XCorr IP core

Stefan Kull, Roy Seitz, Marco Zollinger

December 31, 2016

1 Overview

This Document describes the functionality and usage of the XCorr IP core. It is supplied as a complete, ready to use IP core but without high level software support. This is due to the project extent which is higher than expected. Low level software access is provided but must be used carefully. High level software support will be added in a future release.

The project Soundloc contains three microphones that deliver a $\Sigma\Delta$-modulated bitstream. These are processed by another IP core SDM_DECIMATOR, that delivers signed microphone data and an interrupt that indicates new values.

This IP core then calculates the cross-correlation between the three microphone signals and findes the Tau where the correlation is maximal. To do this efficiently and in real-time, the correlation is calculated iteratively, using fast block RAM and DSP slices. Detailed information to each stage is provided in the following sections.

2 Parameter description

The IP core can be configured at compile time by several parameters, listed in table 1. The number of stored samples is calculated according to (1). The calculated number of Taus is derived from (2). Tau ranges from $\mathrm{Tau}_{\mathrm{min}}$ to $\mathrm{Tau}_{\mathrm{max}}$, given in (3) and (4)

$$\mathrm{N}_{\mathrm{Sample}} = 2^{\mathrm{D\_SAMPLE\_ADDR\_WIDTH}} - 1 \tag{1}$$

$$\mathrm{N}_{\mathrm{Tau}} = 2^{\mathrm{D\_TAU\_ADDR\_WIDTH}} - 2 \tag{2}$$

$$\mathrm{Tau}_{\mathrm{min}} = -2^{\mathrm{D\_TAU\_ADDR\_WIDTH}-1} + 1 \tag{3}$$

$$\mathrm{Tau}_{\mathrm{max}} = 2^{\mathrm{D\_TAU\_ADDR\_WIDTH}-1} + 1 \tag{4}$$

Parameter	Default	Range	Type	Description
D_WIDTH	16	1…18	integer	Width of incoming microphone data
D_SAMPLE_ADDR_WIDTH	10	7…16	integer	Address width for stored microphone samples
D_TAU_ADDR_WIDTH	4	4…6	integer	Address width for calculated Taus

Table 1: Parameters for the XCorr

3 Cross-correlation

This section describes the calculation of the cross-correlation xcorr01 and xcorr02. The first index indicates the reference microphone, which is fixed to mic0 (microphone 2 on PCB). The correlation is recalculated each time a new value is available. The recalculation takes $\mathrm{N_{Tau}} + 5$ clock cycles. An interrupt is asserted each time the cross-correlation has been recalculated.

3.1 Implementation

The cross-correlation of two discrete signals is defined per (5). Since only $\mathrm{N_{Sample}}$ values are stored and available, the summation simplifies to (6) for positive and (7) for negative Taus.

$$\mathrm{XCorr_{ab}}(\tau) := \sum_{n=-\infty}^{\infty} \mathrm{a}\,[n]\,\mathrm{b}\,[n+\tau] \tag{5}$$

$$= \sum_{n=0}^{\mathrm{N_{Sample}}-\tau-2} \mathrm{a}\,[n]\,\mathrm{b}\,[n+\tau]\ \forall\,\tau \geq 0 \tag{6}$$

$$= \sum_{n=0}^{\mathrm{N_{Sample}}+\tau-2} \mathrm{a}\,[n-\tau]\,\mathrm{b}\,[n]\ \forall\,\tau < 0 \tag{7}$$

$$\tag{8}$$

Each time new data is available, all stored values are shifted by one storage position. Therefore not the whole cross-correlation has to be calculated each time. Only the newest value pair must be added and the oldest pair subtracted, resulting in two MAC-instructions per Tau and correlation.

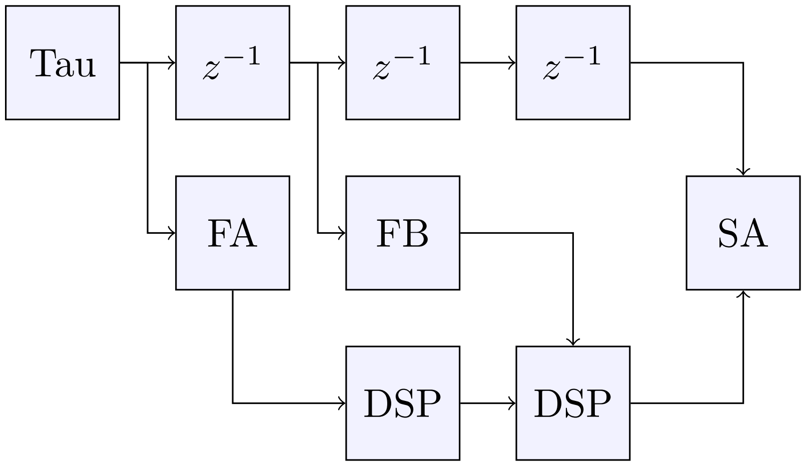

Figure 1: Block diagram of pipelined calculation of cross-correlation

To further improve performance, the calculation is pipelined. Figure 1 shows the data flow during recalculation.

- Block Tau generates the actual Tau that is to be calculated.

- FA denotes fetch on RAM port A. It reads the corresponding cross-correlation and the (old) microphone values that are to multiply and subtract.
- The first DSP performs $P = -(AB) + C$, with A and B connected to the microphone values from FA and C to the cross-correlation value.
- FB fetches the (new) microphone values that are to multiply and add.
- The second DSP calculates $P = (AB) + C$, with the microphone values from FB and the prepared cross-correlation from the first DSP.
- Finally, SA denotes store on port A, which performs a write-back of the new calculated cross-correlation value.

Flip-flops are needed to store and shift the Taus corresponding to the single stages, as each block RAM fetch and store, as well as each MAC instruction needs one clock cycle. Consequently the store command comes always three clock cycles after load. Therefore neither collision nor simultaneous read / write is possible and no collision-detection is implemented.

Initially an internal variable is set to the minimal Value -2^{31}. Each time a new Value is available at the output of the second DSP stage, these two values are compared. If the new Value from the DSP is greater, the corresponding Tau is saved. At the end of the recalculation, this tau is stored in the corresponding AXI register.

4 Register description

Three registers are present. Register 0 is used to clear the internal RAM. This starts an internal state machine that iterates over the whole RAM region. While the internal RAM is being cleared, the whole block is locked and does not provide further functionality. From register 1 and 2, the Taus corresponding to the maximal correlation value can be read. Each register can be accessed directly by their address as described in section 5.

Register Offset	Description
0x0	Write 1 to start clearing of internal RAM Returns 1 if RAM is being cleared
0x4	Tau for xcorr01
0x8	Tau for xcorr02

Table 2: Tau address

5 Driver

The IP core comes along with a driver to allow easy access by software. There are two categories of functions, high and low level. The usage of high level functions is recommended. Care must be taken to not confuse the microphones. On the PCB they are numbered from 1 to 3 but in logic the numbering goes from 0 to 2.

5.1 High level functions

A typical use case with high level functions is as follows:

- call XCORR_Init(baseaddr, return_imm) to initialize the IP core.
  This clears the internal RAM. If return_imm is FALSE, the function will wait until the RAM is cleared, otherwise it will return immediately.
  Typically: XCORR_Init(XCORR_BASEADDR, 0).
- Call XCORR_GetTau(baseaddr, &tau01, &tau02).
  This stores the taus in the provided variables tau01 and tau02.

5.2 Low level functions

The core can also be used by addressing the registers directly. For this purpose two low-level functions are available. These functions are implemented as macros and should only be used if really necessary.

- XCORR_mWriteReg(BaseAddress, RegOffset, Data)
  Write data to the specified Register.
- XCORR_mReadReg(BaseAddress, RegOffset)
  Read the content of the specified register.

6 Test and verification

The core can easily be simulated by using the AXI traffic generator IP from Xilinx and feeding some well known values to the inputs. No additional simulation files are available. Software verification is not supported.

7 Compatibility and License

The core is tested under Vivado version 2016.2 and on Artix7 and Zynq7 FPGA. The core does use hardware specific resources. It is therefore not guaranteed to run on other FPGAs. Since only 4 DSP slices and block RAM is used, it should run on nearly every FPGA with enough block RAM available. However, this is not tested and may require changes to the core hdl files. The core is supplied under no license or copyright but is the intellectual property of the authors.