GPAC3 data_rec Data Sheet

Content

Table of Contents

Figures

1 Introduction

This component implements a data recorder for up to eight synchronous data streams. Data is recorded to a BRAM and can be read out sequentially over AXI. The data recording is either triggered by an external trigger source or by the data channels leaving/entering a specified band. The amount of pre-trigger data to be recorded is configurable.

1.1 Purpose

The document here describes this very generic firmware developed for all kinds of projects.

1.2 Scope

This document provides a detailed overview of the firmware interface and specifies the user interface.

1.3 Definitions, acronyms, and abbreviations

This document is based on the “IEEE Recommended Practice for Software Requirements Specifications” [1].

BSP	Board Support Package. Collection of software drivers adding functionality to easily accessing the components used in a System On Chip.
FPGA	Field Programmable Gate Array. Programmable logic device.

1.4 References

[1] IEEE Std 830-1998, Recommended Practice for Software Requirements Specifications.

1.5 Overview

Chapter 2 provides an overview and how the firmware is related to the other firmware used.
Chapter 3 contains all the detail information on the interfaces.
Chapter 4 is meant for developers working on the core.

2 Overall description

The data_rec feature is loaded into a System On Chip (SOC) as a AXI slave component. The number of data channels is configurable and the data streams are synchronous to “Clk” (not to s00_axi_clk). The reset is propagated from the AXI clock domain to the data clock domain internally.

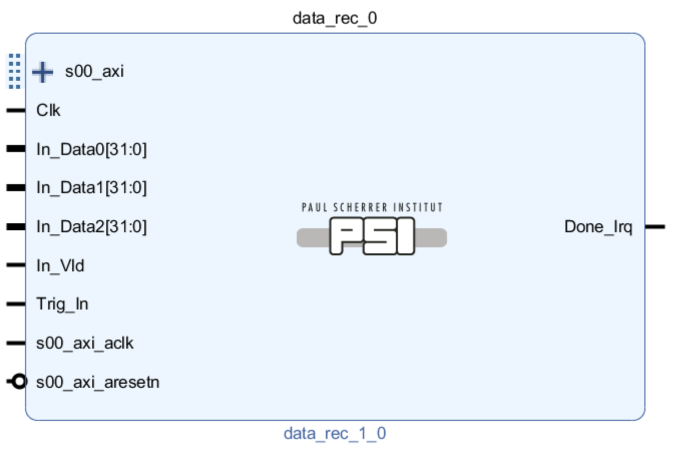
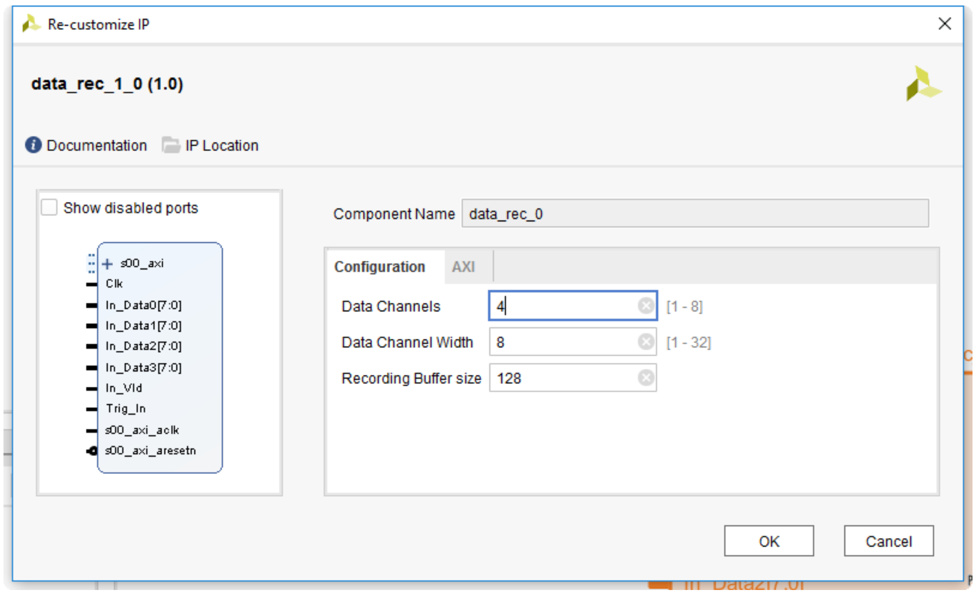


Figure 1: Component Overview

The *Done_Irq* output is pulsed (going high and low again) whenever a recording is completed. The maximum recording rate (and thus the interrupt rate) can be limited.

Width and count of inputs as well as the size of the recording buffer can be chosen in the component GUI. Note that recorder buffer sizes other than 2^x require a bit more logic and may lead to more timing problems. So use a 2^x buffer size whenever possible.



Component Name: data_rec_0

Data Channels	4	[1 - 8]
Data Channel Width	8	[1 - 32]
Recording Buffer size	128	

Figure 2: Component Configuration GUI

3 Request composition

3.1 Configuration

The following short notations for component parameters are used:

Address offset	R/W	Bit	Name	Description
0x00	R	3:0	Status	Status of the recorder: 0 – Idle 1 – Acquiring pre-trigger data 2 – Waiting for trigger 3 – Acquiring post-trigger-data 4 – Recording Done
0x04	W	0	Arm	Write a ‘1’ to this bit to arm the recorder.
	W	16	TrigCntClr	Write a ‘1’ to this bit to clear the trigger counter (*TrigCnt*).
0x08	RW	31:0	PreTrig	This register holds the number of samples that should be recorded before the trigger. Allowed values: 0 … *TotSpl*-1
0x0C	RW	31:0	TotSpl	This register holds the total number of samples (including pre-trigger) that are recorded. At least one sample must be recorded. Allowed values: *PreTrig*+1 … buffer size
0x10	RW	31:0	SelftrigLo	Lower border of the self-trigger range (see description below)
0x14	RW	31:0	SelftrigHi	Upper border of the self-trigger range (see description below)
0x18	RW	7:0	SelftrigChEna	Channel enable for the self-trigger mode (see description below)
	RW	8	SelfTrigExit	Generate a trigger when one of the enabled channels (*SelftrigChEna*) moves from within the defined range (*SelftrigLo … SelftrigHi*) to outside of the range.
	RW	16	SelfTrigExit	Generate a trigger when one of the enabled channels (*SelftrigChEna*) moves from outside of the defined range (*SelftrigLo … SelftrigHi*) to within the range.
0x1C	W	0	SwTrig	If this register is set to one while *TrigEna_Sw* is ‘1’, a trigger is detected. Note that the bit stays ‘1’ (continuous triggers are generated) until reset.
0x20	R	31:0	TrigCnt	Trigger counter. The counter is incremented with each trigger event. Use *TrigCntClr* to clear the counter. Note that only triggers that really lead to a recording are counted. Triggers that are masked by *TrigEnaX* are not counted.
0x24	R	31:0	DoneTime	Time the recorder spent in *Done* state. The counter is reset on a trigger and starts counting when the state machine enters the *Done* state. After quitting the *Done* state by reading the status or arming the recorder, the counter stops but keeps its value.
0x28	RW	0	TrigEna_Ext	Write ‘1’ to this register to enable the external trigger. If this register contains a ‘0’, the external trigger input is ignored.

	RW	2	TrigEna_Sw	Write ‘1’ to this register to enable the SW-trigger. If this register contains a ‘0’, SW-trigger events are ignored.
	RW	1	TrigEna_Self	Write ‘1’ to this register to enable the self-trigger. If this register contains a ‘0’, self-trigger events are ignored.
0x2C	RW	31:0	MinTrigPeriod	Minimum period between two triggers in clock cycles. Is triggers arrive faster, the triggers within a *MinTrigPeriod* are discarded and the first one after the period will lead to the next recording
0x30	RW	7:0	ExtTrigEna	Each external trigger input can be enabled individually. A trigger is generated if a rising edge occurs on any one of the enabled inputs (i.e. the trigger inputs are ORed)

Table 1: Registers

3.1.1 Status State Machine

When the recorder gets armed, it changes its state from *idle* to *pre-trigger*. After the specified number of pre-trigger samples is recorded, the recorder switches to *wait for trigger* and waits until a trigger condition occurs. Trigger conditions that occur during the *pre-trigger* state are ignored.

After a trigger, the remaining samples are recorded. After that, the recorder switches to *idle*. When users read data from the recorder, it should always be in *idle*.

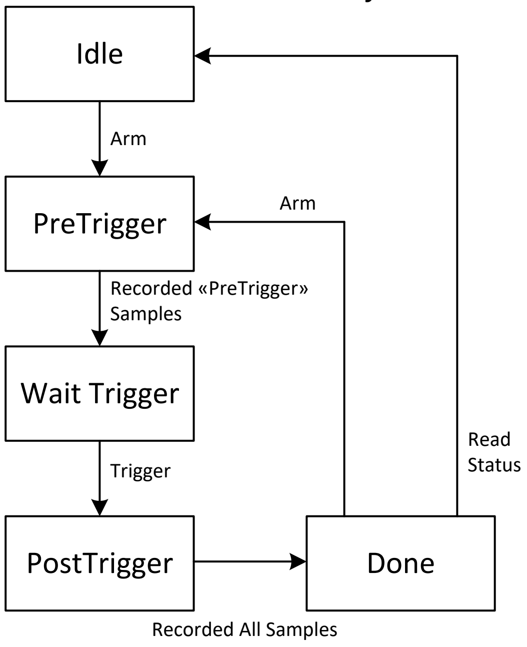


Figure 3: Status state machine

3.1.2 Self-Triggered Mode

The recorder can be operated in self-triggered mode. This means that it works like an oscilloscope in “single-shot” mode: If the recorder is armed, a measurement is triggered if one of the signals moves into or out of a specified range.

The trigger event (trigger on entering or exiting a defined range) as well as the channels that can produce a trigger are configurable.

The upper and lower limit can be used in signed as well as in unsigned notation.

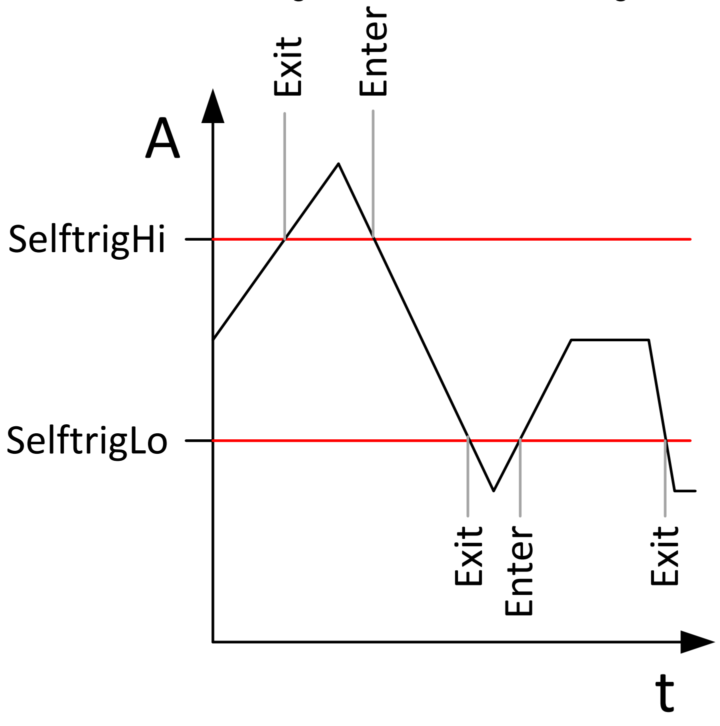


Figure 4: Self-Trigger Conditions

3.2 Data

The address of the data recorded can be calculated using the formula below:

$$Addr_{Data,\#ch} = 0x80 + \#CH \cdot \mathrm{Offs}_{buffer} \cdot 0x04$$

Where:

$\#CH$:	Channel number
Offs_{buffer}:	Offset between buffers (see below)

$$\mathrm{Offs}_{buffer} = 2^{ceil\left(\log_2\left(\mathrm{Size}_{buffer}\right)\right)}$$

Size_{buffer}:	Recording buffer size as entered in the IPI GUI

4 Developer Information

4.1 Tools

To work on this core, the following tools are required:

- Vivado
- Modelsim PE

4.2 Simulation

A selfchecking testbench including a regression test script exists for this core. To run the regression test, follow the steps below:

1. Open ModelSim
2. Navigate to the “sim” directory
3. Execute “source ./run_tcl.tcl”

The regression script automatically compiles all VHDL files, runs all testbenches and checks if any errors occurred.

For interactive work during development, execute the steps below:

1. Execute a regression test as described above
2. For compiling all sources execute “psi::sim::compile –all”
3. Simulations can be ran either manually or by “psi::sim::run_tb –all”

4.3 Packaging

To simplify re-packaging of the IP-Core after changes and avoid trouble with the Vivado GUI, a packaging script was written. To re-package the IP-Core, follow the steps below:

1. Open Vivado
2. In the TCL console, navigate to the “scripts” directory (using “cd <path>”)
3. Execute “source ./package.tcl”