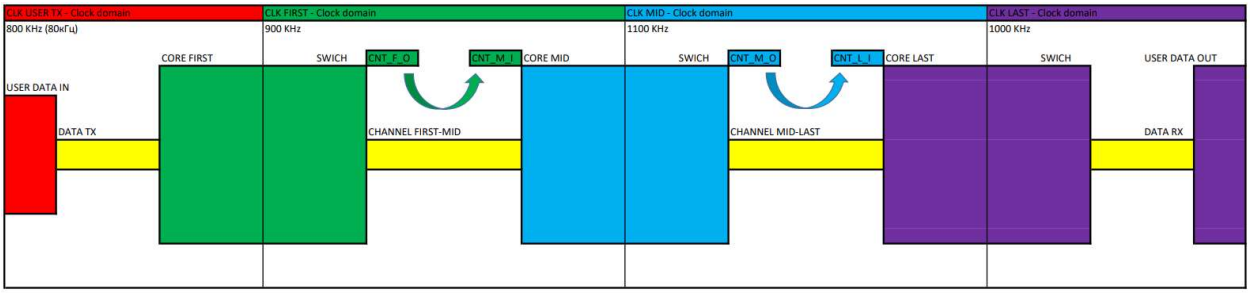
Алгоритм противодействия эффекту проскальзывания в синхронных системах.

1. Эффект проскальзывания, из-за разницы частот тактовых генераторов, приводит к вставке лишних или потере передаваемых символов.
2. Рассмотрим канал передачи данных, где каждый такт данные пишутся и читаются во множестве промежуточных ФИФО. Каждое ФИФО представляет ядро коммутатора, канал передачи данных или ФИФО переноса данных между различными тактовыми доменами.
3. Все коммутаторы тактируются очень близкими (по частоте), но не одинаковыми тактовыми сигналами. Канал передачи данных является плезиохронным.
4. Для устранения потери данных из-за переполнения, необходимо что бы данные всегда поступали на вход ФИФО (для всех промежуточных ФИФО тоже) с меньшей скоростью, чем они будут потом прочитаны. В таком случае ФИФО будет содержать или один элемент данных или пусто.
5. Тогда появляется вопрос: как добиться этого в синхронной системе, где все изменения происходят по фронту тактового импульса.
6. Предлагаю передавать полезные (пользовательские) данные с частотой (скоростью) немного меньшей чем определяется тактовой частотой коммутаторов. Полезная частота (скорость) передачи должна быть меньше, чем частота (скорость), задаваемая тактовым генератором любого из промежуточных генераторов на удвоенную неточность тактирующего генератора. Для современных тактовых генераторов эта величина не превышает 200 ppm в самом худшем случае, соответственно скорость данных должно быть меньше скорости канала на величину: Частота генератора, умноженная на 2*200E-6.
7. Как это сделать, если устройство жестко тактируется от локального генератора и других тактовых частот нет? Для получения эффекта уменьшения скорости будет достаточно добавить к передаваемым данным служебные символы «нет данных», таким образом скорость передачи полезных данных упадет пропорционально числу добавленных служебных символов
8. Тогда возникает вопрос: А когда и в каком количестве их добавлять? Предлагаю добавлять символы «нет данных» в момент чтения из пустого ФИФО, для всех промежуточных ФИФО, кроме самого первого. Для первого ФИФО необходимо или контролировать частоту, с которой в него пишутся данные или писать всегда, но каждые 2500 записей добавлять символ нет данных (для удвоенной ошибки 200 ppm).
9. В передаваемых данных появляется небольшое количество служебных символов «нет данных». В момент приема данного символа из канала, необходимо запрещать запись в ФИФО (отбрасывать принятый служебный символ). Таким образом данный символ исчезает из передаваемой последовательности, в ФИФО остаются только пользовательские данные.

10. Данный механизм создания и удаления служебных символов позволяет гарантировать, что средняя скорость записи всегда будет меньше скорости чтения (с учетом добавленных служебных символов «нет данных») для всех промежуточных ФИФО и это же гарантирует невозможность переполнения буферов, соответственно потери данных по этой причине исключаются.
11. Полезным эффектом работы данного алгоритма является малый и фиксированный размер промежуточных буферов, а значит стабильное время коммутации в среднем равное половине времени на передачу одного символа.

Моделирование синхронного канала передачи данных на FPGA.

Крайне упрощенная модель участка сети ССИ (три промежуточных коммутатора):



- Теоретическая частота задающих генераторов выбрана в 1МГц (1E6 символов в секунду).
- Ошибка установки частоты синхронизации 10% (1МГц +- 100кГц).
- Максимальная скорость передачи данных (без потерь от переполнения) 800к символов в секунду.
- В скобках указан вариант работы, когда данные занимают 10% от максимальной пропускной способности.
- CNT_x_x являются счетчиками переданных и принятых символов. Прошу заметить, что каждая пара счетчиков находится в одном тактовом домене, а значит никаких проблем с из синхронизацией быть не может. Сейчас такой счетчик существует в каждой высокоскоростной сети, но считает он только принятые биты (для корректного формирования передаваемых слов). Проблема синхронизации давно и надежно решена.

Алгоритм разделения единого физического канала на множество виртуальных каналов.

1. Современные системы (оптика в частности) построены на передаче «символов» (битовые последовательности некоторой заданной длинны). На передающей стороне с какой-то постоянной частотой (тактируется локальным генератором) символы подаются в модуль преобразования из параллельного вида в последовательный и далее последовательный поток уходит в передатчик. На приемной стороне сначала в модуле PLL происходит восстановление тактовой частоты, далее преобразование в параллельный вид и запись в ФИФО, разделяющее домены с различной тактовой частотой. Запись в этот буфер происходит с использованием восстановленной частоты локального тактового генератора (PLL), передавшего данные.
2. На передатчике и приемнике есть (создано для работы алгоритма) по одному счетчику, которые инкрементируются каждый раз, когда передается или принимается символ, вне зависимости от типа символа (даже для символа нет данных). И вот именно их (на модели счетчики называются CNT_x_x) необходимо синхронизировать, так что бы в них

содержалось одинаковое число в момент отправки и приема любого символа. Например, если в момент передачи конкретного символа в счетчике число N, то и в момент приема этого символа счетчик на приемной стороне должен содержать именно число N.

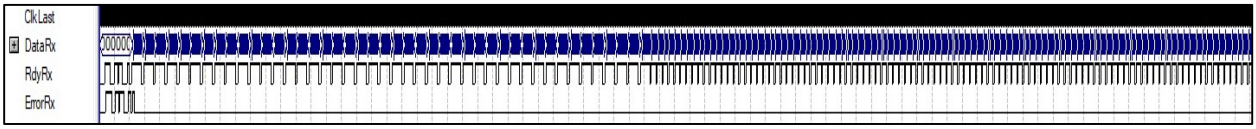
3. Оба счетчика тактируются одинаковой тактовой частотой, только для передатчика это локальный генератор, а для приемника это восстановленная из принимаемой последовательности частота PLL. Получается, что в нормальном состоянии счетчики не будут со временем разбегаться. Еще раз повторюсь: Счетчики на приемной и передающей стороне находятся в одном тактовом домене.
4. Остается выполнить компенсацию расхождения (постоянного смещения). Для этого предлагаю периодически передавать служебный символ «синхронизация», в котором содержатся данные о значении счетчика в момент отправки. При приеме такого символа, необходимо проверять равенство значения в символе «синхронизация» и в счетчике на приемной стороне. В случае неравенства, выполнять некий алгоритм по синхронизации. Например, подождать три подряд ошибки синхронизации и при четвертой подряд изменить значение счетчика на приемной стороне.
5. Когда такой символ передавать: можно выделить определенную полосу, так же, как и для символа «нет данных». Можно на передающей стороне заменять символы «нет данных» на символы «синхронизация», а на приемной стороне производить обратную замену. Вариантов реализации данного алгоритма может быть очень много, главное делать это не реже чем раз в установленное время.
6. Возможные ошибки:
    a. Начальное смещение будет за единицы приняты символов устранено (полезных данных еще нет, только символы «нет данных»).
    b. Сбой счетчика на передающей стороне (вероятность крайне мала и может быть еще сильнее подавлена через дублирование счетчиков), после передачи нескольких символов будет синхронизирована.
    c. Сбой счетчика на приемной стороне, примерно тоже самое что и сбой на передающей стороне.
    d. Ошибка в передачи символа «синхронизация», например в виде неправильного значения счетчика. Достаточно хорошо парируется простым алгоритмом синхронизации, да и в самом простом способе, ошибка просуществует ровно до следующего приема символа «синхронизации».
    e. Сбой PLL на приемной стороне, данная ситуация считается достаточно «тяжелым» событием и время восстановления канала может затянуться на десятки или сотни миллисекунд, разрушительно влияет на любые реально существующие сети и в большинстве случаев разрывает соединение для всех активных сеансов связи.
7. Разрядность счетчиков, соответственно и максимальное число «тайм-слотов» ограничено только размером символа (желательно значение счетчика упаковывать в один символ, но не обязательно). С практической точки зрения, число тайм-слотов можно определять как скорость в битах в секунду деленая на размер символа и равняется числу передаваемых символов в секунду.
8. **Итог:** получается очень простая система синхронизации, если и допускающая сбои, то на очень короткий промежуток времени и с крайне малой вероятностью возникновения такового события, пропорциональной вероятности приема от одного до нескольких дефектных символов «синхронизация» подряд (для типичной вероятности ошибки в оптическом канале 10e-12 очень редко).

Моделирование синхронизации счетчиков в канале передачи данных на FPGA (Quartus 9.0).

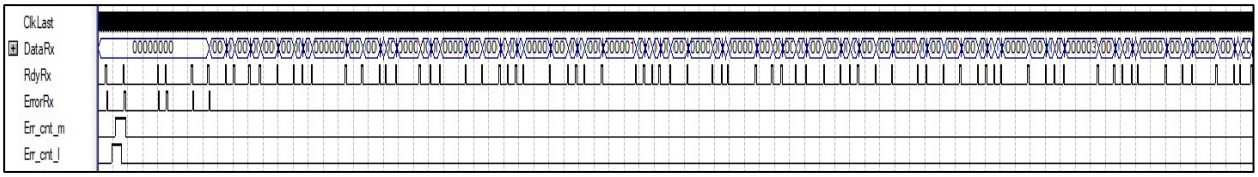
1. Демонстрацию работы алгоритмов сети ССИ буду проводить в варианте полной утилизации физического канала и в варианте с 10% утилизацией.
2. DataRx – принимаемые данные.
3. RdyRx - строб готовности данных.
4. ErrorRx – ошибка в принимаемой последовательности (на начальном этапе генерируется из-за особенностей ФИФО)

Демонстрация отсутствия ошибок передачи (100% загрузка).

1. Передаются данные, генерируемые счетчиком, а значит для проверки достаточно сравнивать каждое следующее слово с предыдущим плюс один.



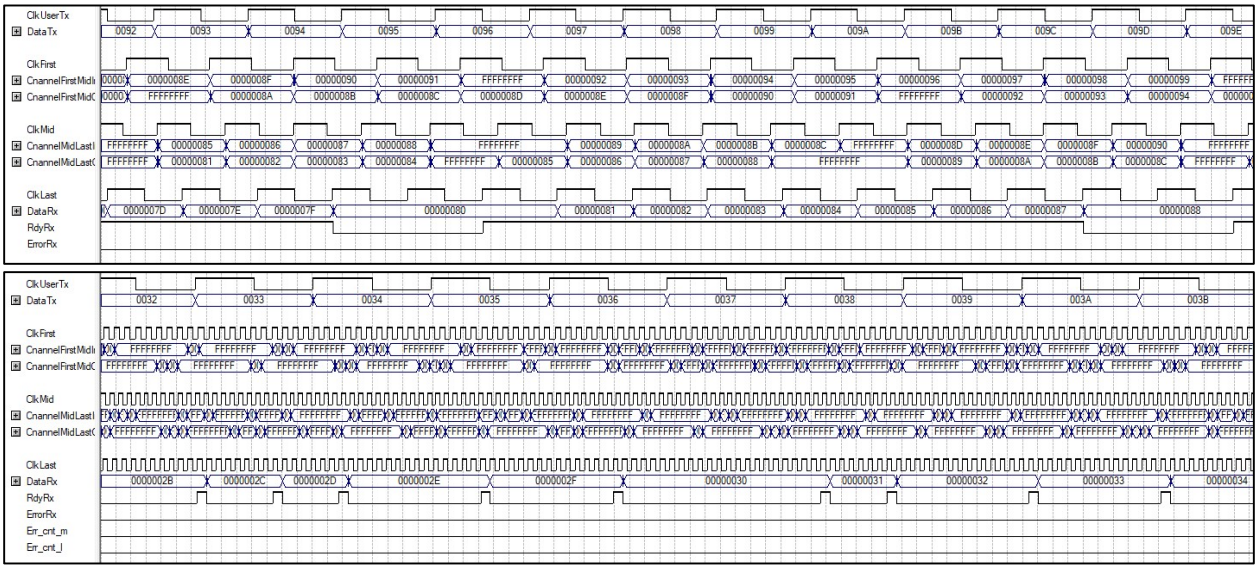
2. Демонстрация отсутствия ошибок передачи (10% загрузка). В данном случае добавлено отсутствие ошибок синхронизации счетчиков слов.



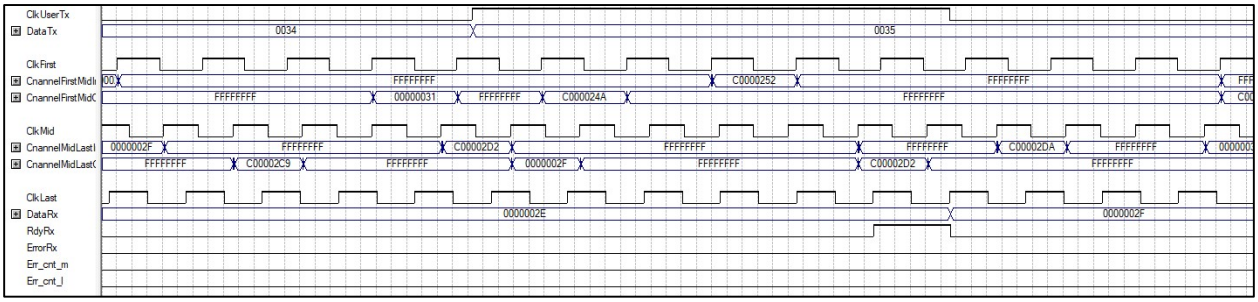
3. Err_cnt_m – ошибка синхронизации счетчиков первого и второго коммутатора.
4. Err_cnt_l – ошибка синхронизации счетчиков второго и третьего коммутатора.
5. Ошибки приема данных и обновления счетчиков присутствуют только в момент начала работы.

Демонстрация структуры (последовательности) передаваемых данных.

1. Первый рисунок для 100% загрузки, второй рисунок для 10% загрузки.



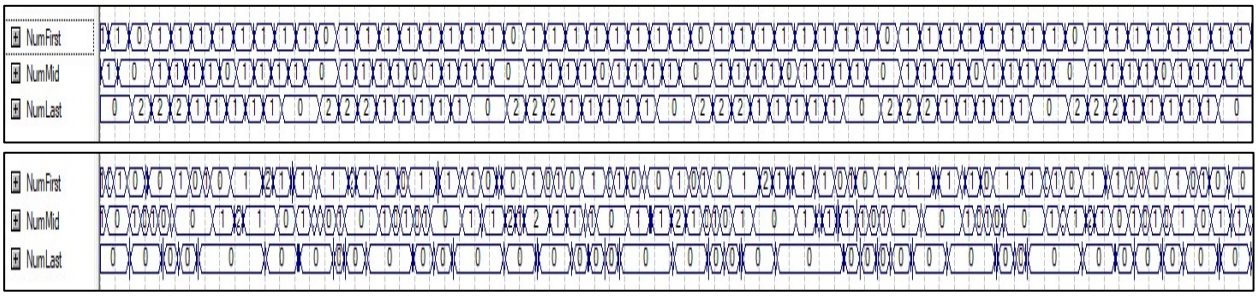
2. Увеличенная картинка для 10% загрузки.



3. DataTx – передаваемые данные
4. Символы начинающиеся на «0» - передаваемые данные
5. Символы начинающиеся на «F» - символ «нет данных» или пусто.
6. Символы начинающиеся на «C» - символ синхронизации счетчиков. Передается каждые восемь символов «нет данных».
7. На первом рисунке число вставок символа «нет данных» минимально и пропорционально ошибке установления частоты синхронизации локального генератора.
8. Для варианта с 10% загрузкой таких символов существенно больше, для передачи используется не вся производительность канала.
9. Разрядность счетчика равняется 16 двоичным разрядам
10. ФИФО ядра коммутатора, содержащее передаваемые данные, откликается не на все значений счетчика (сравнение происходит с зеркальным отображением счетчика).
11. Происходит уменьшение числа обращений, соответственно и уменьшение скорости передачи данных для конкретного виртуального канала.
12. В первом коммутаторе канал передачи данных лежит по адресам от H”0000”- H”1999”.
13. Во втором коммутаторе канал передачи данных лежит по адресам H”ABCD”- H”C566”.
14. Скорость передачи задается размером интервала. В варианте с 10% загрузкой размер интервала вычисляется как 0.1 (10%) умноженное на размер счетчика (16 разрядов - 65536) и равняется 6553 (H”1999”).
15. В модели увеличение скорости учтено при выборе частоты тактирования генератора передаваемых данных (меньше на удвоенную неточность применяемых тактовых генераторов).

Демонстрация числа символов в буферах коммутаторов.

1. Число символов в ФИФО каждого из коммутаторов.

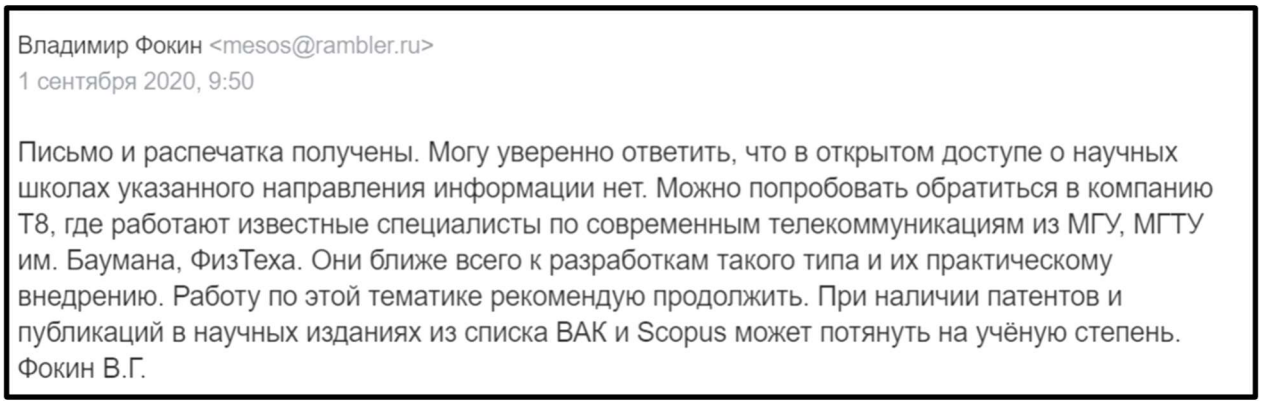


2. Видно, что в ФИФО никогда не содержит больше двух символов.
3. **Внимание:** Необходимо математическими методами доказать, что ни при каких обстоятельствах ФИФО не будет содержать больше двух символов или опровергнуть данное утверждение.
4. Если утверждение будет подтверждено, то коммутатор на современном техпроцессе сможет поддерживать миллионы отдельных виртуальных каналов. Такое количество

излишне, но дает уверенность в том, что проблем с числом одновременных каналов не будет.
5. Максимальное число символов в ФИФО однозначно определяет максимальное время коммутации (прохождение отдельного коммутатора) равное двум периодам передачи символа в виртуальном канале.
6. Время коммутации, привязанное в скорости виртуального канала «справедливо» распределяет задержку между различными каналами. Больше скорость передачи, меньше время коммутации. (Для асинхронных каналов это не так).
7. Время коммутации никак не зависит от степени загруженности разделяемого (физического) канала. Проблема может возникнуть только с выделением необходимой полосы пропускания при создании еще одного канала.

Выводы:

1. Главной проблемой является новизна предлагаемой сети. Средний человек перестает понимать тексты после уровня новизны в 20-30%. Описания сети ССИ, пусть и по примитивной оценке сайта «Антиплагиат», содержит степень новизны около 95%. Это значит, что понимания сути описываемых алгоритмов (именно суммы их взаимодействия), требуются значимые научные компетенции.
2. При отсутствии в нашей стране профильных научных школ и самостоятельных ученых надеяться на таковое понимание очень трудно.
3. Мне не удалось найти ничего похожего в зарубежных изданиях. Максимум что есть из современного это технология FlexM компании Huawei (https://info.support.huawei.com/info-finder/encyclopedia/en/FlexE.html), но это слишком «далеко» для упоминания в разделе литература. Все материалы помечены как свободно распространяемые, при условии указания авторства.
4. Ответ из профильного учебного учреждения СибГУТИ:



Владимир Фокин <mesos@rambler.ru>
1 сентября 2020, 9:50

Письмо и распечатка получены. Могу уверенно ответить, что в открытом доступе о научных школах указанного направления информации нет. Можно попробовать обратиться в компанию Т8, где работают известные специалисты по современным телекоммуникациям из МГУ, МГТУ им. Баумана, ФизТеха. Они ближе всего к разработкам такого типа и их практическому внедрению. Работу по этой тематике рекомендую продолжить. При наличии патентов и публикаций в научных изданиях из списка ВАК и Scopus может потянуть на учёную степень.
Фокин В.Г.

При полном отсутствии научных компетенций, все что можно ожидать от научного и экспертного сообщества, это указание на отсутствие логических ошибок и рекомендации к дополнительному углубленному изучению предложений.

Для внедрения сети ССИ, требуются дополнительные исследования, эксперименты, убедительные доказательства и делать это все должны ученые (или крупные компании), которые имеют авторитет в научном сообществе.

Текст модели на VERILOG.

```
//-------------------------------------------------------------------------------------------------//
//-- Developed by Balyberdin Andrey (Rutel@Mail.ru)
//-------------------------------------------------------------------------------------------------//
//-- All the ideas and algorithms described in this article are the result of my independent and
//-- completely independent intellectual activity. As an author,I allow any person or organization
//-- to freely use, modify, supplement all ideas and algorithms in any type of projects,
//-- with the mandatory indication of my authorship.
//--
//-- Author: Balyberdin Andrey (Rutel@Mail.ru)
//--
//-------------------------------------------------------------------------------------------------//
//--          Attribution 4.0 International (CC BY 4.0)
//-------------------------------------------------------------------------------------------------//
//
module appeal
        (
        // User data
        input   wire                    ClkUserTx,// 800kHz
        output  wire    [15:0]  DataTx,
        // First switch
        input   wire                    ClkFirst,// 900kHz
        output  wire    [3:0]   NumFirst,
        // Intermediate switch
        input   wire                    ClkMid,// 1100kHz
        output  wire    [3:0]   NumMid,
        // Last switch
        input   wire                    ClkLast,// 1000kHz
```

```
        output  wire    [3:0]   NumLast,
        // Channel
        output  wire    [31:0]  CnannelFirstMidInp,
        output  wire    [31:0]  CnannelFirstMidOut,
        output  wire    [31:0]  CnannelMidLastInp,
        output  wire    [31:0]  CnannelMidLastOut,
        //
        output  wire    [31:0]  DataRx,
        output  wire            RdyRx,
        output  wire            ErrorRx,
        output  wire            Err_cnt_m,
        output  wire            Err_cnt_l
        );
//----------------------------------------------------------------------------------------------------//
// DataTx
//----------------------------------------------------------------------------------------------------//
reg     [15:0]  DataUserRg;
assign DataTx[15:0]=DataUserRg[15:0];
//
always @(posedge ClkUserTx)
begin
DataUserRg[15:0]<=DataUserRg[15:0]+16'h01;
end
//----------------------------------------------------------------------------------------------------//
// Core first swich
//----------------------------------------------------------------------------------------------------//
wire                    SwichFirst_rdempty;
wire [31:0]             SwichFirst_q;
//
Fifo    SwichFirst
        (
        .data({16'h0,DataUserRg[15:0]}),
```

```
        .rdclk(ClkFirst),
        .rdreq(!SwichFirst_rdempty & (Cnt_f_o_m[15:0]>=16'h0) & (Cnt_f_o_m[15:0]<=16'h01999)),
        .wrclk(ClkUserTx),
        .wrreq(1'b01),
        .q(SwichFirst_q[31:0]),
        .rdempty(SwichFirst_rdempty),
        .rdusedw(NumFirst[3:0])
        );
//-------------------------------------------------------------------------------------------------//
// Channel First to Mid swich
//-------------------------------------------------------------------------------------------------//
wire                    ChannelFirstMid_rdempty;
wire [31:0]             ChannelFirstMid_q;
reg     [31:0]          ChannelFirstMid_data;
//
Fifo    ChannelFirstMid
        (
        .data(ChannelFirstMid_data[31:0]),
        .rdclk(ClkFirst),
        .rdreq(!ChannelFirstMid_rdempty),
        .wrclk(ClkFirst),
        .wrreq(1'b01),
        .q(ChannelFirstMid_q[31:0]),
        .rdempty(ChannelFirstMid_rdempty),
        .rdusedw()
        );
assign CnannelFirstMidInp[31:0]=ChannelFirstMid_data[31:0];
assign CnannelFirstMidOut[31:0]=ChannelFirstMid_q[31:0];
//-------------------------------------------------------------------------------------------------//
// Core Mid swich
//-------------------------------------------------------------------------------------------------//
wire                            SwichMid_rdempty;
```

```
wire    [31:0]          SwichMid_q;
//
Fifo    SwichMid
        (
        .data(ChannelFirstMid_q[31:0]),
        .rdclk(ClkMid),
        .rdreq(!SwichMid_rdempty & (Cnt_m_o_m[15:0]>=16'h0abcd) &
(Cnt_m_o_m[15:0]<=16'h0c566)),
        .wrclk(ClkFirst),
        .wrreq(ChannelFirstMid_q[31:28]==4'h0 &  (Cnt_m_i_m[15:0]>=16'h0) &
(Cnt_m_i_m[15:0]<=16'h01999)),
        .q(SwichMid_q[31:0]),
        .rdempty(SwichMid_rdempty),
        .rdusedw(NumMid[3:0])
        );
//-------------------------------------------------------------------------------------------------//
// Channel Mid to Last swich
//-------------------------------------------------------------------------------------------------//
wire                    ChannelMidLast_rdempty;
wire    [31:0]  ChannelMidLast_q;
reg     [31:0]  ChannelMidLast_data;
//
Fifo    ChannelMidLast
        (
        .data(ChannelMidLast_data[31:0]),
        .rdclk(ClkMid),
        .rdreq(!ChannelMidLast_rdempty),
        .wrclk(ClkMid),
        .wrreq(1'b01),
        .q(ChannelMidLast_q[31:0]),
        .rdempty(ChannelMidLast_rdempty),
        .rdusedw()
        );
```

```
assign CnannelMidLastInp[31:0]=ChannelMidLast_data[31:0];

assign CnannelMidLastOut[31:0]=ChannelMidLast_q[31:0];

//-----------------------------------------------------------------------------------------------//

// Core Last swich

//-----------------------------------------------------------------------------------------------//

wire                                SwichLast_rdempty;

wire    [31:0]              SwichLast_q;

reg                                         ErrorRx_rg;

reg             [31:0]              DataRx_Old;

//

Fifo    SwichLast

        (

        .data(ChannelMidLast_q[31:0]),

        .rdclk(ClkLast),

        .rdreq(!SwichLast_rdempty),

        .wrclk(ClkMid),

        .wrreq(ChannelMidLast_q[31:28]==4'h0 &  (Cnt_l_i_m[15:0]>=16'h0abcd) &
(Cnt_l_i_m[15:0]<=16'h0c566)),

        .q(SwichLast_q[31:0]),

        .rdempty(SwichLast_rdempty),

        .rdusedw(NumLast[3:0])

        );

//

assign DataRx[31:0]=SwichLast_q[31:0];

assign RdyRx=!SwichLast_rdempty;

assign ErrorRx=ErrorRx_rg;

assign test=ChannelMidLast_q[31:28]==4'h0 &  (Cnt_l_i_m[15:0]>=16'h0) &
(Cnt_l_i_m[15:0]<=16'h01999);

//

always @(*)

begin

if ((SwichFirst_rdempty | Cnt_f_o_m[15:0]<16'h0 | Cnt_f_o_m[15:0]>16'h01999) &
Cnt_f_ff[2:0]!=3'h07)ChannelFirstMid_data[31:0]<=32'hffffffff;
```

```
if ((SwichFirst_rdempty | Cnt_f_o_m[15:0]<16'h0 | Cnt_f_o_m[15:0]>16'h01999) &
Cnt_f_ff[2:0]==3'h07)ChannelFirstMid_data[31:0]<={16'h0c000,Cnt_f_o[15:0]};

if (!SwichFirst_rdempty & Cnt_f_o_m[15:0]>=16'h0 &
Cnt_f_o_m[15:0]<=16'h01999)ChannelFirstMid_data[31:0]<=SwichFirst_q[31:0];

//

if ((SwichMid_rdempty | Cnt_m_o_m[15:0]<16'h0abcd | Cnt_m_o_m[15:0]>16'h0c566) &
Cnt_m_ff[2:0]!=3'h07)ChannelMidLast_data[31:0]<=32'hffffffff;

if ((SwichMid_rdempty | Cnt_m_o_m[15:0]<16'h0abcd | Cnt_m_o_m[15:0]>16'h0c566) &
Cnt_m_ff[2:0]==3'h07)ChannelMidLast_data[31:0]<={16'h0c000,Cnt_m_o[15:0]};

if (!SwichMid_rdempty & Cnt_m_o_m[15:0]>=16'h0abcd &
Cnt_m_o_m[15:0]<=16'h0c566)ChannelMidLast_data[31:0]<=SwichMid_q[31:0];

//

end

// SwichFirst

reg             [15:0]          Cnt_f_o;

wire    [15:0]          Cnt_f_o_m;

reg             [2:0]           Cnt_f_ff;

reg                                     Err_cnt_mid;

//

assign Cnt_f_o_m[0]=Cnt_f_o[15];

assign Cnt_f_o_m[1]=Cnt_f_o[14];

assign Cnt_f_o_m[2]=Cnt_f_o[13];

assign Cnt_f_o_m[3]=Cnt_f_o[12];

assign Cnt_f_o_m[4]=Cnt_f_o[11];

assign Cnt_f_o_m[5]=Cnt_f_o[10];

assign Cnt_f_o_m[6]=Cnt_f_o[9];

assign Cnt_f_o_m[7]=Cnt_f_o[8];

assign Cnt_f_o_m[8]=Cnt_f_o[7];

assign Cnt_f_o_m[9]=Cnt_f_o[6];

assign Cnt_f_o_m[10]=Cnt_f_o[5];

assign Cnt_f_o_m[11]=Cnt_f_o[4];

assign Cnt_f_o_m[12]=Cnt_f_o[3];

assign Cnt_f_o_m[13]=Cnt_f_o[2];

assign Cnt_f_o_m[14]=Cnt_f_o[1];
```

```
assign Cnt_f_o_m[15]=Cnt_f_o[0];
//
assign Cnt_m_i_m[0]=Cnt_m_i[15];
assign Cnt_m_i_m[1]=Cnt_m_i[14];
assign Cnt_m_i_m[2]=Cnt_m_i[13];
assign Cnt_m_i_m[3]=Cnt_m_i[12];
assign Cnt_m_i_m[4]=Cnt_m_i[11];
assign Cnt_m_i_m[5]=Cnt_m_i[10];
assign Cnt_m_i_m[6]=Cnt_m_i[9];
assign Cnt_m_i_m[7]=Cnt_m_i[8];
assign Cnt_m_i_m[8]=Cnt_m_i[7];
assign Cnt_m_i_m[9]=Cnt_m_i[6];
assign Cnt_m_i_m[10]=Cnt_m_i[5];
assign Cnt_m_i_m[11]=Cnt_m_i[4];
assign Cnt_m_i_m[12]=Cnt_m_i[3];
assign Cnt_m_i_m[13]=Cnt_m_i[2];
assign Cnt_m_i_m[14]=Cnt_m_i[1];
assign Cnt_m_i_m[15]=Cnt_m_i[0];
//
assign Cnt_m_o_m[0]=Cnt_m_o[15];
assign Cnt_m_o_m[1]=Cnt_m_o[14];
assign Cnt_m_o_m[2]=Cnt_m_o[13];
assign Cnt_m_o_m[3]=Cnt_m_o[12];
assign Cnt_m_o_m[4]=Cnt_m_o[11];
assign Cnt_m_o_m[5]=Cnt_m_o[10];
assign Cnt_m_o_m[6]=Cnt_m_o[9];
assign Cnt_m_o_m[7]=Cnt_m_o[8];
assign Cnt_m_o_m[8]=Cnt_m_o[7];
assign Cnt_m_o_m[9]=Cnt_m_o[6];
assign Cnt_m_o_m[10]=Cnt_m_o[5];
assign Cnt_m_o_m[11]=Cnt_m_o[4];
assign Cnt_m_o_m[12]=Cnt_m_o[3];
```

```
assign Cnt_m_o_m[13]=Cnt_m_o[2];

assign Cnt_m_o_m[14]=Cnt_m_o[1];

assign Cnt_m_o_m[15]=Cnt_m_o[0];

//

assign Cnt_l_i_m[0]=Cnt_l_i[15];

assign Cnt_l_i_m[1]=Cnt_l_i[14];

assign Cnt_l_i_m[2]=Cnt_l_i[13];

assign Cnt_l_i_m[3]=Cnt_l_i[12];

assign Cnt_l_i_m[4]=Cnt_l_i[11];

assign Cnt_l_i_m[5]=Cnt_l_i[10];

assign Cnt_l_i_m[6]=Cnt_l_i[9];

assign Cnt_l_i_m[7]=Cnt_l_i[8];

assign Cnt_l_i_m[8]=Cnt_l_i[7];

assign Cnt_l_i_m[9]=Cnt_l_i[6];

assign Cnt_l_i_m[10]=Cnt_l_i[5];

assign Cnt_l_i_m[11]=Cnt_l_i[4];

assign Cnt_l_i_m[12]=Cnt_l_i[3];

assign Cnt_l_i_m[13]=Cnt_l_i[2];

assign Cnt_l_i_m[14]=Cnt_l_i[1];

assign Cnt_l_i_m[15]=Cnt_l_i[0];

//

always @(posedge ClkFirst)

begin

Cnt_f_o[15:0]<=Cnt_f_o[15:0]+16'h01;

if(ChannelFirstMid_q[31:28]==4'h0c)Cnt_m_i[15:0]<=ChannelFirstMid_q[15:0]+16'h01; else
Cnt_m_i[15:0]<=Cnt_m_i[15:0]+1;

if(ChannelFirstMid_q[31:28]==4'h0c)Err_cnt_mid<=Cnt_m_i[15:0]!=ChannelFirstMid_q[15:0];

if(SwichFirst_rdempty | Cnt_f_o_m[15:0]<16'h0 |
Cnt_f_o_m[15:0]>16'h01999)Cnt_f_ff[2:0]<=Cnt_f_ff[2:0]+3'h01;

end

// SwichMid

reg             [15:0]          Cnt_m_i;

reg             [15:0]          Cnt_m_o;
```

```
reg             [15:0]          Cnt_l_i;
wire    [15:0]          Cnt_m_i_m;
wire    [15:0]          Cnt_m_o_m;
wire    [15:0]          Cnt_l_i_m;
reg             [2:0]           Cnt_m_ff;
reg                                     Err_cnt_last;
//
always @(posedge ClkMid)
begin
Cnt_m_o[15:0]<=Cnt_m_o[15:0]+16'h01;
if(ChannelMidLast_q[31:28]==4'h0c)Cnt_l_i[15:0]<=ChannelMidLast_q[15:0]+16'h01; else
Cnt_l_i[15:0]<=Cnt_l_i[15:0]+1;
if(ChannelMidLast_q[31:28]==4'h0c)Err_cnt_last<=Cnt_l_i[15:0]!=ChannelMidLast_q[15:0];
if(SwichMid_rdempty | Cnt_m_o_m[15:0]<16'h0abcd |
Cnt_m_o_m[15:0]>16'h0c566)Cnt_m_ff[2:0]<=Cnt_m_ff[2:0]+3'h01;
end
//
assign Err_cnt_m=Err_cnt_mid;
assign Err_cnt_l=Err_cnt_last;
//
always @(posedge ClkLast)
begin
if (!SwichLast_rdempty)DataRx_Old[31:0]<=SwichLast_q[31:0];
ErrorRx_rg<=!SwichLast_rdempty & (DataRx_Old[31:0]+32'h01)!=SwichLast_q[31:0];
end
//
endmodule
```