EMTF reclockers logic

A. Madorsky, UF

# MPCX reclocker

File name in emtf\_core\_mpcx project:  
**emtf\_core\_mpcx.srcs/sources\_1/mpcx/rx/rx\_reclock.sv**

Constraints:  
**mtf7\_core\_impl.xdc**

(find all constraints that have **mpcx\_rx\_i** anywhere in text)

## Source and destination domain parameters:

|  |  |  |
| --- | --- | --- |
| **Parameter or signal** | **Value** | **Name** |
| Source clock, coming from MGT | 80 MHz | **rx\_clk** |
| Source clock synchronous to LHC clock | yes |  |
| Source data bits per source clock period | 38 |  |
| Source data frame # ID method | Header flag = 1 for frame 0 | **rx\_header** |
| Destination clock, FPGA fabric = LHC clock | 40 MHz | **clk40** |
| Destination data bits per dest. clock period | 76 |  |
| Auxiliary “assistance” clock, synchronous to fabric clock, phase-aligned on rising edge | 320 MHz | **clk320** |

## Logic description:

### In Source domain:

1. Combine 38-bit words into 76-bit word using Header flag. The 76-bit word is changing every 25 ns. Name: **inreg\_80**
2. Form 6 copies of the 76-bit data word, each delayed by 6.25 ns relative to the previous. This is done by using rising and falling edges of the source clock. Name: **inreg\_40[5:0]**

### In Assistance clock domain:

1. Create a history shift register for Header flag. This register is necessary to find the optimal **inreg\_40** index for transferring into fabric clock domain. Name: **rx\_header\_r**

### In Destination clock domain:

1. Lock header history register state into local register. Name: **rx\_header\_40**
2. Analyze this history for transitions in **rx\_header\_40** (0 to 1 and 1 to 0) to determine which of the **inreg\_40** copies of the data word is optimal relative to destination clock rising edge.
3. The optimal **inreg\_40** index is recorded in **dest\_sel** signal.
4. Use **dest\_sel** signal to select **inreg\_40** copy which is locked into **fdre\_i** FF. The output word of that FF is the reclocked data word, name: **rx\_data\_76\_o**

## Explanation of why we need six **inreg\_40** copies instead of just four:

Technically, it’s sufficient to have just 4 copies of inreg\_40. However, the clk40 phase may become so unfortunate that the detected dest\_sel is oscillating between 0 and 3 due to inevitable jitter of the relative clock phase. This leads to data words being skipped or duplicated each time it switches between 0 and 3. To avoid that, two more copies are made, and the **dest\_sel** transitions are made as follows:

|  |  |  |
| --- | --- | --- |
| **Detected optimal dest\_sel** | **Same phase as, but with word skip or duplication** | **Allowed Previous dest\_sel** |
| 0 | 4 | 0,1 |
| 1 | 5 | 0,1,2,3 |
| 2 |  | Any |
| 3 |  | Any |
| 4 | 0 | 2,3,4,5 |
| 5 | 1 | 4,5 |

Graphical user interface

Description automatically generated

The idea is to limit transitions between states that lead to word duplication or skips to the situation when the source to destination clock phase is actually changing. When the relative clock phase is stable, no word duplication or skips should be allowed.

## Example:

Let’s say the relative phase is such that we’re at **dest\_sel=0** currently. Now the relative clock phase drifted so that the **dest\_sel** needs to wrap around. The phase identical to **dest\_sel=0** is **dest\_sel=4**, so wrapping around means going to **dest\_sel=3**. Once it switches to **dest\_sel=3**, one data word duplicated. This is inevitable when the relative clock phase is drifting.

Let’s assume that from this point on the relative clock phase becomes stable. The new **dest\_sel=3** is in the middle of the range, so any future jitter leads to switching between **dest\_sel=2,3,4**, which does not lead to word duplication or skips.

The logic on the other end of the range (**dest\_sel=5**) works identically, except that during clock phase drift it wraps around to **dest\_sel=2** and one data word is lost.