

Development and Test of the Global Trigger of the BGO-OD Experiment at ELSA

von

Daniel Hahne

Diplomarbeit in Physik

angefertigt am

Physikalischen Institut

vorgelegt der

Mathematisch - Naturwissenschaftlichen Fakultät

der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Juni 2011

Ich versichere, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie die Zitate kenntlich gemacht habe.

Referent:
Koreferent:

Prof. Dr. Hartmut Schmieden
PD Dr. Jörg Pretz

Contents

1	Introduction	1
2	The BGO-OD Experiment	3
2.1	Tagger	5
2.2	Central Detector	7
2.3	Forward Spectrometer	8
2.4	Data Acquisition	11
3	Logics, Electronics and Hardware	13
3.1	Boolean Algebra	13
3.2	Signals and Signal Processing	16
3.2.1	Sampling	16
3.2.1.1	Efficiency	17
3.2.1.2	Signal Resolution	18
3.3	Hardware Standards	20
3.3.1	NIM	20
3.3.2	LVDS	20
3.3.3	VME	21
3.4	FPGA	21
3.4.1	The FPGA board	27
4	Requirements	29
5	Solution and Implementation	31
5.1	FPGA Firmware Modules	34
5.1.1	Overview	34
5.1.2	ScalerTrigger	35
5.1.3	CreateInternalSpill	35
5.1.4	SpillTimeCounter	36
5.1.5	Counter	36
5.1.6	TriggerLogiX	36

5.1.6.1	Input	37
5.1.6.2	TriggerLogic	41
5.1.7	CreateTriggerPulse	47
5.2	b1GlobalTrigger — C++ Interface	50
5.3	TriggerGUI — Graphical User Interface	51
6	Efficiency	55
6.1	Measurement	55
6.2	Results	57
7	Summary and Outlook	59
A	Appendix	61
A.1	Logic Gates	61
A.2	b1GlobalTrigger Class Reference	63
A.2.1	Detailed Description	71
A.2.2	Constructor & Destructor Documentation	72
A.2.2.1	b1GlobalTrigger	72
A.2.3	Member Function Documentation	72
A.2.3.1	GetTriggerLogicVetoRegister	72
A.2.3.2	SetEnableTriggerLogic	72
A.2.3.3	SetTriggerLogicVetoRegister	72
A.3	Example Config File of the Global Trigger	73

Chapter 1

Introduction

Since the beginning of time, men have wanted “to penetrate the power that holds the universe together” [Goe08]. So far leptons and quarks and their antiparticles are the smallest experimentally proofed constituents of matter, whose dynamics are described by the standard model of particle physics. There are six different flavors of quarks known: up, down, charm, strange, top and bottom. Each quark comes with a color charge: red, green or blue. Quarks of different color are attracted to each other. This interaction is called the strong interaction, which is described by quantum chromodynamics (QCD). Its magnitude depends on the coupling constant α_s , which varies with the distance of the quarks. For larger distances the coupling constant increases, which makes it impossible to observe free quarks, because at one point there is enough energy to form new quark-antiquark pairs. This effect is known as *confinement*. For very small distances or high energies, such as they occur in high energy experiments, the coupling constant becomes very small ($\alpha_s \ll 1$). This allows the quarks to move quasi-free (*asymptotic freedom*).

The interaction of quarks in regions of the *asymptotic freedom* is well described within perturbative QCD. However, for distances of about the size of hadrons ($\sim 10^{-15}$ m, $\alpha_s > 1$) perturbative QCD is unable to describe the experimental results. Various models have been developed to describe the excitation spectra of hadrons, but there are still discrepancies between the model’s predicted and the observed excitation spectra. For example, the amount of

predicted excitations of hadrons is much larger than the actual observed. To examine the nature of these deviations new experiments are needed. The BGO-OD experiment, which is currently under development is designed to observe the photoproduction of mesons off nucleons with both charged and neutral final states. To select particular reactions from background and other reactions certain criteria must apply, for example the coincidence of two or more detectors. If these criteria apply, the readout of the detectors must be induced. The electronic logic to implement this is called *trigger*. The goal of this thesis is the development of the global trigger of the BGO-OD experiment.

In chapter 2 the setup of the BGO-OD experiment is described. Chapter 3 gives an introduction into the hardware used by the global trigger. The requirements of the global trigger are presented in chapter 4 and in chapter 5 the solution and implementation of the global trigger is described. Chapter 6 presents an efficiency measurement of the global trigger.

Chapter 2

The BGO-OD Experiment

The BGO-OD (Bismuth Germanate Oxide - Open Dipole) experiment is funded by the German Research Foundation (DFG) within the Transregional Collaborative Research Centre 16 — Subnuclear Structure of Matter. Together with the Crystal Barrel [Uni10a] and the Two Arm Photon Spectrometer [Uni10b] (CBELSA/TAPS) experiment it is centered at the Electron Stretcher Accelerator (ELSA) at the Physics Institute of the Rheinische Friedrich-Wilhelms University of Bonn. An overview of ELSA is shown in figure 2.1. The linear accelerators LINAC1 and LINAC2 produce unpolarized and polarized electrons. Only LINAC2 is currently used for electron production for experiments. The electrons are then injected into the booster synchrotron and accelerated to 1.2 GeV. From the booster synchrotron the electrons are injected into the stretcher ring and accelerated up to 3.5 GeV and extracted into the experimental areas of the CBELSA/TAPS or the BGO-OD experiment. According to the desired beam intensity the duration of extraction is from a few seconds up to several minutes. The extractions are called spill. Typical durations of a spill, used for the CBELSA/TAPS and BGO-OD experiments, are in the order of a few seconds.

The BGO-OD experiment is designed to investigate charged and neutral multi-particle states of meson photoproduction off nucleons. It is a fixed-target

Electron Stretcher Accelerator (ELSA)

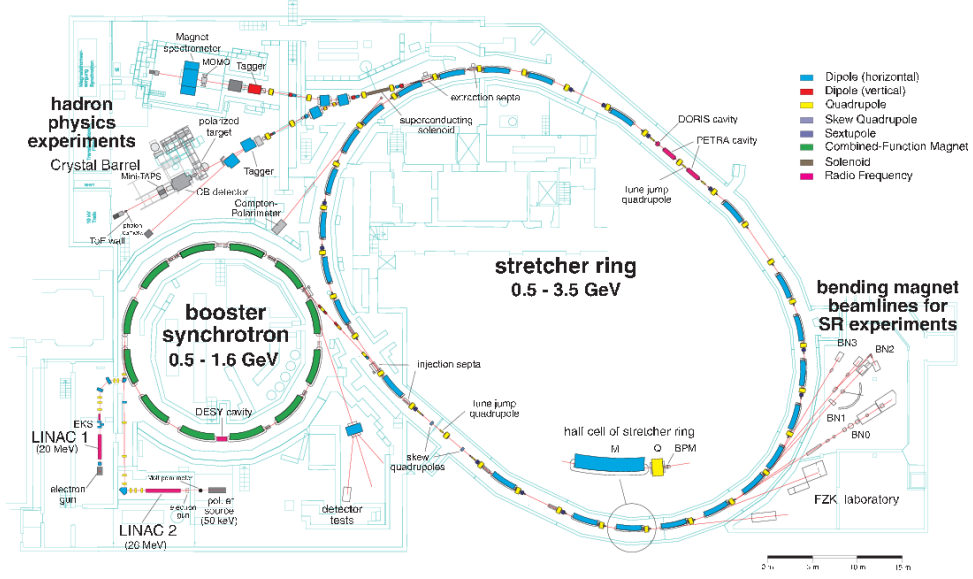


Figure 2.1: Overview of the Electron Stretcher Accelerator (ELSA).

experiment with high acceptance. The setup is shown in figure 2.2. The electron beam from ELSA enters the detector setup from the left. In the goniometer tank electrons undergo bremsstrahlung on a radiator and photons are produced [Bel11]. The tagger [Sie10, Mes] determines the photon energy. The photons hit inside the BGO-Ball on a liquid hydrogen or deuterium target. The BGO-Ball is a crystal calorimeter designed to detect charged states. It covers nearly the whole solid angle. Reactions with a 3.5 GeV incident photon will have a strong forward boost. To identify charged particles in forward direction the momentum and velocity of the particles has to be determined. The momentum is measured by the deflection of the particles in the magnetic field of the open-dipole magnet. Therefore their tracks are determined by the tracking detectors, MOMO [Bel99, Bel07] and SciFi2 [Bös] in front and the drift chambers [Ham08, Sch10] behind the magnet. An aerogel Čerenkov detector will be placed between MOMO and SciFi2, to aid the identification of charged pions and kaons. The time of flight is measured by the TOF [Ram07] wall behind the drift chambers. A gamma intensity monitor (GIM) then measures the effective

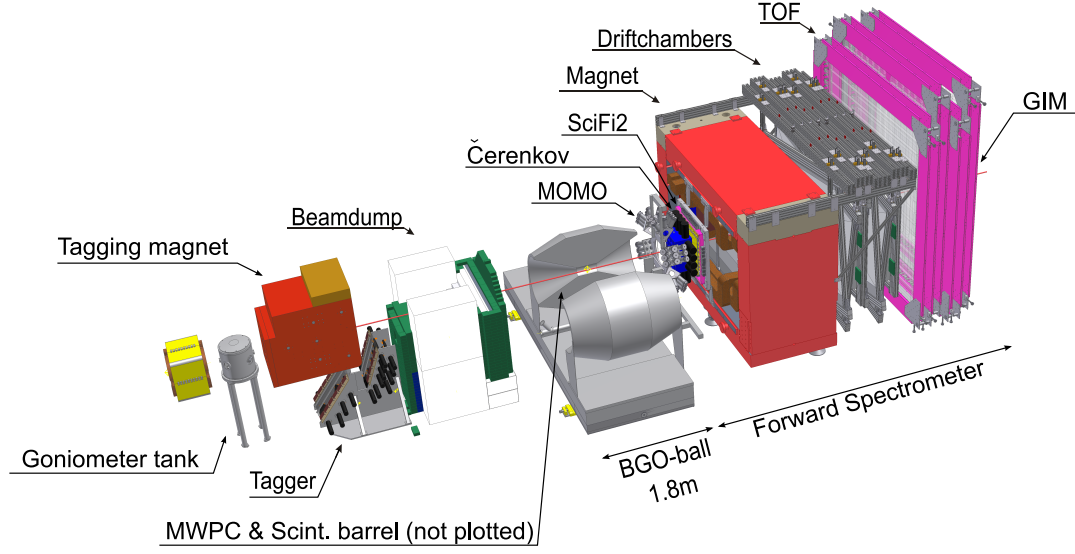


Figure 2.2: Overview of the BGO-OD experiment

photon flux of the experiment[Zim]. The detectors are summarized in table 2.1

Below the components of the experiment will be described in more detail. In addition a description of the data acquisition (DAQ)[Ham] including the global trigger is given.

2.1 Tagger

The electrons from ELSA undergo bremsstrahlung on a radiator inside the goniometer tank and photons are produced. A diamond target is used to produce polarized photons and a copper target to produce non-polarized photons. For further information about the goniometer see [Bel11]. The scattered electrons are momentum analyzed in a magnetic field. The photon energy is determined by the energy difference of the incoming and the scattered electron,

$$E_\gamma = E - E'$$

Name	Function	Detector material	Trigger
Tagger	energy measurement of photon beam	plastic scintillating bars and PM	yes
MWPC	tracking	gas and wires	no
scintillator barrel	particle identification	plastic scintillator and PM	yes
BGO Ball	4π detection of charged and neutral particles	scintillator of anorganic crystals and PM	yes
MOMO	particle tracking before magnet	scintillating fibers and PM	no
Čerenkov	particle identification	aerogel and PM	yes
SciFi2	particle tracking before magnet	scintillating fibers and PM	planned
Drift chambers	particle tracking behind magnet	gas and wires	no
Time-of-Flight wall	Flight time measurement and particle identification	plastic scintillating bars and PM	yes
GIM	photon rate measurement	lead glass and PM	yes

Table 2.1: Overview of the detectors of the BGO-OD experiment.

where E is the energy of the incoming electrons and E' the energy of the scattered electrons. For the ongoing tests the tagger of the former SAPHIR experiment is in use. A completely new tagging system is currently under construction. It consists of an array of overlapping scintillator bars. The bars are arranged such that the electron tracks at least two bars. A trigger signal is created if there is a coincidence of two overlapping bars. Tagger rates up to 50 MHz are possible, due to high rate capability counters.

For more information about the tagger see [Sie10] and [Mes].

2.2 Central Detector

The photon beam hits a liquid hydrogen or liquid deuterium target at the center of the BGO-Ball. The particles produced in the reaction will be tracked by a cylindrical Multi-Wire Proportional Chamber (MWPC) tracking detector, that surrounds the target, but is not installed yet. A barrel consisting of 32 plastic scintillators is placed around the inner tracking detector and identifies charged particles that move in radial direction.

The BGO-Ball is shown in figure 2.3. It consists of 480 $Bi_4Ge_3O_{12}$ crystals. The crystal length is 240 mm, which equates to about 21 radiation lengths. The crystals are arranged *Rugby Ball* shaped in 15 rings of 32 crystals around the target. The placement covers the polar angles from 25° to 155° and the whole azimuth with a precision of 11.3 sr. Together with the crystal length that makes a calorimeter with an excellent energy resolution. The crystals are read out by two types of photomultipliers: Hamamatsu R329 and Hamamatsu R580.

At the moment the BGO-ball creates a trigger signal if the energy sum of the crystals exceeds a specific threshold. It is possible that each ring creates a separate trigger signal.

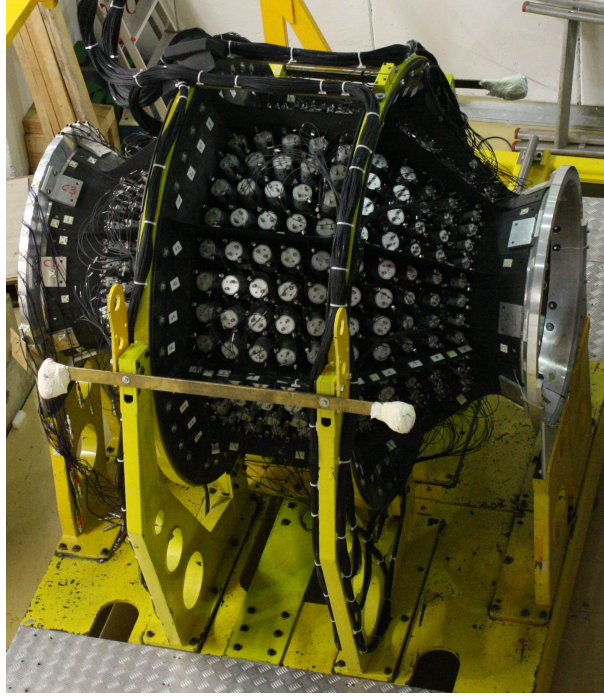


Figure 2.3: The BGO *Rugby Ball* of the BGO-OD experiment.

2.3 Forward Spectrometer

The forward spectrometer is developed to identify charged particles on the basis of the particles momentum and velocity. The momentum of the particles is measured by their deflection in the magnetic field of the open dipole (OD) magnet. The magnet has a maximal magnetic field strength of 0.54 T at a current of 1340 A. The particles are tracked by two tracking detectors, MOMO and SciFi2, in front of the magnet, shown in figure 2.4.

MOMO consists of 672 scintillating fibers of 2.5 mm diameter that are subdivided into three layers, which are rotated by 60° against each other. The active detector area is circularly shaped with a diameter of 44 cm and a spatial resolution of 1.5 mm. A 5 cm hole in the center allows the primary beam to pass through. The scintillating fibers are read out by 16-channel photomultiplier

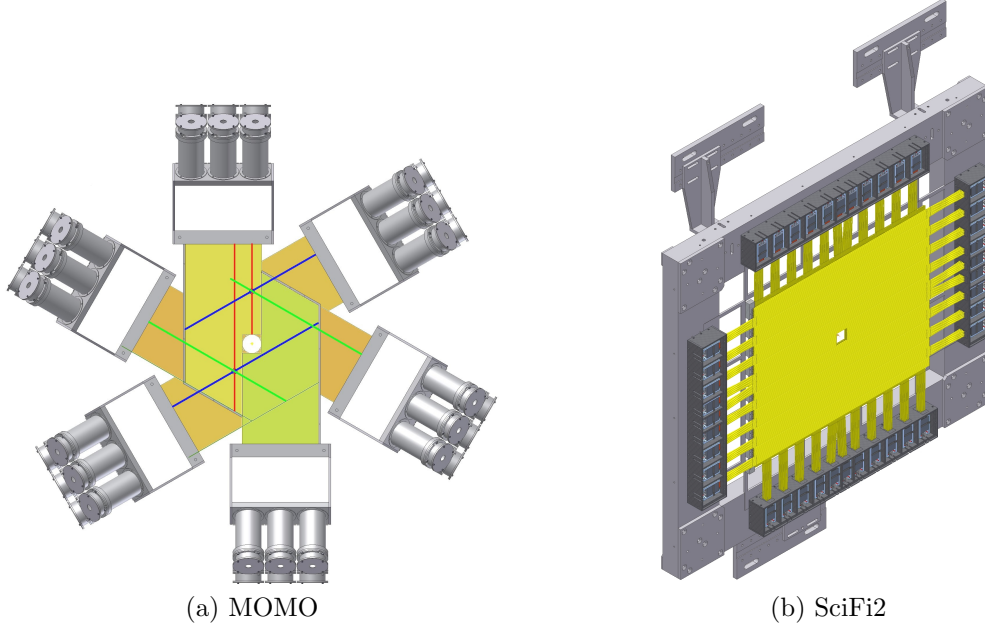


Figure 2.4: The tracking detectors MOMO and SciFi2 of the BGO-OD experiment.

(Hamamatsu R4760). Originally the detector was developed for the MOMO experiment at COSY. There it was used for the detection of charged pion and kaon pairs, which makes it very suitable for the BGO-OD experiment.

The SciFi2 detector[Bös] consists of 640 scintillating fibers with a diameter of 3 mm. The fibers are subdivided into two layers which are rotated by 90° against each other. The active detector area is $66\text{ cm} \times 51\text{ cm}$, which yields an angular coverage of $\pm 10^\circ$ in the horizontal direction and $\pm 8^\circ$ in the vertical direction. A $4\text{ cm} \times 4\text{ cm}$ hole allows the primary beam to pass through. The scintillating fibers are read out by 16-channel photomultipliers (Hamamatsu H6568). SciFi2 has a timing resolution of about 2 ns (FWHM), which makes them ideal to use as a timing trigger.

Behind the magnet the particles are tracked by eight drift chambers. Each drift chamber consists of two layers of sensitive wires. Two of the drift chambers have horizontal and two have vertical orientated wires. The other four drift chambers have vertical wires, but are tilted by $\pm 9^\circ$ around the beam axis, two

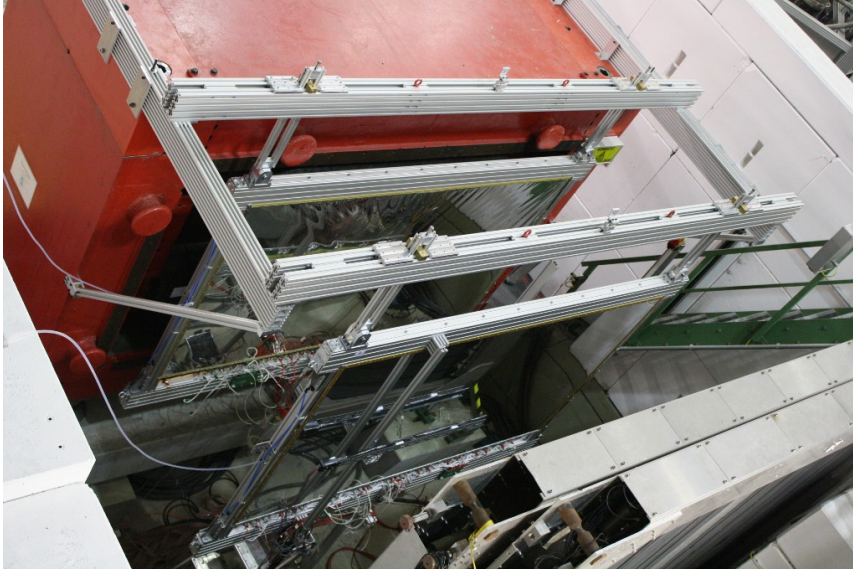


Figure 2.5: The drift chambers of the BGO-OD experiment.

of them clockwise and two counterclockwise. At the beam position, galvanized gold strengthens the wires to make them insensitive to the particles produced by the primary beam. The drift chambers are described in detail in [Ham08, Sch10]

The particle velocity is measured by their time of flight (TOF) from the target to the TOF wall at the end of the spectrometer. The TOF wall consists of four $3\text{ m} \times 3\text{ m}$ walls of scintillating bars. Each wall is divided into 14 bars to distinguish multiple particles at the same time. The bars are read out at both sides. The calculation of the mean time of the detected particle reduces the time jitter of the TOF wall. Like the other detectors it has a central hole for the primary beam to pass through.

A gamma intensity monitor (GIM) is positioned in the beamline, behind the TOF wall. It consists of lead glass, that is designed to absorb the photon beam completely. The lead glass is read out by a photomultiplier. The GIM measures the effectively photon flux of the experiment by counting the photons of the primary beam, that pass the target unaffected. The GIM is developed in the diploma thesis [Zim].

The installation of an aerogel Čerenkov detector (ACD) is planned. It will be placed between MOMO and SciFi2 and improve the discrimination of charged

kaons from charged pions. It is planned to use it as a veto detector. By suppressing the readout of events with pions the amount of events with kaons is increased.

2.4 Data Acquisition

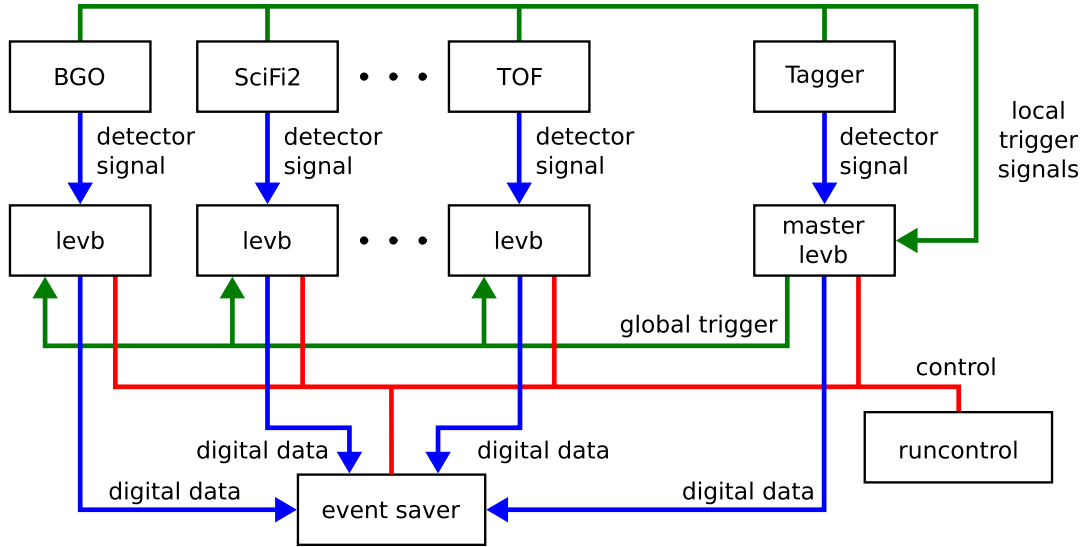


Figure 2.6: Schematic of the DAQ of the BGO-OD experiment. The *runcontrol* initiates the data acquisition. The master levb creates from the local trigger signals of the detectors a global trigger signal, that notifies the levbs to readout their data. The data is then sent to the event saver.

The data acquisition (DAQ) [Ham] of the BGO-OD experiment consists of the *runcontrol*, the event saver (*evs*) and a local event builder (*levb*) for each detector. Figure 2.6 illustrates the setup of the DAQ. The *runcontrol* is a C++ program, that sets the experiment's settings and initiates the data acquisition. The levbs consist of hardware and software components. On the hardware side are readout modules (TDCs, ADCs, scalers) and synchronization boards (sync client). The software part handles the configuration of the modules and transfers the acquired data to the evs. The *master levb* assures the synchronization of the levbs and sends a global trigger signal to the levbs to initiate their data readout.

In principle, the *master levb* is a normal levb, but has a sync master instead of the sync client and also includes the global trigger module that produces the global trigger signals. The sync master is directly connected with the sync clients and provides a 40 MHz clock and the global trigger signal as well as information about the trigger type and the event number. The 40 MHz clock is provided to have a standard clock for all levbs.

The global trigger module inside the master levb receives a local trigger signal from each of the single detectors if the detector has detected a particle. On basis of the local trigger signals the global trigger module decides if a notable event occurred and the experiment's data has to be stored. The global trigger module then sends a global trigger signal to the sync master from where it is distributed it to all levbs. This induces the levbs to read out their detector and sending the data to the event saver. The event saver then assembles the single event data to one data entry and stores it on disk. After the global trigger signal is sent, all further trigger attempts are blocked until all levbs are ready for new data acquisition.

Chapter 3

Logics, Electronics and Hardware

In this chapter the basics of the hardware and electronics are introduced. A short introduction into boolean algebra is given, which is important for understanding signal processing in logic components and especially in the trigger logic.

The central unit of the global trigger is a Xilinx Spartan 3 FPGA on a carrier board. A general introduction of FPGAs is followed by an overview of the Spartan 3 and the board.

3.1 Boolean Algebra

The boolean algebra, often called boolean logic, describes expressions and logical operations of elements that have only the two possible values false and true, also denoted as 0 and 1. Usually the values are referred as truth values.

Boolean logic applies to signal processing and digital logic circuits consisting of logic gates. In signal processing one often speaks of logic low and logic high for 0 and 1.

In a mathematical view boolean algebra is a six-tuple consisting of a set

$$S = (S, \wedge, \vee, \neg, 0, 1)$$

defined through the following axioms[Mon09]:

Let $a, b, c \in S$, then:

commutativity	(1) $a \wedge b = b \wedge a$	(1') $a \vee b = b \vee a$
associativity	(2) $a \wedge (b \wedge c) = (a \wedge b) \wedge c$	(2') $a \vee (b \vee c) = (a \vee b) \vee c$
complements	(3) $a \wedge \neg a = 0$	(3') $a \vee \neg a = 1$
absorption	(4) $a \wedge (a \vee b) = a$	(4') $a \vee (a \wedge b) = a$
distributivity	(5) $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$	(5') $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$

There are two binary operations, the logical conjunction \wedge , also called AND, and the logical disjunction \vee , also called OR and one unary operation, the negation \neg , also called complement or NOT. The behavior of the operations is shown in the truth tables in figure 3.1.

The negation \neg is not a negation in a mathematical sense, but behaves like $\neg x = x + 1 \bmod 2$ or in other words: $\neg 0 = 1$ and $\neg 1 = 0$.

The negation is also often symbolized with an overbar: $\bar{a} = \neg a$.

a	b	$(a \wedge b)$
0	0	0
0	1	0
1	0	0
1	1	1

AND

a	b	$(a \vee b)$
0	0	0
0	1	1
1	0	1
1	1	1

OR

a	$\neg a$
0	1
1	0

NOT

Figure 3.1: Truth table for \wedge , \vee and \neg operators of the boolean algebra.

Combinations of these operations can emulate any logic operation. For example one can build a XOR, that is an exclusive OR, where the output is true if and only if one of its inputs is true, out of NANDs (NOT AND). Let $A, B \in S$:

$$\neg(\neg(\neg(A \wedge B) \wedge A) \wedge \neg(\neg(A \wedge B) \wedge B)) \quad (3.1)$$

In addition to the mathematical notation a symbolic notation is in use. Table A.1.1 in the appendix shows a selection of logic gates and its truth tables of electronic circuits in IEC (International Electrotechnical Commission) 60617-12 norm [IEC94]. Figure 3.2 shows formula 3.1 in symbolic notation. In this case, the basic logic operations are referred as logic symbols or primary (logic) gates.

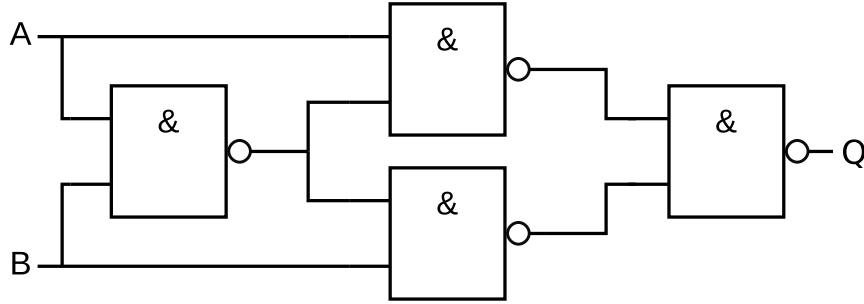


Figure 3.2: A XOR (exclusive OR) gate build of NAND (NOT AND) gates.

Bitwise Operations Bitwise operations are used on each bit of a bit pattern or vector of bits. For example a bitwise AND used on the two bit pattern (a_0, \dots, a_n) and (b_0, \dots, b_n) yields,

$$\begin{pmatrix} a_0 \\ \vdots \\ a_n \end{pmatrix} \wedge \begin{pmatrix} b_0 \\ \vdots \\ b_n \end{pmatrix} = \begin{pmatrix} a_0 \wedge b_0 \\ \vdots \\ a_n \wedge b_n \end{pmatrix}$$

3.2 Signals and Signal Processing

Signals can be either continuous or discrete in respect to their range of values and their time domain. In nature signals are usually continuous, at least on a macroscopic scale, but to digitally process and work numerically with signals it is necessary that they have a finite range of values.

The reduction of time-continuous to time-discrete signals is called sampling. Quantization is the reduction of a value-continuous signal into a value-discrete signal, which is also often called sampling. Coding describes the transformation of a time- and value-discrete signal into another time- and value-discrete signal, for example into a binary signal.

Since the local trigger signals of the single detectors, which are the input signals of the global trigger, come time-continuous, but with two discrete logical values and hence is value-discrete, only sampling is described in detail in section 3.2.1. The FPGA implementation for sampling is described in section 5.1.6.1.

3.2.1 Sampling

Sampling is the process where a time-continuous signal is reduced into a time-discrete signal. The values of digital signals are attached to certain thresholds of analog signals. If the signal amplitude lies in a certain range a dedicated digital value is assigned, see section 3.3 for examples.

A digital system, such as a FPGA, that periodically samples, will sample on clock edges. Therefore the clock edge that is sampled on is called the sampling edge and the system will have a fixed sampling rate. The sampling edge can either be the rising or the falling clock edge or it can be sampled on both clock edges, in which case it is called Double Data Rate (DDR) sampling. At each sampling edge the current signal amplitude is measured. Thus, the sampling resolution is equivalent to the clock frequency or rather to the period of the clock signal.

3.2.1.1 Efficiency

If more than one signal is sampled and the signals time relation is of interest, for example the coincidence of two signals, the signals will not always be sampled at the same clock cycle. Thus, not every coincidence is detected. The efficiency describes the ratio of the detected events E_{det} and the overall events E_{all} ,

$$\text{eff} = \frac{E_{\text{det}}}{E_{\text{all}}}$$

Consider the case, where the coincidence of two signals A and B, is detected and the signals are only sampled on one clock edge. Further, let the coincidence window be of one cycle, so that there must be a coincidence at the same clock cycle. Then, if the time difference of the signals is greater than zero, it is possible that the second signal is not sampled in the first signal's cycle, but the next and therefore no coincidence is detected. The probability for this to happen rises linearly with increasing time difference. The efficiency would be triangular shaped, with a maximum efficiency of one at zero and a total width of two times the clock signal length t_{clk} . The efficiency diagram is shown in figure 3.3a and is symmetrical about $t_{\text{clk}} = 0$. Without loss of generality, let the entries on the left be those where signal A arrives first and the entries on the right, according to this, those where signal B arrives first.

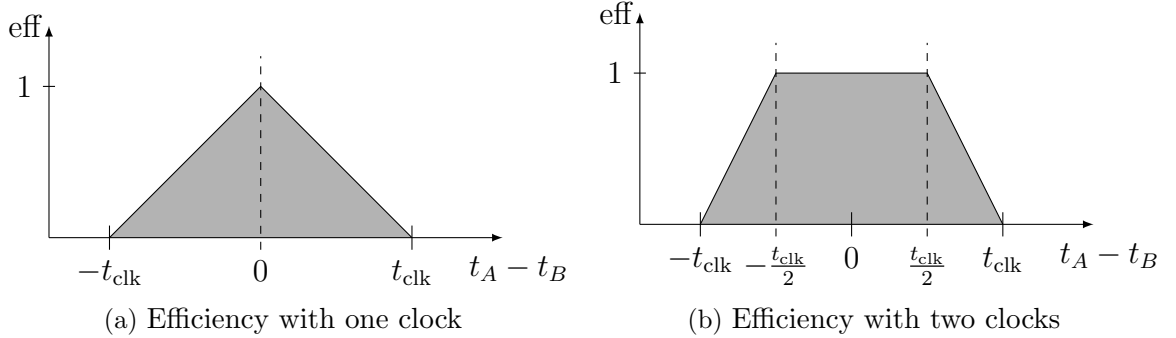


Figure 3.3: Difference in efficiency of one and two input bitmasks

Now, consider the case that DDR sampling is used, which is made by two clocks shifted in phase by 180° and that for each clock an independent coincidence is built. If the second signal is not sampled by the first clock it is still sampled by the other clock up to a time difference of half the clock signal's length, $\frac{t_{\text{clk}}}{2}$ as it is shown in the timing diagram in table 3.1. This results in an efficiency plateau from $-\frac{t_{\text{clk}}}{2}$ to $\frac{t_{\text{clk}}}{2}$ with linearly falling edges and a total width of again two clock signal lengths as it is shown in figure 3.3b.

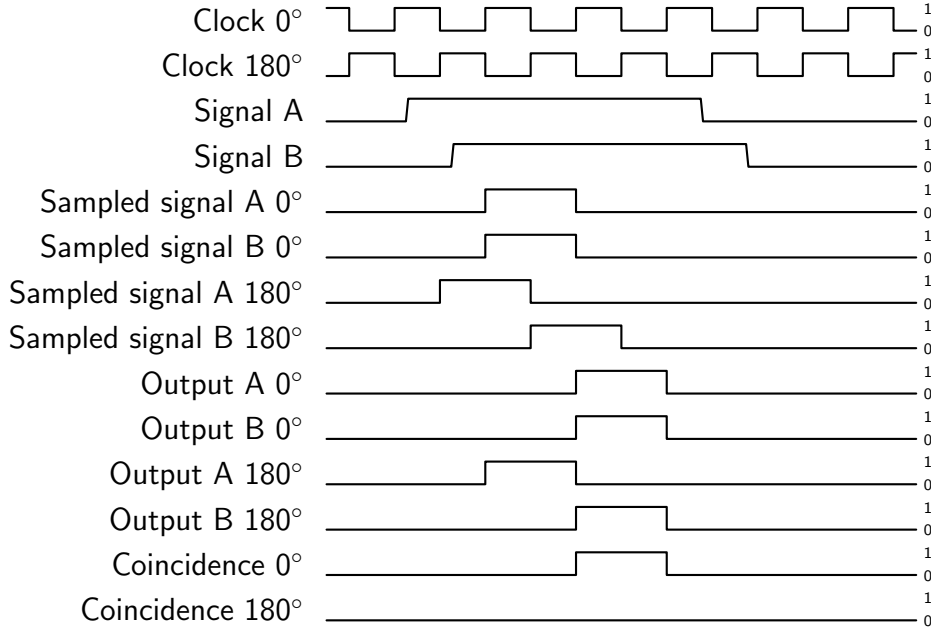


Table 3.1: Timing diagram of the coincidence of two signals sampled with two clocks phase shifted by 180° . The signals are sampled on their leading edge and their length is normalized to one clock cycle. The output is synchronous to the 0° shifted clock.

3.2.1.2 Signal Resolution

For digital signals the determining factor of the time resolution is the accuracy of the measurement of the point in time when a signal passes a threshold. The ideal (binary) digital signal is a rectangular pulse, because it changes its value at exactly one point in time. However, in practice it is impossible to create ideal pulses. A rectangular pulse can be approximated, using a Fourier series consisting of superposed sinusoidal waves of different frequencies. It is illustrated

in figure 3.4.

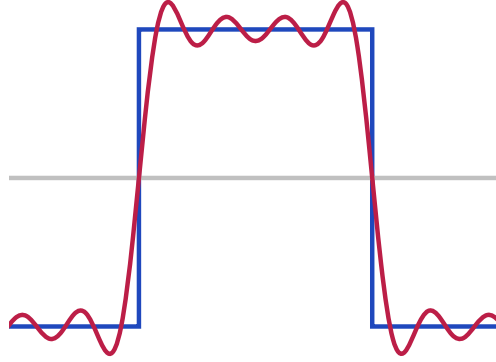


Figure 3.4: Illustration of an ideal rectangular pulse and an approximation with superposed sinusoidal waves.

Due to the longer rise time of the approximated signal the crossing of the threshold moves in time. However, if this applies to each signal it appears as a constant phase shift and does not alter the overall timing.

There are two further effects that affect the signal timing: *walk* and *jitter*. The first effect, is caused by different amplitudes of analog signals. Signals with a higher amplitude have a shorter rise time and hence, are detected earlier. A more detailed description and methods to avoid this effect are specified in [Leo94]. However, the input signals of the global trigger are digitized to LVDS standard with a constant amplitude and would have a constant rise time if there were not the second effect. Since the signal shape is not perfect, due to noise and statistical fluctuations, the coincidence of the signals might be detected at different times. This is called *jitter* and also described in [Leo94].

For the global trigger, *jitter* has a second meaning. It describes a variation of the time relation between two input signals coming from two detectors. For example the time of arrival of the signals of the TOF wall is dependent of the velocity of the detected particles. Thus, the time of detection *jitters* between the arrival of the fastest to the slowest particle.

3.3 Hardware Standards

This section covers signal and bus standards, that are used in the experiment.

3.3.1 NIM

The Nuclear Instrumentation Module (NIM) standard was developed for nuclear and high energy physics in 1961. It defines mechanical and electrical specifications for electronics modules. The chassis where the modules are plugged in is called the NIM crate. The NIM standard crates must supply power for the modules by the backplane. $\pm 12\text{ V}$, $\pm 24\text{ V}$ DC and 110 V AC. must be provided. The newer standard also requires $\pm 6\text{ V}$. The NIM modules have a faceplate height of 22.2 cm , a depth of 24.6 cm and the width must be a multiple of 3.43 cm . [W-I08]

The NIM standard also defines a standard for logic signals [Leo94]: Standard logic (NIM logic) signals have rise times and widths on the order of a few nanoseconds. Hence, it is used for detector systems that need high count rates or fast timing. All input, output and cable impedances are required to be 50Ω . The logic levels are shown in table 3.2.

	Output must deliver	Input must accept
Logic 1	-14 mA to -18 mA	-12 mA to -36 mA
Logic 0	-1 mA to +1 mA	-4 mA to +20 mA

Table 3.2: Standard NIM logic. [Leo94]

3.3.2 LVDS

Another common signal standard is LVDS, which stands for low-voltage differential signaling. It is designed for high speed data transfer. Low-voltage means an absolute voltage of about 1.25 V . The term differential stands for two wires that are used for signal transfer. The signals are defined by a voltage change between the wires of about 350 mV .

3.3.3 VME

VME bus stands for VERSAmodule Eurocard bus. It was originally designed for the Motorola 68000 CPUs in 1981, but is today widely used and standardized by the IEC.

The VME cards come in three sizes based on Eurocards ($1U = 44.45\text{ mm}$). $3U \times 160\text{ mm}$, $6U \times 160\text{ mm}$ and $9U \times 400\text{ mm}$. Each card has a width of 20.3 mm . The cards are plugged into a VME crate, which provides 5 V and $\pm 12\text{ V}$ power supply and provides space for up to twenty-one cards. The connection between VME card and the crate's backplane is made by 96-pin connectors. The number of connectors determines the address and data space and depends on the card size. $3U$ cards have one (J1), $6U$ cards two (J1&J2) and $9U$ cards three (J1&J2&J3) connectors.

The VME bus has 32 address lines and 32 data lines. On the J1 connector are 23 address lines plus one internal address bit and the low order 16 data lines, that is cards with only the J1 connector have 24 address bits and 16 data bits. The J2 connector has the remaining eight address lines and the high order 16 data lines[W-I08]. Thus, cards with the J1 and J2 connectors naturally provide 32 address bits and 32 data bits. 64 bit modes are achieved by multiplexing, that is using the address and data lines twice.

The VME standard supports several protocols. The protocol used by the FPGA board, see 3.4.1, is VME32. It provides 32 bit address space and 32 bit data transfer with a maximum data rate of 40 MB/s .

3.4 FPGA

A field-programmable gate array (FPGA) is the central element of the global trigger of the BGO-OD experiment. The nomenclature of the structures inside of FPGAs depends on their vendor and model. The FPGA of the global trigger is a Spartan 3-4000 XC3S1500 FG676 5C from Xilinx, so their nomenclature is used in this diploma thesis.

FPGAs are integrated (re)programmable circuits. The Spartan 3 has 3,328 configurable logic blocks (CLB), which can be configured and connected to build

any logical circuit from simple counters up to microcontrollers. Further FPGA elements are multiplier, block RAM for data storage and input/output blocks (IOB) that control the data flow of the input/output ports and the internal logic. Four digital clock manager (DCM) provide digital functions to monitor and modulate the clock signals, that is it is possible to have several clocks at a time running with different frequencies or shifted phases. The organization of the elements is illustrated in figure 3.5.

The CLBs are arranged in a checker board shape. Between them are wiring channels with routing wires, where the CLBs connect to. At the connection points are switching matrices, that manage the routing of the CLBs. The topology of a switch matrix is shown in figure 3.6. The points where the wires from the CLBs intersect with the routing wires are called switching points. The switching points consist of programmable switches that, if are set, connect the wires. It is possible that more than one programmable switch is set in a switching point.

The CLBs consist of slices that are connected with the switch matrix. The slices consist of two 4-input LUTs, two flip-flops and arithmetic logic, consisting of two mux, a XOR component for summation and an AND gate for multiply-

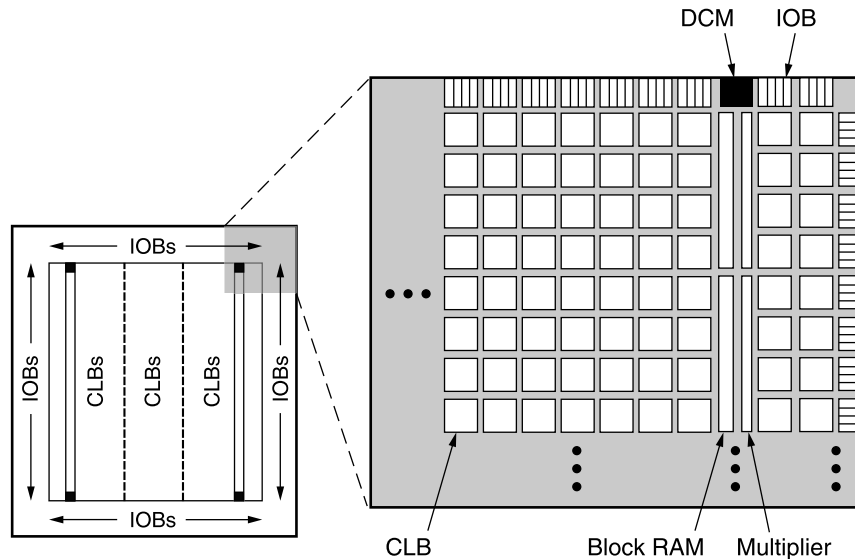


Figure 3.5: Organization of the elements of the Spartan 3. Source: [Xil10a]

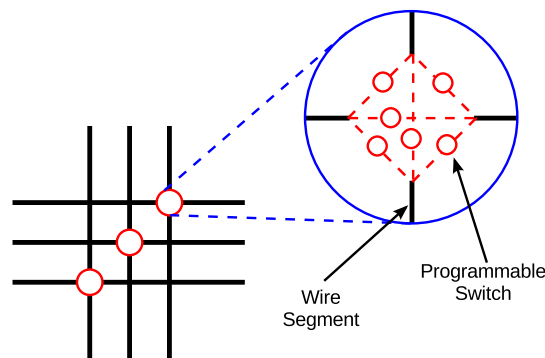


Figure 3.6: Topology of the switch matrix of FPGAs.[Wik06]

ing. Figure 3.7 shows the general design of a CLB in the Spartan 3. The two slices on the right-hand SLICEL support logic only, whereas the two slices on the left-hand SLICEM have additional memory features. Their LUTs can be used as distributed 16×1 bit RAM, where the flip-flops of the LUT are used for storage, or as a 16 bit addressable shift register. Addressable means that the length of the shift register is dynamically adjustable by selecting the flip-flop's designated number as output. A schematic view is shown in figure 3.8. The shift registers have a dedicated output to connect them in series. This makes it possible to build a cascadable shift register of whatever size is needed.

Spartan 3 basic data:

- 13312 slices in 3,328 CLBs.
- 576 kbit Block RAM of 18 kbit blocks for data storage
- Each 18 kbit block is associated with a dedicated multiplier block capable of calculating the product of two 18-bit numbers.
- 221 differential I/O pairs. The data flow between the I/O pins and the internal logic of the FPGA is controlled by I/O blocks (IOBs).
- Four digital clock manager (DCM).

The firmware of a FPGA consists of its configuration and behavior. It is generated from a hardware description language (HDL) file, commonly in VHDL

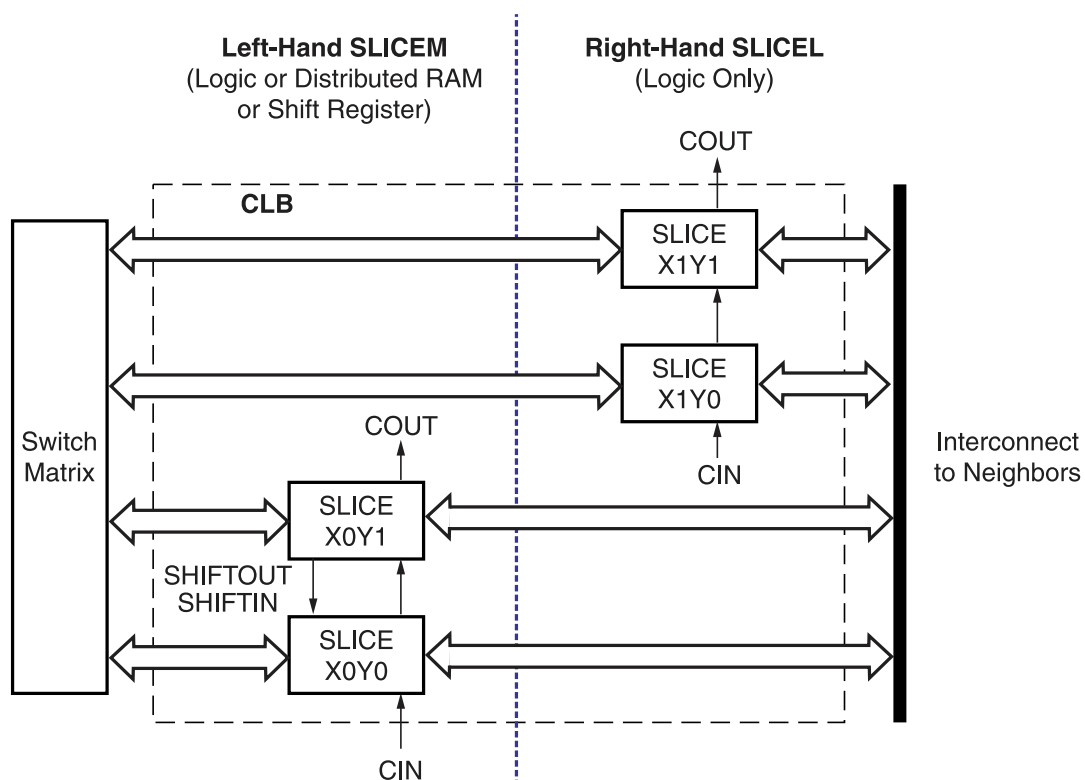


Figure 3.7: General design of a CLB. It consists of four slices, that are connected to the switch matrix. All four slices support logic operations. The left-hand slices have additional memory features.[Xil10b]

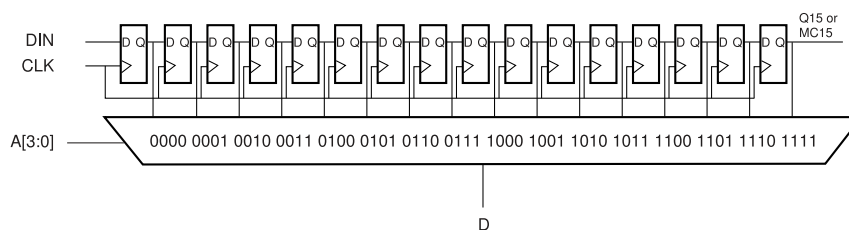


Figure 3.8: Schematic view of LUT configured as addressable shift register. [Xil10b]

or Verilog, or by drawing the schematic design. The global trigger firmware is written in VHDL. VHDL stands for very high speed integrated circuit hardware description language. It describes the behavior of electronic circuits. A VHDL design consists of entities and architectures. Entities describe the interface of the component. They are associated with lists of I/O ports. Ports describe the outwards communication of the entity. The port direction is defined either as in, out or inout. Architectures describe the behavior of the entities. It is possible to include libraries and use previous defined components. The architecture that includes the component only needs information about the I/O ports of the entity not about its behavior. Listing 3.9 gives an example of a VHDL component that sets its output high if forty-two input signals were counted.

The generation of the firmware from the VHDL code is made from the following steps.

Synthesis In the synthesis a netlist is generated that contains the logical design data, that is a list of basic logic elements needed for the design and their connectivity among each other. This netlist can be used to simulate the behavior of the design and to verify its functionality. This simulation does not contain timing information about the routing paths or the switching delays of the logic elements.

Translate In the translation process a new netlist is produced that contains approximate timing information about the switching delays of the logic elements.

Mapping During mapping the netlist is mapped on specific device resources, that is flip-flops, LUTs and other. The output file of the mapping process contains precise information about switching delays, but, since the devices are not routed, has no timing information about the routing paths yet.

Place and Route During place and route (PAR) the device resources are located and interconnected. In a synchronous design the signal propagation time must not exceed a maximum value, that is the output signals of the device

```
-- import std_logic from the IEEE library
library IEEE;
use IEEE.std_logic_1164.all;

-- import counter library
library CNT;
use CNT.8bitcounter;

-- this is the entity
entity InCounter is
    port (
        IN : in std_logic;
        OUT: out std_logic);
end InCounter;

architecture RTL of InCounter is

    -- define 8bit variable
    signal cnt : std_logic_vector(7 downto 0);

begin

    --import 8 counter
    -- map the input signal IN, and the variable cnt
    8bitcounter(
        port map
        (
            CNT_IN => IN,
            VALUE => count
        );

    -- set output high if counter value greater than 42
    OUT <= '1' when count >= 42 else '0';

end RTL;
```

Figure 3.9: VHDL code example of a component that sets its output high if forty-two or more input signals were counted.

resources must be available for the target inputs at the next clock cycle. The maximum signal propagation time between the device resources is defined by constraints. This can either be the clock frequency that the resources are clocked with or user defined constraints.

3.4.1 The FPGA board

The FPGA is mounted on a carrier board or FPGA board ELB-VFB2 from ELB (ELB-Elektroniklaboratorien Bonn UG) shown in figure 3.10.

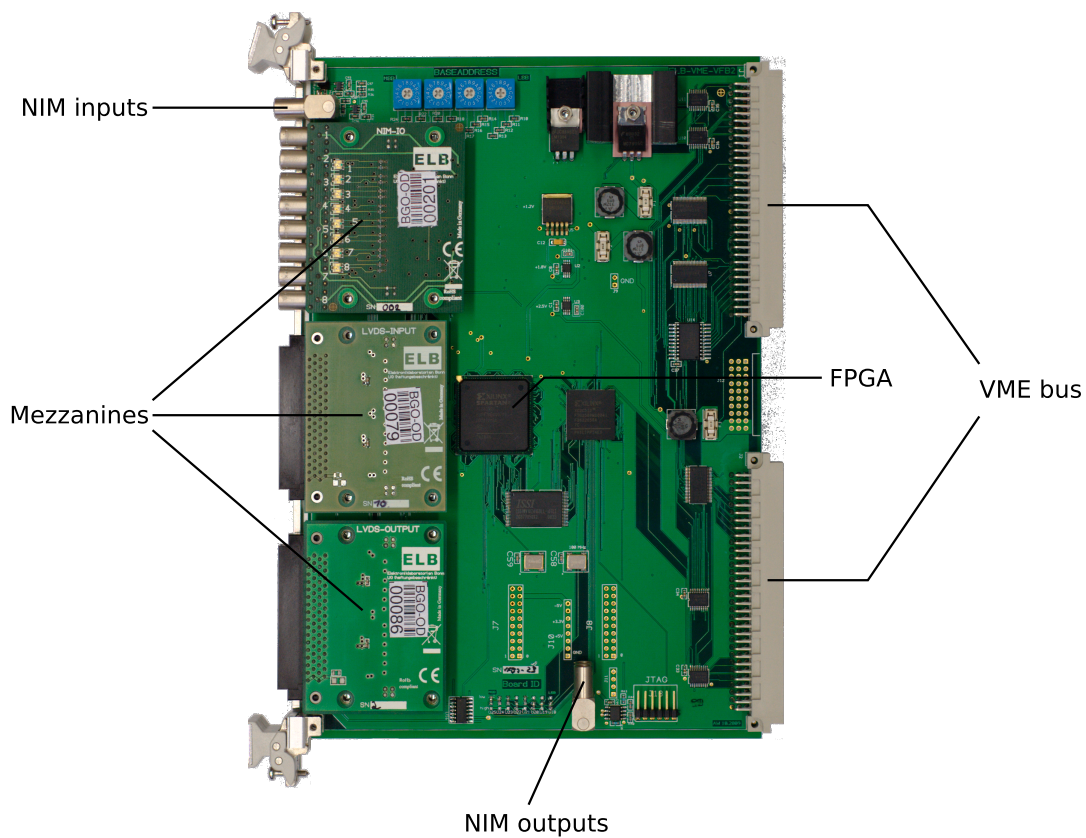


Figure 3.10: Picture of a FPGA board ELB-VFB2. It is attached with three mezzanines that provide additional in- and output ports.

The following information is based on the board's datasheet[ELB11]. The FPGA board has VME standard dimensions. It has two NIM inputs and two NIM outputs. Optionally it is possible to attach it with up to three mezzanine

cards or short mezzanines. The mezzanines provide miscellaneous input/output (I/O) extensions. For example 32 bit width LVDS input and output, 8 bit width NIM input, output and input/output. Furthermore the FPGA board implements a VME interface to program the FPGA and provides run time access to the FPGA.

The VME interface uses a VMEbus module with 32 bit address space and 32 bit data transfer to connect to the FPGA. The upper 16 bit of the address space, the *baseaddress*, are used to identify the boards. The *baseaddress* has a range from 0x00010000 to 0xFFFF0000 and can be selected by four hardware switches on the board. The lower 16 bit address the register inside the FPGA, where each address maps to 8 bit or a byte. Thus, the theoretical amount of registers is 16384.

Chapter 4

Requirements

The global trigger shall be a flexible and easily adjustable module. The requirements are described here in detail:

- A FPGA board with a Spartan 3 from ELB shall be used as the global trigger module. There are no restrictions in the choice of mezzanines.
- The design has to be completely synchronous.
- The global trigger settings have to be adjustable without firmware change.
- The global trigger has to deal with four different kinds of trigger types.
 1. The data trigger: The incoming signals of the local detectors are compared with several preset trigger conditions. If they match a data trigger signal has to be created and sent to the levbs to initiate the data acquisition. For each signal it has to be possible to veto one or more trigger conditions.
 2. Spill start and Spill stop trigger: The electron beam is not continuous, but has times with (spill) and times without electrons. If the electron beam is injected into the experimental area a *spill start* signal is send from ELSA to the global trigger. Respectively, if the injection stops a *spill stop* signal is send. That notifies the run control if the extraction of electrons into the experimental area has begun or stopped. The duration without beam can be used to change run

time settings e.g. the orientation of the goniometer. Thus, the global trigger has to create spill start and spill stop trigger signals to notify the levbs. In case that no spill signals are available, because for example ELSA is not running, internal spill signals have to be created to guarantee a normal course of operations.

3. The scaler trigger: All levbs have to be triggered frequently to read-out their scalers. Therefore a trigger signal with an adjustable frequency has to be created.

- If a trigger signal is created the global trigger has to block further trigger attempts until it is unlocked by the master levb. If one of the spill triggers or the scaler trigger tries to trigger, while the lock is still set, the attempts has to be remembered and postponed until the global trigger is unlocked.
- One of the standard NIM outputs of the FPGA board of the global trigger module must provide a 40 MHz clock as clock source for the levbs.
- The global trigger signal has to be sent over the other standard NIM output of the FPGA board. Behind the global trigger signal the trigger type and the event number must follow as serial data. The global trigger signal has to be sent as soon as possible, but the trigger type and the event number data must be sent synchronously to the 40 MHz clock.
- The input signals have to connect through to outputs of the FPGA to be read out by a TDC.
- For each local trigger signal there must exist an input channel of the global trigger.
- Each input requires an internal adjustable delay to reduce external cable length and to simplify the adjustment of the mean time of the incoming signals.
- The global trigger has to compensate the time jitter of the local trigger signals.

Chapter 5

Solution and Implementation

The development of the global trigger includes the development of the firmware of the FPGA and a C++ interface, called `b1GlobalTrigger`, to access and configure the global trigger. The configuration is stored in a XML file that can be easily generated with a graphical user interface (GUI), called `TriggerGUI`.

The development of the global trigger firmware, the C++ interface and the GUI are part of my diploma thesis. I also contributed on the modules *trigger_link_input* and *trigger_link_output*, that manage the communication between the global trigger module and the sync master.

The central element of the global trigger is the firmware of the FPGA on the FPGA board from ELB with a Spartan 3 from Xilinx, which is described earlier in section 3.4.1. The FPGA board is mounted with two mezzanines. One LVDS input and one LVDS output. Both have 32 channels. On the input mezzanine 24 channels are designated as inputs for the local trigger signals from the single detectors of the experiment. The incoming detector signals are split, that is the signals are doubled, with one part of the signal directly forwarded to the LVDS output mezzanine, where it is output to the TDC. The other part is routed to the trigger logic, where the signals are checked for equality with five adjustable logic combinations, the trigger conditions. If the local trigger signals equals one of the trigger conditions a data trigger signal is created. Since there are only 24 input channels used, there are eight free channels on the LVDS output

mezzanine. The upper five channels show the output of the five trigger conditions. Thus, on the TDC the local trigger signals and the trigger conditions that caused the global trigger signal are shown. It is possible that more than one trigger condition matches the incoming signals and is shown in the TDC spectrum.

Besides the data trigger the global trigger board handles three further trigger types. A spill start, a spill stop and a scaler trigger. The spill trigger events are created when the FPGA receives spill start or spill stop signals from ELSA. These signals are received on the standard NIM inputs of the FPGA board. If no spill signals from ELSA are available, they can be created with freely adjustable frequency and time period by the global trigger module. The scaler trigger is created internally by the global trigger module and triggers with an adjustable frequency (standard 20 Hz) to latch the internal scalers and to inform the levbs to read out their scalers, if they have any. The two standard NIM outputs of the FPGA board are used to provide a 40 MHz clock and to output the global trigger signal. In the remainder of the thesis, the FPGA board with the global trigger firmware will be referred to as the global trigger module. The global trigger module is part of the master levb introduced in section 2.4 on page 11. An overview of the master levb is given in figure 5.1.

The master levb consists of the global trigger module, a TDC and the sync master. The master levb loads the firmware into the FPGA and configures its behavior. Each of the master levb's modules is accessed by a C++ interface. The access to the global trigger module is provided by the C++ interface `b1GlobalTrigger`. It provides functions to program the FPGA as well as read and write settings. Usually the settings are read from a configuration file and written to the global trigger module at startup, but it is also possible to set them independently and at any time by using the functions from the interface. The configuration file can be generated by the program `TriggerGUI`.

The components of the global trigger module are clocked with a frequency of 200 MHz, which is generated from the 100 MHz clock source of the FPGA board. The global trigger module provides a 40 MHz clock to the levbs. The clock signal is sent to the sync master and from there distributed to the levbs.

The main task of the global trigger module is to create the global trigger

signals, which are then sent to the sync master, from where they are forwarded to the levbs. The global trigger signal is attached with information about the trigger type and the event number. Afterwards a global busy is set and sent to the sync master until it is reset by the sync master. As long as the global trigger module is busy any further data trigger attempts are blocked. However, the trigger attempts of the scaler trigger and the spill triggers are stored and sent immediately after the global busy is reset. The trigger attempts of each trigger type will only be stored once, that is if a trigger attempt is made by a trigger type that is already stored only one global trigger signal is created for that type.

If the levbs receive the global trigger signal they will start to read out their associated detector and send the data to the event saver, where it is stored for the analysis. As long as a levb is busy with the readout it sends a local busy signal to the sync master of the master levb. After all local busy signals turned off the master levb resets the global busy and the global trigger module is ready for the next trigger attempt.

5.1 FPGA Firmware Modules

This section covers the firmware of the global trigger. The complete firmware is written in VHDL and is stored in the DAQ repository of the work group of Prof. Dr. H. Schmieden.

5.1.1 Overview

Figure 5.2 gives an overview of the components of the FPGA firmware. The basic frequency of the global trigger module is 200 MHz, and each component that is described below is clocked with this frequency if not said otherwise. All components are connected via a VME bus that allow their settings to be adjusted. The component *TriggerLogiX* gets the local trigger signals and if they equal dedicated trigger conditions the data trigger signal is created. The component *ScalerTrigger* creates the scaler trigger signal using a predefined frequency . If no external spill signal is available the component *CreateInternalSpill* creates

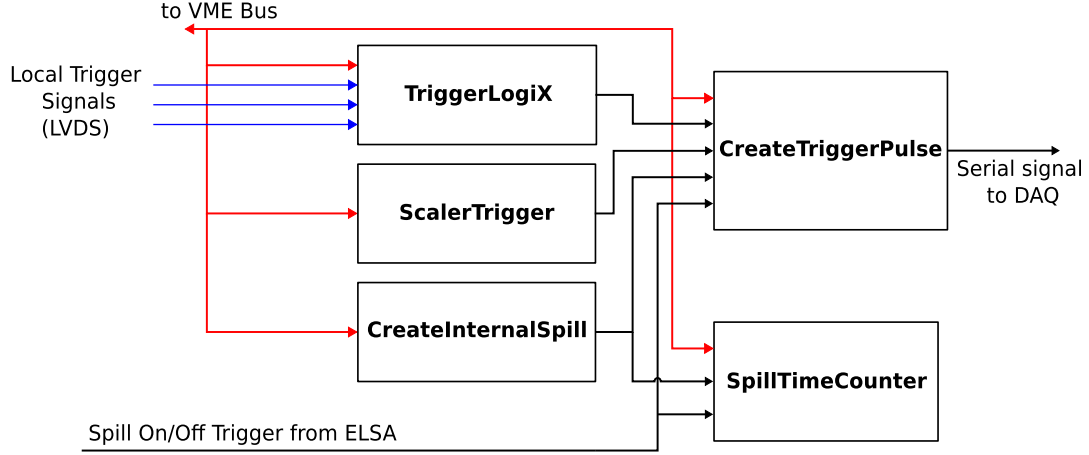


Figure 5.2: Overview of the firmware modules.

the spill start and spill stop trigger signals. All trigger signals are sent to the component *CreateTriggerPulse*. There the global trigger signal is created and the information about the trigger type and the event number is attached and serially sent to the DAQ. The time since the last spill start trigger and its number is measured in the component *SpillTimeCounter*.

5.1.2 ScalerTrigger

The component *ScalerTrigger* creates trigger signal with a fixed frequency. Usually a frequency of 20 Hz is used. To achieve this a counter is driven by a 1 MHz clock whose actual value is compared with a preset value. If the comparison is true a trigger signal is sent to the *CreateTriggerPulse* module and the counter is reset. It is possible to turn the scaler trigger on or off.

5.1.3 CreateInternalSpill

If no spill signals from ELSA are available, an internal spill can be created. This is done by setting the spill length and the duration between two spills as well as an enable signal via the VME bus. After the spill stop trigger a counter counts up until it matches with the value for the duration with no spill and the spill start trigger is created. After the spill start trigger, the counter is compared

with the value for the spill length and if it matches a spill stop trigger is created.

5.1.4 SpillTimeCounter

This component counts the amount of spill start signals, either from ELSA or the component *CreateInternalSpill*. Further does it count the clock cycles since the last spill start signal from which the time since the last spill start can be calculated.

5.1.5 Counter

The global trigger provides counters for every input channel, the data trigger attempts, the actual data trigger and the lifetime, that is the time between two global trigger signals.

5.1.6 TriggerLogiX

The component *TriggerLogiX* receives the local trigger signals from the single detectors of the experiment and checks them for equality with the adjusted trigger conditions. Figure 5.3 gives an overview of the components of *TriggerLogiX*. In the input block the incoming signals are sampled and prepared, so that they can be properly compared in the trigger logic block. The prescaler allows to scale down the output of the trigger logic.

To achieve the efficiency as described in section 3.2.1.1 the local trigger signals are DDR sampled. Thus, there are two sampled signals of each input

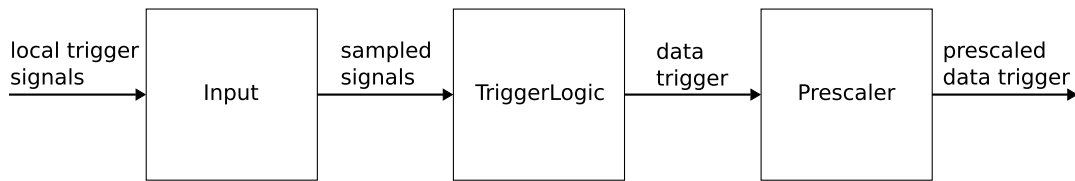


Figure 5.3: Schematic view of the *TriggerLogiX* component. The local trigger signals are sampled in the input block. The trigger logic block creates the data trigger if the local trigger signals meet at least one of the five trigger conditions. The prescaler allows to scale the output of the trigger logic down if necessary.

signal. Until the generation of the data trigger the sampled signals must be processed separately — both, of course, with the same settings.

5.1.6.1 Input

The local trigger signals from the single detectors, that arrive at the global trigger, are LVDS signals. More than 20 local trigger signals are expected. They are in detail: tagger, scintillator barrel, up to 15 from the BGO-ball, Čerenkov, SciFi2, up to 4 from TOF and GIM. Further planned detectors will provide more trigger signals. To meet the requirements of chapter 4 the global trigger provides 24 input channels.

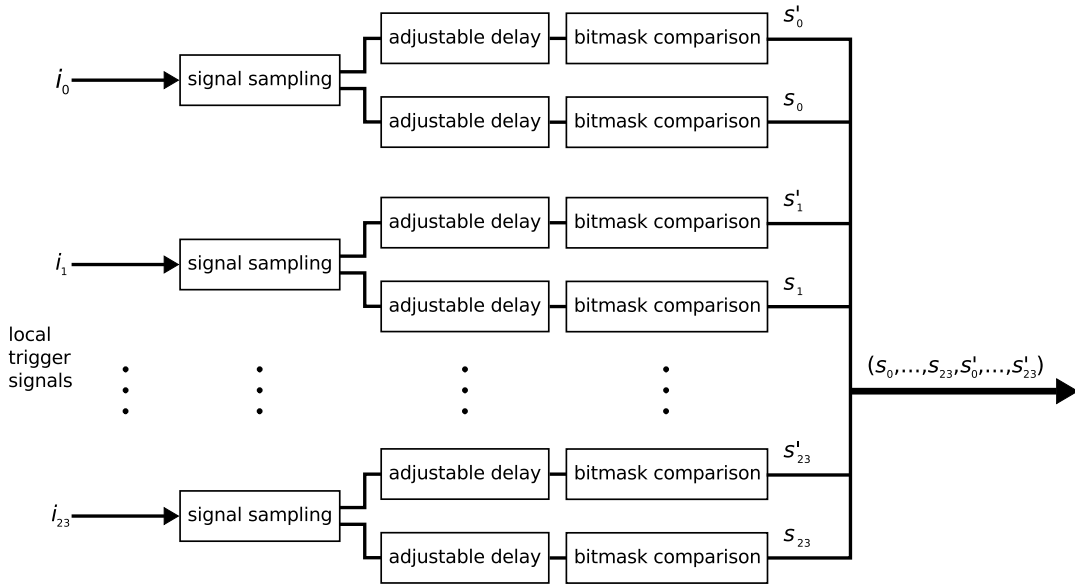


Figure 5.4: Schematic view of the input block. The signals are DDR sampled, which yields to two output signals s and s' , which must be separately processed. Then the signals are delayed and compared with a bitmask to take the signal's time jitter into account.

Figure 5.4 shows a schematic overview of the input block of component *TriggerLogiX*. First the local trigger signals are DDR sampled, which yields to two output signals of each input signal. Both output signals of an input signal are processed with the same settings. The sampled signal lines are written as a vector $(s_0, \dots, s_{23}, s'_0, \dots, s'_{23})$.

As described in section 3.2.1 the local trigger signals jitter in time by a constant amount for each detector. How the jitter is compensated is described below. To compare two or more signals their mean time of arrival must be equal. This is achieved in two steps. Each signal runs through an adjustable delay. Since the delay is clocked with the 200 MHz clock its timing precision is 5 ns. The fine tuning is done at signal sampling by a phase shift of a multiple of 90° of the sampling clock. This yields a timing precision of 1.25 ns for each signal for the adjustment of the mean time.

Signal Sampling To achieve a high efficiency for the global trigger the local trigger signals are DDR sampled, see section 3.2.1. The sampling frequency is 200 MHz. Thus, the sampling resolution is 5 ns, which means that the local trigger signals must have a length of at least 5 ns.

Figure 5.5 shows a schematic of the component *4phase_leading_edge_clipper*. The functions of this component are the sampling of the input signals and the fine tuning of the mean time of arrival.

The incoming signals are arbitrary in time and their length is adjusted so that it is always longer than the 5 ns of one clock cycle of the global trigger. The leading edge clipper is a module that DDR samples the incoming signals with the 200 MHz clock. Therefore the signals are sampled on both the rising edge of the normal clock and on the rising edge of its inverted clock. The sampling results are then output on two separated lines, but synchronous to the normal clock and with a fixed length of one clock cycle. A timing diagram of the leading edge clipper is shown in table 5.1.

To adjust the mean time of arrival the sampling clocks are phase shifted. Due to DDR sampling the clock phases of the normal and the inverted clock

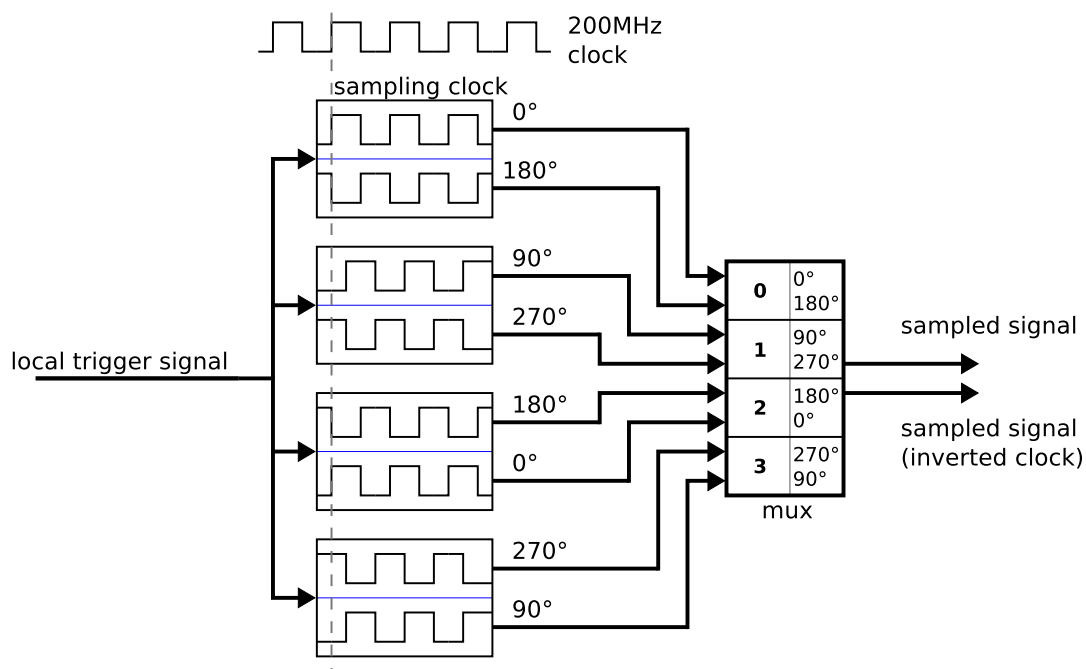


Figure 5.5: Schematic view of the component *4phase_leading_edge_clipper*. The input signal is DDR sampled by four clock phase combinations. The desired combination is then selected by a multiplexer and output afterwards. Note, that the sampling depends on the clock phases, but the output signals are synchronous to the 0°clock signal.

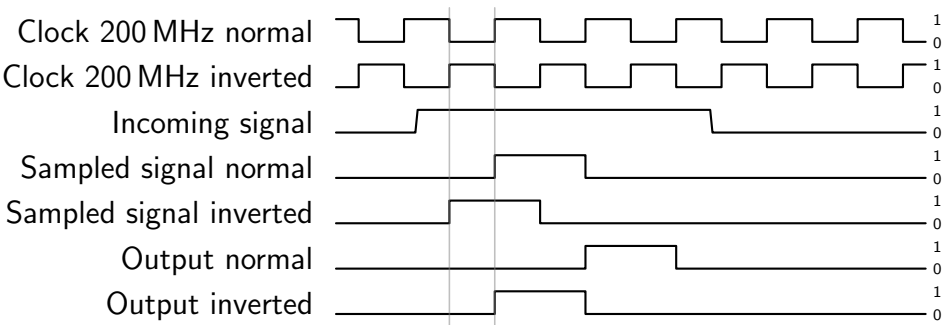


Table 5.1: Timing diagram of the leading edge clipper for DDR sampling. The incoming signal is sampled on the rising edge of the clocks and output on the next rising edge of the normal clock. Thereby the output of the inverted clock is synchronized to the normal clock.

must be shifted simultaneously. For each signal one may choose between four possible alignments: $0^\circ/180^\circ$, $90^\circ/270^\circ$, $180^\circ/0^\circ$ and $270^\circ/90^\circ$. The selection is made by a multiplexer in the component `4phase_leading_edge_clipper` as shown in figure 5.5. Four clock phases of a 200 MHz clock results in a precision of 1.25 ns for the sampling of one input signal. From now on the sampled signals are doubled.

The straightforward way of generating the four 200 MHz clocks with different phases would be to use one DCM with the 100 MHz clock source as input and select the CLK2X output which doubles the frequency of the input clock. Then in the next step the 200 MHz signal would be used as the input of a second DCM, with the CLK0, CLK90, CLK180 and CLK270 outputs, which shift the clock phases respectively. However, due to architectural reasons of the DCM it is not possible to use the common phase shifted outputs with an input frequency higher than the maximum basis frequency of 167 MHz. Thus, the capability of the DCMs to phase shift the input clock signal, which affects all outputs equally, is used. To generate the four clock phases two DCMs, which both take the 100 MHz clock signal as input and the CLK2X (double frequency) and CLK2X180 (double frequency, inverted) outputs, are used. For one DCM the input signal is phase shifted by 45° , which means a phase shift of 90° after the output frequency has doubled. This results in four clocks with 200 MHz and phase shifts of 0° , 90° , 180° and 270° .

Adjustable Delay The adjustable delay is made of four connected LUTs that work as a shift register. The sampled signal enters the shift register and is shifted by one field each clock cycle. The point of exit is adjustable. Thus, the delay is adjustable from a minimum of 5 ns up to a maximum of 320 ns.

Bitmask Comparison To take the jitter into account the signal that arrived first must wait for the duration of the jitter length of the second signal to allow the second signal to arrive. This is achieved by running each signal through a shift register, that is read out every clock cycle and compared with a bitmask.

The bits of the bitmask can be set freely at any time of operation using the C++ interface, of course they are best set before data acquisition.

Since the clock frequency is 200 MHz and the maximum jitter comes from the TOF wall with a calculated value of 80 ns, the bitmask must have a minimum width of $80 \cdot 10^{-9} \text{ s} \cdot 200 \cdot 10^6 \text{ s}^{-1} = 16 \text{ ([bit])}$. The maximum bitmask width is 16 bit, with a 200 MHz clock that equals 80 ns.

The functionality of the bitmask comparison is illustrated for one incoming signal in figure 5.6. The signal is sent through a shift register which is compared with a bitmask at every clock cycle. A bitwise coincidence of the sampled signal and the bitmask is the output of the input module. The incoming signal is moving with every clock cycle at one bit, whereas the bitmask pattern is fixed in time. The signal will therefore hold, and hence the output will be true, as long as it is under the high bits of the bitmask.

5.1.6.2 TriggerLogic

In this module the sampled local trigger signals are compared respectively to specific trigger conditions and if the comparison is true a trigger signal is sent to the module *CreateTriggerSignal*. Figure 5.7 shows the functionality of the trigger logic.

The trigger logic consists of two logic blocks and a prescaler block. The first logic block consists of two times eight AND components (*TriggerLogicAND*) and eight OR components (*TriggerLogicOR*) that receive the sampled signals.

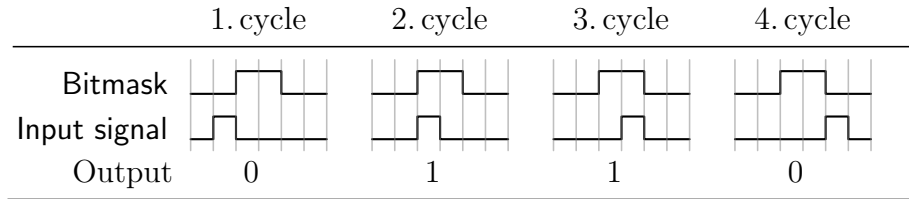


Figure 5.6: Illustration of the functionality of the input bitmasks: Pictured is a section of 6 bits of an input bitmask and an input signal and their output (coincidence) at four sequenced clock cycles. The output is an OR of a bitwise coincidence of the bitmask and the input signal. The output becomes high if the input signal is under the high bits of the bitmask.

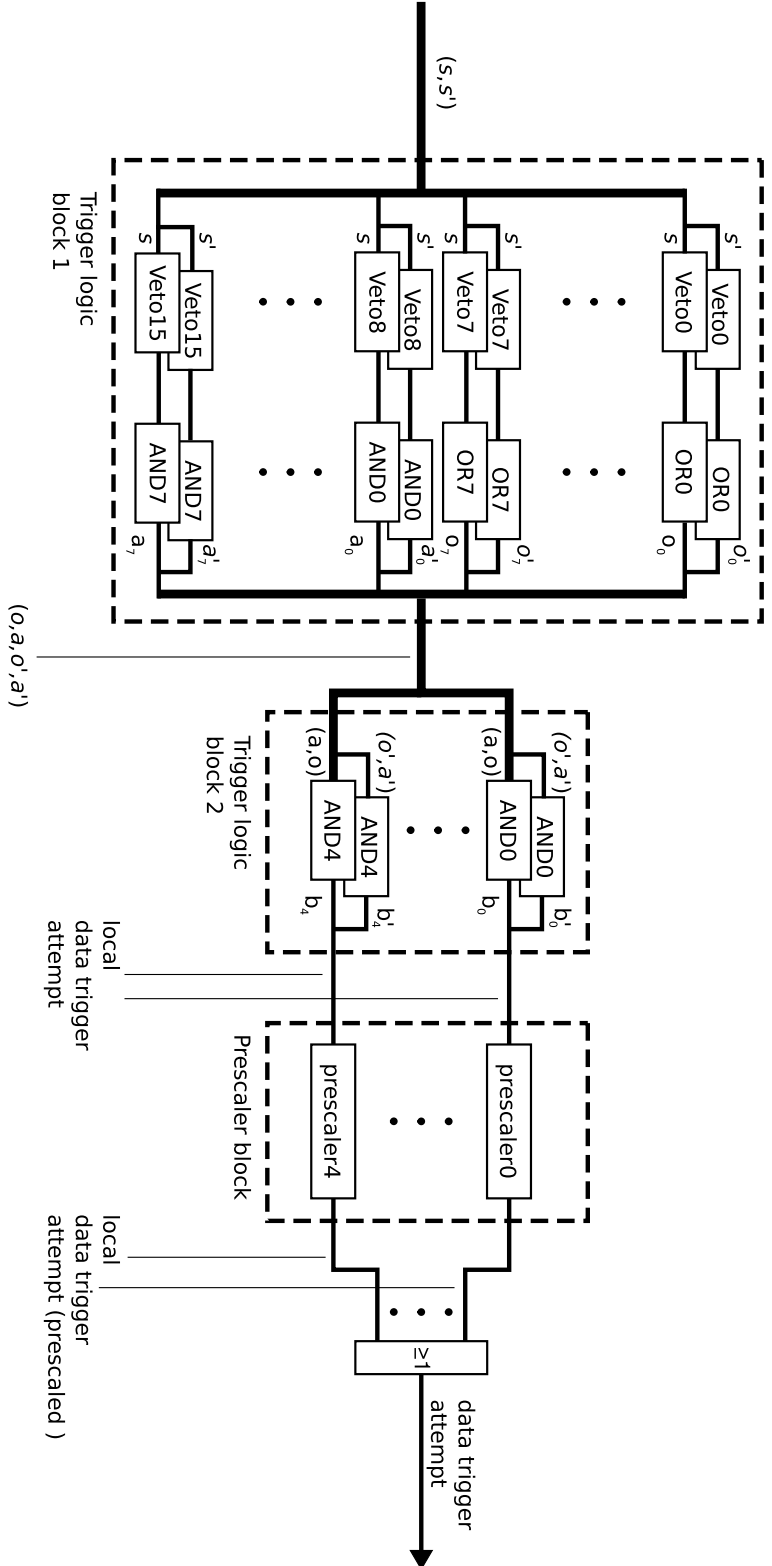


Figure 5.7: The trigger logic can be thought of three blocks. The first logic block consists of several components with adjustable conditions. Each component exists twice for the signals sampled with the normal clock and the signals sampled with the inverted clock. The components get the local trigger signals as input. If the signals match the trigger conditions inside a component the component's output is set to logic high. The second logic block gets the outputs of the first logic block. If they match the block's condition its output is set to logic high. The prescaler block allows to scale the output rate of the second trigger logic block down. The output signal, then, go into an OR, whose output is the data trigger.

Each component exists twice, because of the doubling of the sampled signals. In front of each component a veto component (*TriggerLogicVeto*) is placed. For each component a selection is made about which signals are taken into account for the internal operations. The veto component forces the output of the following component to be zero if one or more of the selected input signals are logic high. The input signals are forwarded unchanged to the following AND or OR component. For the OR components it is sufficient that one or more of the selected incoming signals is logic high to set the output high. For the AND components on the other hand the incoming signals have to match the selection exactly to set the output high. The output signals of the AND a_{out} and OR o_{out} components are the input signals of the second block. This block consists of two times five AND components, that differ to the first block's AND components only in not having vetos in front of it. The output signals of the second block's ANDs are already the data trigger attempts. The signals are still doubled, but we are only interested in the first one. Thus, the doubled signals are combined with an OR and then forwarded to the prescaler block. In the prescaler block the output rate of the data trigger attempts can be scaled down. The output signals of the prescalers, then, go into an OR, whose output is the actual data trigger.

Now, let us have a look on an example to see how the trigger conditions work. Lets assume you want to detect the Δ -resonance, where it decays into a π^0 and a proton or neutron. The π^0 decays, with a probability of 98%, into two photons. Then you would trigger on two photons and a proton or a neutron. That means for the BGO-OD experiment you would probably want to trigger on the tagger to know the Energy of the incident photon, the BGO ball for the remaining photons and probably the neutron and on the OR of the Scifi2 detector and the TOF wall. Your trigger conditions in its minimal form would look like

$$(\text{tagger} \wedge \text{BGO}) \wedge (\text{Scifi2} \vee \text{TOF})$$

Thus, you would need one AND and one OR of the first trigger logic block and one AND of the second trigger logic block of the trigger logic.

Now consider the case, where you want to detect the kaons of the process $\gamma p \rightarrow K^+ \Lambda(1405) \rightarrow K^+ n 2\pi^0 / K^+ p \pi^- \pi^0 \gamma$. The Čerenkov detector can be used to distinguish between charged pions and kaons. The light pions are fast enough to emit Čerenkov light and therefore are detected by the Čerenkov detector. On the other hand, kaons with the same energy are too slow to emit Čerenkov light. Thus, one would only want events where the Čerenkov detector does not trigger. To achieve this a veto can be assigned to one or more incoming signals in the first block of the trigger logic, so that if one of this signals is logic high the output of the AND or OR is forced to logic low, see figure 5.7.

The trigger condition can be described as

$$(\text{tagger} \wedge \text{BGO} \wedge \neg \check{\text{Čerenkov}}) \wedge (\text{SciFi2} \vee \text{TOF})$$

which is only true if the Čerenkov detector does not trigger.

The components of the trigger logic are now described in detail. Since there are many different ways implementing logic circuits, the schematics below must be considered as equivalent circuit diagrams and not as the actual FPGA implementation.

1. 1st Trigger Logic Block

TriggerLogicVeto The veto component (*TriggerLogicVeto*) forces the following AND or OR component to logic low if dedicated local trigger signals are logic high. The input signals stay unchanged and are forwarded to the output of the veto component. Which of the input signals $i = (i_0, \dots, i_{23})$ is taken as a veto signal is selected by the veto register $v = (v_0, \dots, v_{23})$, which is set individually for each veto component. The input signals are bitwisely anded with the veto register and passed to a NOR. The output of the NOR is logic high if none of the selected input signals is logic high, otherwise it is logic low. Note that the output is logic high if no veto occurs and logic low otherwise. The schematic of the veto component is shown in figure 5.8.

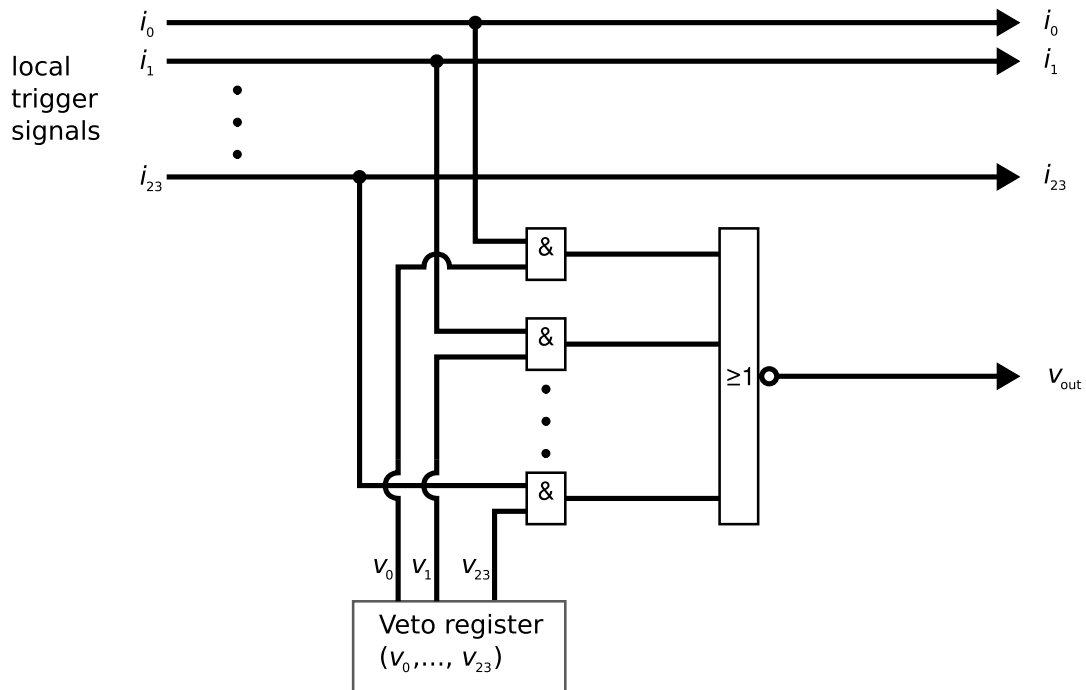


Figure 5.8: Schematic of the veto component (*TriggerLogicVeto*). The output v_{out} is set to logic low if one of the Veto register selected inputs is logic high. The local trigger signals are forwarded unchanged.

TriggerLogicAND The AND component (*TriggerLogicAND*) compares its selected input signals for coincidence. The input signals are the local trigger signals, which are forwarded from the veto component. Which of them are taken into account is selected by the AND register a . If all selected input signals are logic high and there is no veto ($v_{\text{out}} = 1$) of its previous veto component the output of the AND component is set to logic high. Figure 5.9 shows a schematic of the AND component.

TriggerLogicOR For the OR component (*TriggerLogicOR*) the output is set logic high if at least one of the selected input signals is logic high and there is no veto from the previous veto component ($v_{\text{out}} = 1$). The input signals are the local trigger signals, which are forwarded from the veto component. The OR register o selects the input signals that are taken into account, which are the by the previous veto component forwarded local trigger signals. The schematic of the component *TriggerLogicOR* is shown in figure 5.10.

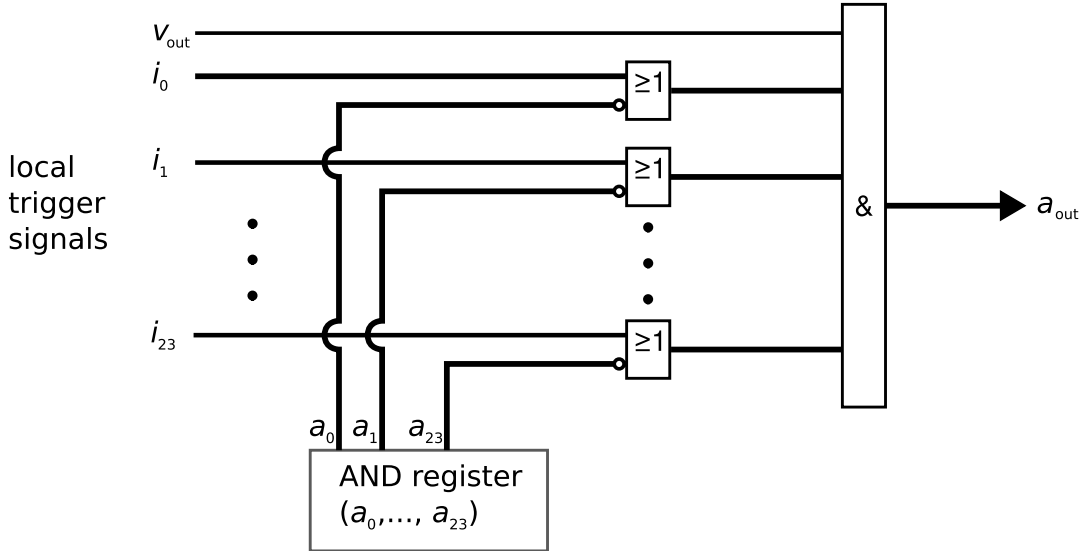


Figure 5.9: Schematic of the AND component (*TriggerLogicAND*). The veto signal v_{out} and all of the AND register selected input signals must be logic high to set the output $a_{\text{out}} = 1$.

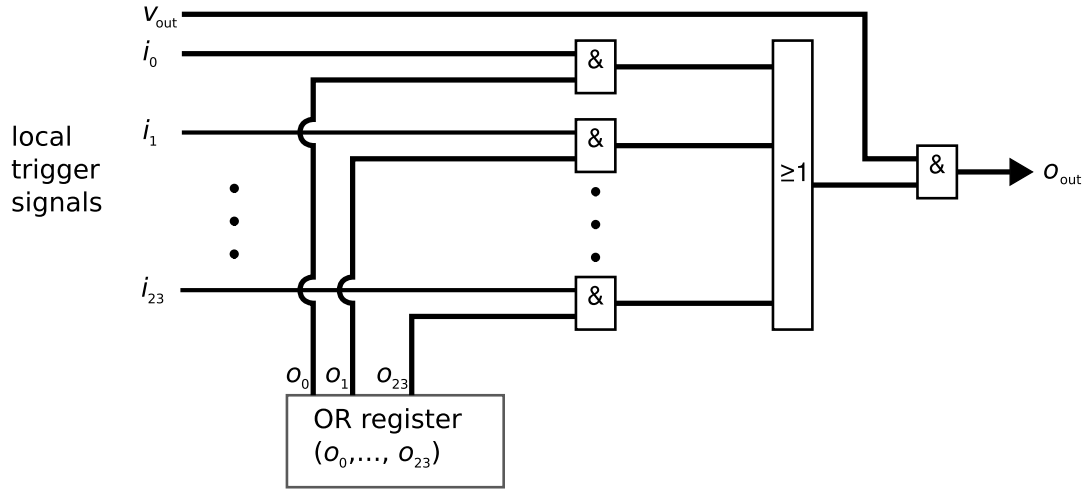


Figure 5.10: Schematic of the OR component (*TriggerLogicOR*). The veto signal v_{out} and one of the OR register selected input signals must be logic high to set the output $o_{out} = 1$.

2. 2nd Trigger Logic Block

2ndTriggerLogicAND The AND component (*2ndTriggerLogicAND*) of the second trigger logic block equals the AND component of the first block, but has no veto component in front of it. Instead of the local trigger signals it receives the output signals of the first block's components as input. A schematic is shown in figure 5.11

3. Prescaler Block

To allow a reduction in trigger attempts the trigger signal runs through a prescaler that only lets every Nth signal pass before it is sent as a trigger attempt to the *CreateTriggerPulse* module. The prescaler amount N is set over the VME bus.

5.1.7 CreateTriggerPulse

This module creates the global trigger pulse. It also adds information about the event number and the trigger source. Possible inputs are trigger logic, spill

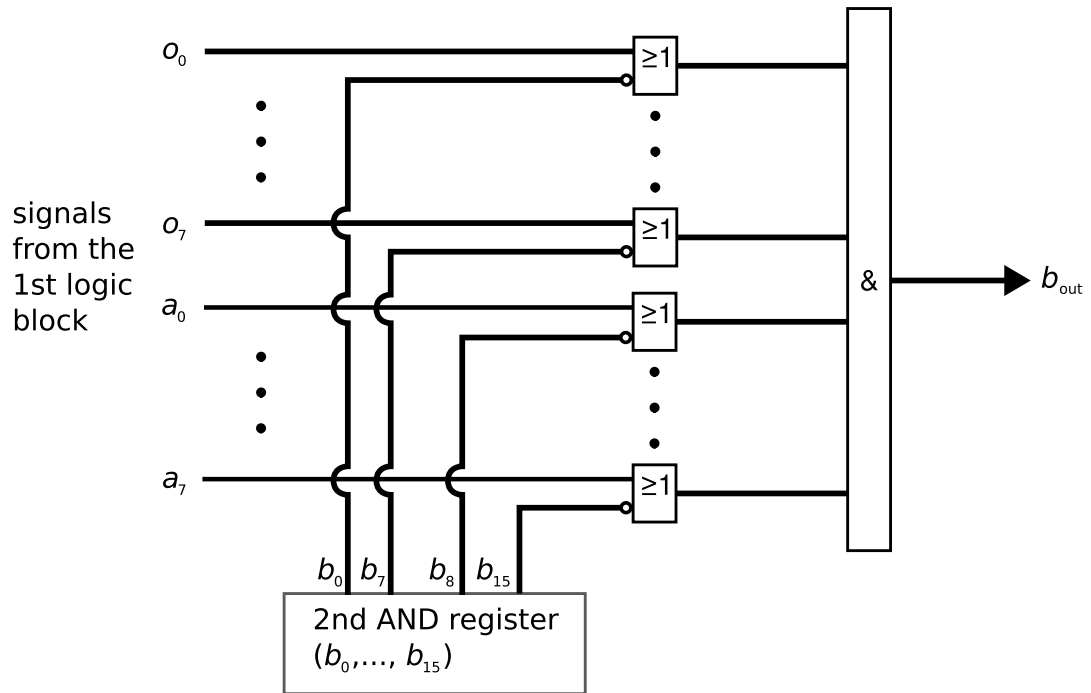


Figure 5.11: The AND component of the second block receives the output signals of the first block as input. The 2nd AND register selects the input signals that are taken into account. If all selected input signals are logic high the output b_{out} is set to logic high.

start, spill stop and scaler trigger. The coding of the trigger source is shown in table 5.2. Only one trigger is allowed at a time. Once a global trigger pulse is created a latch, called trigger latch, is set. This blocks the module for further trigger attempts until the latch is reset over the VME bus. This is done by the master levb after all of the levb modules are ready for the next event. However, if the module is blocked and a scaler trigger or one of the spill triggers occur they are put into a waiting loop by setting a bit, corresponding to its trigger type. The trigger signal is then fired immediately after the trigger latch is reset. If a trigger occurs that is already in the waiting loop, the bit keeps its status, but the trigger signal is not fired twice. This case is very unlikely, however, because the scaler trigger will presumably occur with a frequency of 20 Hz and the spill trigger occur with a frequency in the order of a few seconds. Information about the event number and the trigger source is generated and stored in a register to be read out over the VME bus and also sent as an 8 bit data packet to the DAQ.

The register is called trigger information register and shows the event number, the status of the trigger latch and the trigger source.

Bits	31-15	14-12	11-9	8	7-0
Content	-	trigger source	-	trigger latch status	event number

The data packet consists of 8 bits. The first three bits of the trigger information, that is bits 0 to 2, include the trigger source, as shown in table 5.2. The forth bit on the left is cropped and only the three remaining bits are sent. The event number is attached to the bits 3 to 7. Since there are only 5 bits

coding	trigger source
0000	trigger logic
0001	spill on
0010	spill off
0100	scaler trigger

Table 5.2: Selection mask for the trigger sources in the trigger information

available, that corresponds to 0 to 31 in decimal numbers, only the lower bits of the event numbers are sent. To reduce data lines it was decided to send the global trigger signal and the trigger information as serial data over one of the NIM outputs of the FPGA board to the sync master. For this, the global trigger module uses the component *trigger_link_output* to transmit the global trigger signal and the data to the sync master, where it is received by the component *trigger_link_input*. In the component *trigger_link_output* the global trigger signal is created synchronous to the 200 MHz clock. Then synchronous to the next rising edge of the 40 MHz clock the transmission of the serial data starts. The data bit is sent synchronous to the rising edge of the 40 MHz clock. The serial data consists of a start bit (logic low), eight data bits and a parity bit. Afterwards the line is held logic high until the trigger latch is reset. In the sync master the signals are received by the component *trigger_link_input*. The global trigger signal is then distributed to the levbs and the serial data is decoded. The timing of the transmission is shown in the timing diagram 5.3.

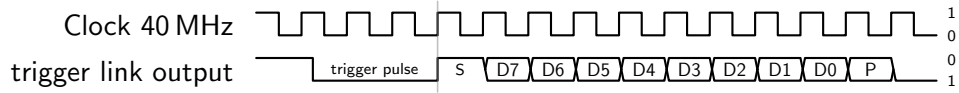


Table 5.3: Timing diagram of the global trigger pulse and the serial data: After the asynchronous trigger pulse, a start bit S, eight data bits and a parity bit P are sent synchronous to the DAQ. Afterwards the signal line is hold high until the trigger latch is reset.

5.2 b1GlobalTrigger — C++ Interface

Besides the VHDL code for the FPGA a C++ interface with name b1GlobalTrigger was developed. It provides functions to load the firmware of the global trigger module into the FPGA and also provides easy read and write access to the settings of the global trigger and allows to read out the status registers and counter values.

At first the constructor of the C++ class loads the firmware into the FPGA. The settings are then read from a XML file and sent to the FPGA. However, it

is also possible to change the settings at runtime with the commands provided by the interface. A class reference of the interface is attached to appendix A.2.

5.3 TriggerGUI — Graphical User Interface

The graphical user interface TriggerGUI is designed to build the XML configuration file easily. It is based on Qt a cross-platform application framework for graphical user interfaces. The GUI is clearly arranged by splitting it into four tabs. The first holds general information, like the location of the bit file for the FPGA and the base address in the VME crate. It also contains panels to set the internal spill and the scaler trigger. In the second tab the input bitmasks and the delay can be set. Also, the input channels can be named for proper identification. In the third tab the trigger logic is set. This tab is therefore split into three additional tabs. The first one has an overview of the trigger logic. The second tab has the settings of the first logic step, that is the ANDs and ORs. The last tab of the trigger logic holds the settings of ANDs of second logic step and the prescaler. The fourth tab shows the actual XML output file. The figures 5.12 and 5.13 shows how the trigger logic is adjusted with the GUI for a theoretical configuration for the detection of the $\gamma p \rightarrow K^+ \Lambda(1405)$ reaction. An example XML file is attached to appendix A.3.

General

Input

Trigger Logic

XML file

SciF2

57.00 ns

B

Phase selection

☒ 0/180

☐ 90/270

☐ 180/0

☐ 270/90

Input Bitmask (in ns)

5

10

15

20

25

30

35

40

45

50

55

60

65

70

75

80

BGO Hall

141.30 ns

1C

Phase selection

☐ 0/180

☒ 90/270

☐ 180/0

☐ 270/90

Input Bitmask (in ns)

5

10

15

20

25

30

35

40

45

50

55

60

65

70

75

80

Trigger

219.20 ns

2B

Phase selection

☐ 0/180

☐ 90/270

☐ 180/0

☒ 270/90

Input Bitmask (in ns)

5

10

15

20

25

30

35

40

45

50

55

60

65

70

75

80

Cherenkov

21.00 ns

4

Phase selection

☐ 0/180

☐ 90/270

☐ 180/0

☐ 270/90

Input Bitmask (in ns)

5

10

15

20

25

30

35

40

45

50

55

60

65

70

75

80

TOF

100.80 ns

14

Phase selection

☒ 0/180

☐ 90/270

☐ 180/0

☐ 270/90

Input Bitmask (in ns)

5

10

15

20

25

30

35

40

45

50

55

60

65

70

75

80

Figure 5.12: Detail of the input tab of the TriggerGUI. Here the delays, the clock phases of the sampling clock and the input bitmasks can be set for the local trigger signals.

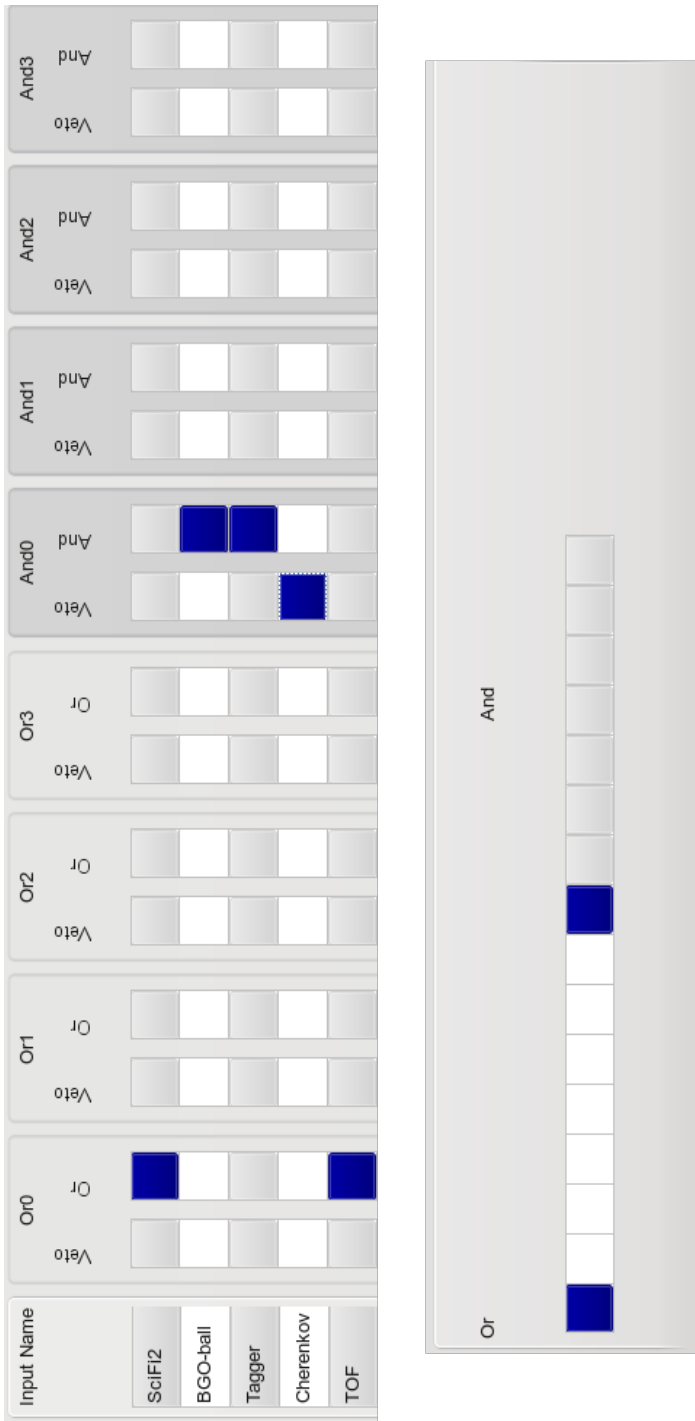


Figure 5.13: Detail of the trigger logic tab of the TriggerGUI. Above the ANDs and ORs with their vetos of the first trigger logic step can be adjusted. Below the second step of the trigger logic is shown. There the outputs of the first step can be selected.

Chapter 6

Efficiency

The efficiency, as described in section 3.2.1.1, is an important property of the global trigger module. A high efficiency is mandatory to achieve a high event rate of the events the experiment is going for. Therefore the efficiency of the global trigger module has been measured.

6.1 Measurement

In this measurement the efficiency of the global trigger module with reference to the coincidence of two input signals is measured. The coincidence of the global trigger module is compared with the coincidence of an external coincidence module. The experimental setup consists of two function generators, the external coincidence module, the global trigger module and a TDC with a timing resolution of 100 ps. The setup is shown in figure 6.1.

Two input signals, S_1 and S_2 , with different frequency and arbitrary phase are generated by the function generators F_1 and F_2 . These signals are the input signals for the global trigger module and the external coincidence module. In the global trigger module the trigger conditions are set for a coincidence of the input signals within coincidence window of 5 ns. The coincidence window of the external coincidence module is several times as large as the coincidence window

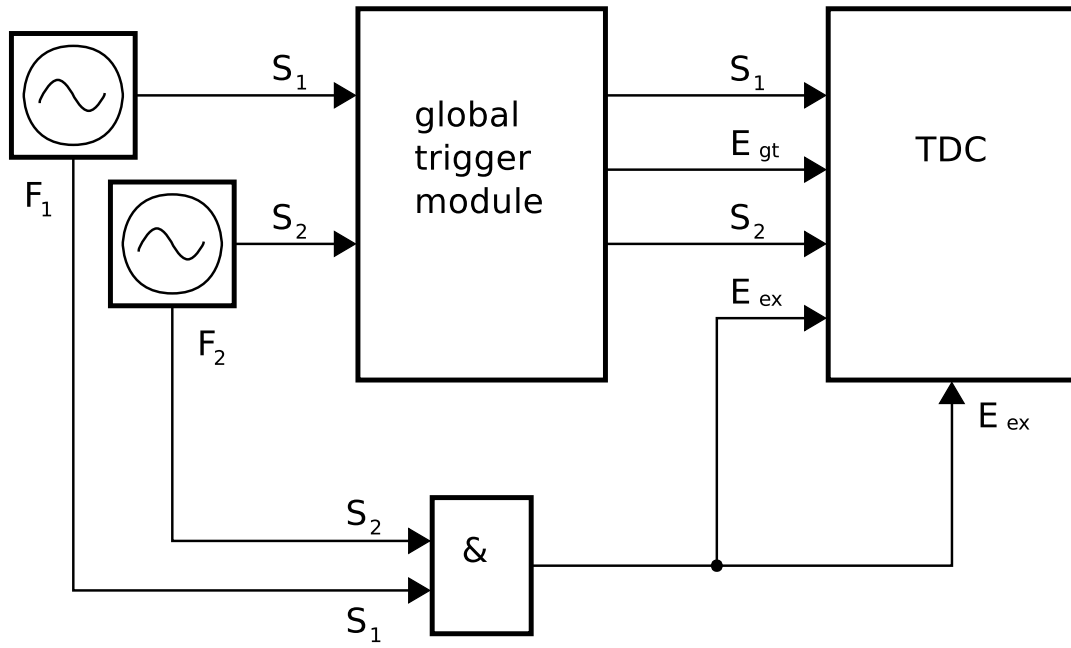


Figure 6.1: Setup of the efficiency measurement. Two independent signals, S_1 and S_2 , are tested for coincidence. The coincidence of the global trigger module E_{gt} and an external coincidence module E_{ex} are compared in a TDC.

of the global trigger module. This is necessary, because the external coincidence is used as reference. As output signal E_{gt} of the global trigger module a trigger attempt signal, which is output over the LVDS output, is used. The global trigger signal is not used, because it needs to be latched after each trigger and to avoid conflicts with the serial data, which is sent on the same output right after the global trigger signal. The input signals that go into the global trigger module are forwarded through to the TDC. The output signals of the global trigger module and of the external coincidence E_{ex} are also sent to the TDC. The TDC's data acquisition is triggered by the output signal of the external coincidence module. From the TDC data the efficiency

$$\text{eff} = \frac{E_{gt}}{E_{ex}}$$

of the global trigger module is calculated and plotted against the time difference

$$d = t(S_1) - t(S_2)$$

of the input signals.

6.2 Results

The result of the measurement is shown in figure 6.2, where on the abscissa the time difference of the two input signals is shown and on the ordinate its attendant efficiency. The theoretically expected efficiency is shown in figure 6.3.

The efficiency of the global trigger module has a plateau with an efficiency of exactly one from -2.5 ns to 2.5 ns and linearly falling edges to $\pm 5\text{ ns}$. The corners at the points $\pm 5\text{ ns}$ and $\pm 2.5\text{ ns}$ of the measured efficiency are rounded. This is due to the resolution of 100 ps of the TDC and the jitter of the signals E_{gt} and E_{ex} . The resolution and the jitter has to be taken into account for both signals. The achieved efficiency of the global trigger module is in good consistency with the theoretically expected efficiency. This makes the global trigger module usable for the BGO-OD experiment.

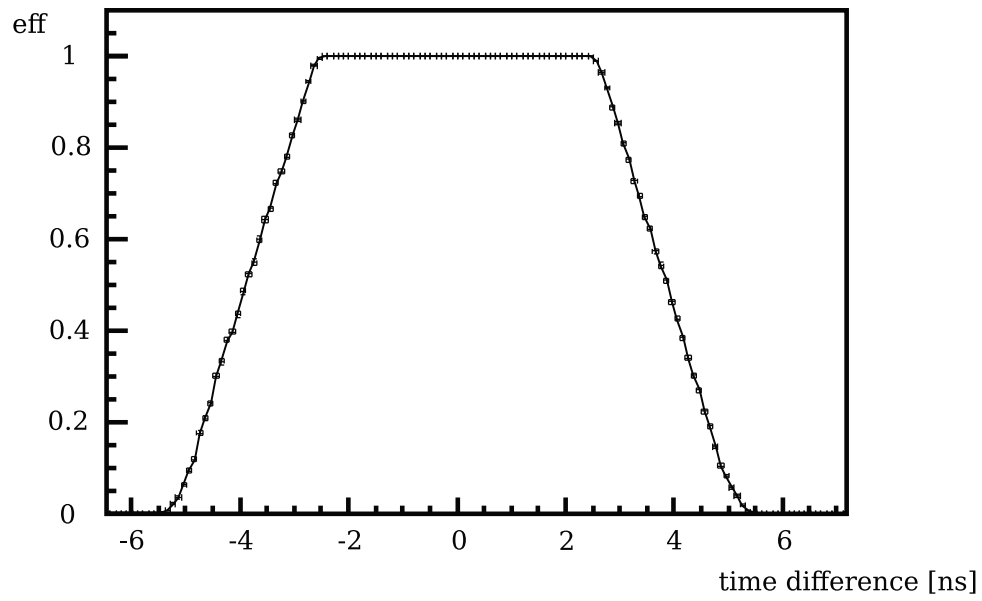


Figure 6.2: Result of the measurement of the efficiency of the coincidence of two DDR sampled signals

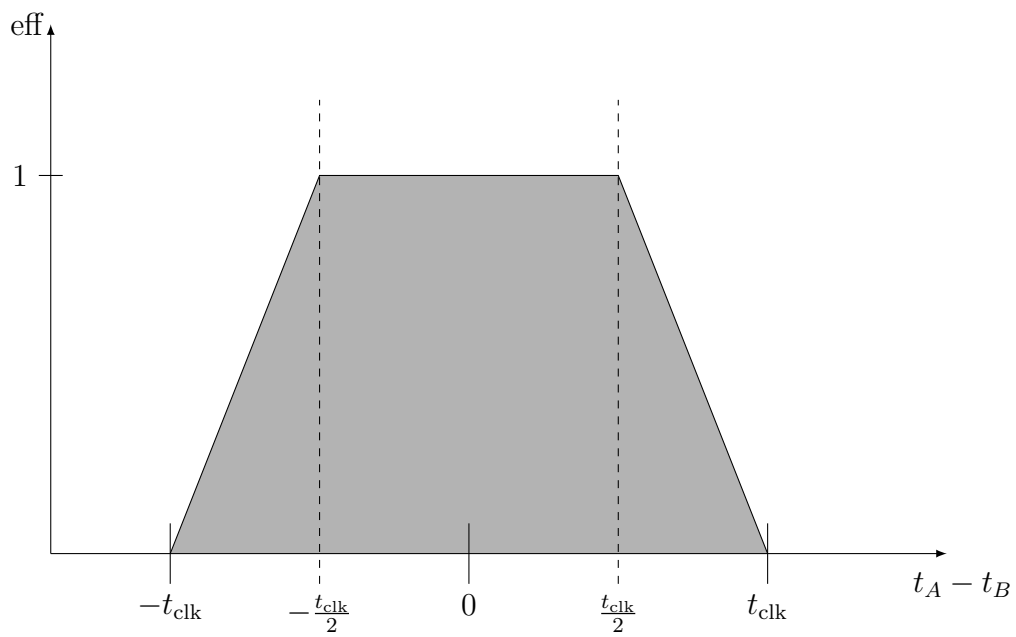


Figure 6.3: The theoretical efficiency of the coincidence of two DDR sampled signals.

Chapter 7

Summary and Outlook

With this thesis the development and test of the global trigger of the BGO-OD experiment is described. This includes the development of the firmware of the FPGA of the global trigger module and its implementation into the master local event builder of the DAQ. Part of the implementation was the development of an interface, which loads the firmware into the FPGA and provides functions to read out the counters and set the trigger conditions. Further, a GUI was developed to easily configure the settings of the global trigger module. The Global Trigger Module provides 24 input channels for the local trigger signals of the detectors with a timing resolution of 5 ns. The mean time of arrival of the signals is adjusted by delaying them up to 320 ns with a precision of 1.25 ns. With a bitmask comparison, up to 80 ns of the timing variation of the local trigger signals are compensated. To increase the efficiency the local trigger signals are sampled on the rising and the falling edge of the sampling clock. The global trigger module handles the spill start and spill stop trigger, the scaler trigger and the data trigger. To create the data trigger, the local trigger signal are compared with five adjustable trigger conditions. This enables to trigger on five different reactions at the same time.

The efficiency of the developed global trigger module was measured on the basis of the coincidence of two independent signals. An external coincidence module was used as a reference. It has been shown that the efficiency for time differences up to 2.5 ns of the input signals is exactly one, then up to time

differences of 5 ns the efficiency is linearly falling to zero, as expected.

The next generation FPGA board from ELB is currently under development. The FPGA of the board will be a Xilinx Spartan 6, which supports higher frequencies. It will be possible to adapt the firmware, in order to improve the sampling resolution of the global trigger module. In addition, the GUI can be upgraded to work as an online interface to monitor and adjust the trigger's settings and status at runtime.

Appendix A

Appendix

A.1 Logic Gates

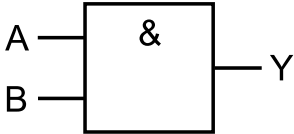
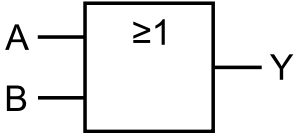
Type	Symbol	Truth table															
AND		<table><tr><td>a</td><td>b</td><td>c</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	a	b	c	0	0	0	0	1	0	1	0	0	1	1	1
a	b	c															
0	0	0															
0	1	0															
1	0	0															
1	1	1															
OR		<table><tr><td>a</td><td>b</td><td>c</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	a	b	c	0	0	0	0	1	1	1	0	1	1	1	1
a	b	c															
0	0	0															
0	1	1															
1	0	1															
1	1	1															
Continued on next page																	

Table A.1 – continued from previous page

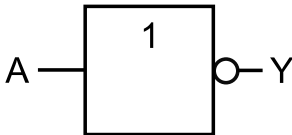
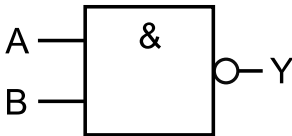
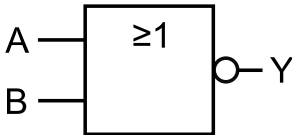
Type	Symbol	Truth table															
NOT		<table><tr><th>a</th><th>$\neg a$</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	a	$\neg a$	0	1	1	0									
a	$\neg a$																
0	1																
1	0																
NAND (NOT AND)		<table><tr><th>a</th><th>b</th><th>c</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	a	b	c	0	0	1	0	1	1	1	0	1	1	1	0
a	b	c															
0	0	1															
0	1	1															
1	0	1															
1	1	0															
NOR (NOT OR)		<table><tr><th>a</th><th>b</th><th>c</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	a	b	c	0	0	1	0	1	0	1	0	0	1	1	0
a	b	c															
0	0	1															
0	1	0															
1	0	0															
1	1	0															
Continued on next page																	

Table A.1 – continued from previous page

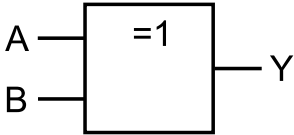
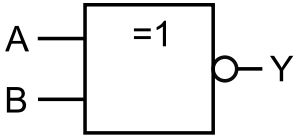
Type	Symbol	Truth table															
XOR (EXCLUSIVE OR)		<table border="1"> <thead> <tr> <th>a</th><th>b</th><th>c</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	a	b	c	0	0	0	0	1	1	1	0	1	1	1	0
a	b	c															
0	0	0															
0	1	1															
1	0	1															
1	1	0															
XNOR (EXCLUSIVE NOT OR)		<table border="1"> <thead> <tr> <th>a</th><th>b</th><th>c</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	a	b	c	0	0	1	0	1	0	1	0	0	1	1	1
a	b	c															
0	0	1															
0	1	0															
1	0	0															
1	1	1															

Table A.1: Overview about common logic symbols. Symbol images: [Wik11]

A.2 b1GlobalTrigger Class Reference

Interface to program the FPGA board and configure the global trigger.

```
#include <b1GlobalTrigger.h>
```

Public Member Functions

b1GlobalTrigger ()

do not use this constructor

b1GlobalTrigger (XMLini *xmlconfig, int number)

The constructor of this class.

bool programFPGA (XMLini *xmlconfig, int number)

Programs the FPGA.

bool loadConfig (XMLini *xmlconfig, int number)

Configures the b1GlobalTrigger.

void EnableTriggerGlobal (bool)

Enable/disable global enable.

void EnableCreateInternalSpill (bool)

Enable/disable internal spill.

void EnableScalerTrigger (bool)

Enable/disable scaler trigger.

void EnableExternalSpill (bool)

Enable/disable external spill.

bool GetEnableTriggerGlobal ()

Returns status of global enable flag.

void ResetTriggerLatch ()

Reset trigger latch.

void ResetTriggerInformation ()

Reset trigger information register.

void ResetTriggerLogicCounters ()

Reset trigger logic counters.

void ResetNofSpills ()

Reset NofSpills counter.

void LatchNofSpills ()

Latch NofSpills counter.

void LatchTriggerLogicCounters ()

Latch trigger logic counters.

bool GetSpillStatus ()

Returns spill status: true = spill on, false = spill off.

unsigned int GetTriggerInformation ()

Returns the trigger information register bit 11 latch status 7-0 event no.

bool GetTriggerLatch ()

Returns status of trigger latch: true = latch set.

unsigned int GetNofSpills ()

Returns number of spills.

unsigned int GetSpillTime ()

Returns time since last spill on trigger.

unsigned int GetEnableTriggerLogic ()

Returns which trigger in trigger logic are active.

unsigned int GetTriggerLogicInputBitmask (unsigned int number)

Returns bitmask of input >number<.

unsigned int GetTriggerLogic2ndRowAndRegister (unsigned int number)

Returns bitmask of 2nd AND in trigger logic >number<.

unsigned int GetTriggerLogicAndRegister (unsigned int number)

Returns bitmask of AND >number<.

unsigned int GetTriggerLogicOrRegister (unsigned int number)

Returns bitmask of OR >number<.

unsigned int GetTriggerLogicVetoRegister (unsigned int number)

Returns bitmask of veto.

unsigned int ReadEventNumber ()

Returns event no.

unsigned int ReadTriggerCounter (unsigned int number)

Returns number of trigger attempts before the prescaler.

unsigned int ReadTriggerAttemptCounter ()

Returns nof trigger attempts after the prescaler.

unsigned int ReadTotalTriggerCounter ()

Returns nof of global triggers.

unsigned int ReadInputCounter (unsigned int number)

Returns number of rising edges in input channel >number<.

unsigned int ReadLifetimeCounter ()

Returns time since last global trigger.

unsigned int GetPrescaler (unsigned int number)

Returns prescaler of trigger >number<.

unsigned int GetInputDelay (unsigned int number)

Returns the delay of input >number<.

void SetTimeSpillIsOn (int SpillIsOn)

Set length of internal spill.

void SetTimeSpillIsOff (int SpillIsOff)

Set pause between two spills.

void SetScalerTriggerFreqDiv (int ScalerTriggerFreqDiv)

Set scaler trigger frequency divider,.

void SetEnableTriggerLogic ()

enables trigger logic

void SetEnableTriggerLogic (unsigned int pattern)

Set which internal trigger contributes to the global trigger. Value depends on actual trigger hardware.

void SetEventNumber (unsigned int number)

Set event number that is stored in trigger information and serial trigger output.

void SetInputDelay (unsigned int delay, unsigned int channel)

Set delay in clock cycles for each input channel.

void SetTriggerLogicInputBitmask (unsigned int bitmask, unsigned int number)

Set bitmask for input channel.

void SetTriggerLogic2ndRowAndRegister (unsigned int bitmask, unsigned int number)

Set the bitmask for the second AND in the trigger logic.

void SetTriggerLogicAndRegister (unsigned int bitmask, unsigned int number)

Set the bitmask for the AND in the trigger logic.

void SetTriggerLogicOrRegister (unsigned int bitmask, unsigned int number)

Set the bitmask for the OR in the trigger logic.

void SetTriggerLogicVetoRegister (unsigned int bitmask, unsigned int number)

Set veto inputs for trigger logic.

void SetPrescaler (unsigned int number, unsigned int prescaler)

Set the prescaler value for each internal trigger.

Protected Member Functions

unsigned int GetUIntFromXML (string xml_string)

Searches the XML file for >xml_string< and returns unsigned int from string.

string GetStringFromXML (string xml_string)

Searches the XML file for >xml_string< and returns string from string.

int GetIntFromXML (string xml_string)

Searches the XML file for >xml_string< and returns int from string.

unsigned long long GetULongLongFromXML (string xml_string)

Searches the XML file for >xml_string< and returns unsigned long long from string.

double GetDoubleFromXML (string xml_string)

Searches the XML file for >xml_string< and returns double from string.

float GetFloatFromXML (string xml_string)

Searches the XML file for >xml_string< and returns float from string.

unsigned int GetUIntFromXML (string xml_string, unsigned int defaultVal)

Searches the XML file for >xml_string< and returns unsigned int from string.

If >xml_string< is not found >defaultVal< is returned.

string GetStringFromXML (string xml_string, string defaultVal)

Searches the XML file for >xml_string< and returns string from string.

If >xml_string< is not found >defaultVal< is returned.

int GetIntFromXML (string xml_string, int defaultVal)

Searches xml file for >xml_string< and returns int from string.

If >xml_string< is not found >defaultVal< is returned.

unsigned long long GetULongLongFromXML (string xml_string, unsigned long long defaultVal)

Searches the XML file for >xml_string< and returns unsigned long long from string.

If >xml_string< is not found >defaultVal< is returned.

double GetDoubleFromXML (string xml_string, double defaultVal)

Searches the XML file for >xml_string< and returns double from string.

If >xml_string< is not found >defaultVal< is returned.

float GetFloatFromXML (string xml_string, float defaultVal)

Searches the XML file for >xml_string< and returns float from string.

If >xml_string< is not found >defaultVal< is returned.

Protected Attributes

volatile unsigned int * **LifetimeCounterAddress**

volatile unsigned int * **TimeSpillsOnAddress**

volatile unsigned int * **TimeSpillsOffAddress**

volatile unsigned int * **TriggerInformationRegisterAddress**

volatile unsigned int * **EventNumberAddress**

volatile unsigned int * **ResetTriggerInformationAddress**

volatile unsigned int * **ResetTriggerLatchAddress**

volatile unsigned int * **NofTotalTriggersAddress**

volatile unsigned int * **ScalerTriggerFreqDivAddress**

volatile unsigned int * **EnableModulesAddress**

volatile unsigned int * **ST_ResetNofSpillsAddress**

volatile unsigned int * **ST_LatchNofSpillsAddress**

volatile unsigned int * **ST_NofSpillsAddress**

volatile unsigned int * **ST_ExtSpillOnAddress**

volatile unsigned int * **SpillTimeAddress**

volatile unsigned int * **SpillStatusAddress**

volatile unsigned int * **TLEnableTriggerLogicAddress**

volatile unsigned int * **TLResetCounterAddress**

volatile unsigned int * **TLLatchCounterAddress**

volatile unsigned int * **TLTriggerCounterAddress**

volatile unsigned int * **TLTriggerAttemptCounterAddress**

volatile unsigned int * **TLDelayValuesRegisterAddress**

volatile unsigned int * **TLInputBitmaskAddress**

volatile unsigned int * **TL2ndRowAndRegisterAddress**

volatile unsigned int * **TLOrRegisterAddress**

volatile unsigned int * **TLAndRegisterAddress**

volatile unsigned int * **TLVetoRegisterAddress**

volatile unsigned int * **TLInputCounterAddress**

volatile unsigned int * **TLPrescalerAddress**

volatile unsigned int * **TLInputPhaseSelectAddress**

A.2.1 Detailed Description

Interface to program the FPGA board and configure the global trigger.

Creating an object will program the FPGA and configure the global trigger with the configuration in the xml file. Any value can be read out or set by the get/read and set functions

A.2.2 Constructor & Destructor Documentation

A.2.2.1 b1GlobalTrigger::b1GlobalTrigger (XMLini * *xmlconfig*, int *number*)

The constructor of this class.

>number< selects the instance, usually it will be zero.

A.2.3 Member Function Documentation

A.2.3.1 unsigned int b1GlobalTrigger::GetTriggerLogicVetoRegister (unsigned int *number*)

Returns bitmask of veto.

>number< is 0 - (max. ORs - 1) for ORs, max. ORs - (max. ORs + max. ANDs - 1) for ANDs

A.2.3.2 void b1GlobalTrigger::SetEnableTriggerLogic ()

enables trigger logic

Sets which internal trigger contributes to the global trigger. Value depends on actual trigger hardware. This function uses the value from the XML file, default 0x00 (all off)

A.2.3.3 void b1GlobalTrigger::SetTriggerLogicVetoRegister (unsigned int *bitmask*, unsigned int *number*)

Sets veto inputs for trigger logic.

Sets the bitmask for the vetos of ORs and ANDs in the trigger logic.

>number< is 0 - (max. ORs - 1) for ORs, max. ORs - (max. ORs + max. ANDs - 1) for ANDs.

A.3 Example Config File of the Global Trigger

```
1 <DAQW>
2 <MODULES>
3 <VME>
4 <B1GLOBALTRIGGER_0
5         ModuleName = "B1GLOBALTRIGGER"
6         ModuleNumber = "0"
7         baseaddress = "0xAAAE0000"
8         binFileName = "/net/homes/hahne/b1TriggerTest/b1GlobalTrigge
9                     r_mk/b1GlobalTrigger.bit"
10        TimeSpillIsOn = "0x0"
11        TimeSpillIsOff = "0x0"
12        ScalerTriggerFreqDiv = "0x0"
13
14        EnableSpill = "0"
15        EnableScalerTrigger = "0"
16 EnableTriggerLogic = "0x1F"
17
18 InputName0 = "test"
19 InputDelay0 = "0x0"
20 InputName1 = "Input1"
21 InputDelay1 = "0x0"
22 InputName2 = "Input2"
23 InputDelay2 = "0x0"
24 InputName3 = "Input3"
25 InputDelay3 = "0x0"
26 InputName4 = "Input4"
27 InputDelay4 = "0x0"
28 InputName5 = "Input5"
29 InputDelay5 = "0x0"
30 InputName6 = "Input6"
31 InputDelay6 = "0x0"
```

```
32 InputName7 = "Input7"
33 InputDelay7 = "0x0"
34 InputName8 = "Input8"
35 InputDelay8 = "0x0"
36 InputName9 = "Input9"
37 InputDelay9 = "0x0"
38 InputName10 = "Input10"
39 InputDelay10 = "0x0"
40 InputName11 = "Input11"
41 InputDelay11 = "0x0"
42 InputName12 = "Input12"
43 InputDelay12 = "0x0"
44 InputName13 = "Input13"
45 InputDelay13 = "0x0"
46 InputName14 = "Input14"
47 InputDelay14 = "0x0"
48 InputName15 = "Input15"
49 InputDelay15 = "0x0"
50 InputName16 = "Input16"
51 InputDelay16 = "0x0"
52 InputName17 = "Input17"
53 InputDelay17 = "0x0"
54 InputName18 = "Input18"
55 InputDelay18 = "0x0"
56 InputName19 = "Input19"
57 InputDelay19 = "0x0"
58 InputName20 = "Input20"
59 InputDelay20 = "0x0"
60 InputName21 = "Input21"
61 InputDelay21 = "0x0"
62 InputName22 = "Input22"
63 InputDelay22 = "0x0"
64 InputName23 = "Input23"
```

```
65 InputDelay23 = "0x0"
66
67 InputBitmask0 = "0x8000"
68 InputBitmask1 = "0x0"
69 InputBitmask2 = "0x0"
70 InputBitmask3 = "0x0"
71 InputBitmask4 = "0x0"
72 InputBitmask5 = "0x0"
73 InputBitmask6 = "0x0"
74 InputBitmask7 = "0x0"
75 InputBitmask8 = "0x0"
76 InputBitmask9 = "0x0"
77 InputBitmask10 = "0x0"
78 InputBitmask11 = "0x0"
79 InputBitmask12 = "0x0"
80 InputBitmask13 = "0x0"
81 InputBitmask14 = "0x0"
82 InputBitmask15 = "0x0"
83 InputBitmask16 = "0x0"
84 InputBitmask17 = "0x0"
85 InputBitmask18 = "0x0"
86 InputBitmask19 = "0x0"
87 InputBitmask20 = "0x0"
88 InputBitmask21 = "0x0"
89 InputBitmask22 = "0x0"
90 InputBitmask23 = "0x0"
91
92 TLOrRegister0 = "0x1"
93 TLOrRegister1 = "0x0"
94 TLOrRegister2 = "0x0"
95 TLOrRegister3 = "0x0"
96 TLOrRegister4 = "0x0"
97 TLOrRegister5 = "0x0"
```

```
98  TLOrRegister6 = "0x0"
99  TLOrRegister7 = "0x0"
100
101  TLandRegister0 = "0x0"
102  TLandRegister1 = "0x0"
103  TLandRegister2 = "0x0"
104  TLandRegister3 = "0x0"
105  TLandRegister4 = "0x0"
106  TLandRegister5 = "0x0"
107  TLandRegister6 = "0x0"
108  TLandRegister7 = "0x0"
109
110  TLVetoRegister0 = "0x0"
111  TLVetoRegister1 = "0x0"
112  TLVetoRegister2 = "0x0"
113  TLVetoRegister3 = "0x0"
114  TLVetoRegister4 = "0x0"
115  TLVetoRegister5 = "0x0"
116  TLVetoRegister6 = "0x0"
117  TLVetoRegister7 = "0x0"
118  TLVetoRegister8 = "0x0"
119  TLVetoRegister9 = "0x0"
120  TLVetoRegister10 = "0x0"
121  TLVetoRegister11 = "0x0"
122  TLVetoRegister12 = "0x0"
123  TLVetoRegister13 = "0x0"
124  TLVetoRegister14 = "0x0"
125  TLVetoRegister15 = "0x0"
126
127  TL2ndRowAndRegister0 = "0x1"
128  TL2ndRowAndRegister1 = "0x0"
129  TL2ndRowAndRegister2 = "0x0"
130  TL2ndRowAndRegister3 = "0x0"
```

```
131  TL2ndRowAndRegister4 = "0x0"
132
133  TLPrescaler0 = "0x0"
134  TLPrescaler1 = "0x0"
135  TLPrescaler2 = "0x0"
136  TLPrescaler3 = "0x0"
137  TLPrescaler4 = "0x35F0F7"
138
139  />
140  </VME>
141  </MODULES>
142  </DAQW>
```


Bibliography

- [Bel99] F. Bellemann. Pion-pion p-wave dominance in the $pd^3\text{he } \pi^+\pi^-$ reaction near threshold. *Phys. Rev. C* **60**, 061002, 1999. 4
- [Bel07] F. Bellemann. Experimental study of the $pd \rightarrow ^3\text{he} k^+k^-$ and $pd \rightarrow ^3\text{he} \phi$ reactions close to threshold. *Phys. Rev. C* **75**, 015204, 2007. 4
- [Bel11] Andreas Bella. *Setup of a Goniometer System for the Production of Linear Polarized Photons for the BGO-OD Experiment at ELSA*. PhD thesis, Physics Institute, University Bonn, 2011. 4, 5
- [Bös] Sabine Böse. *doctorial thesis in preparation*. PhD thesis, Physics Institute, University Bonn. 4, 9
- [ELB11] ELB-Elektroniklaboratorien Bonn UG. *ELB-VFB2 rev 2.1 Versatile VME-FPGA-Carrier Board*, 2011. 27
- [Goe08] Johann Wolfgang von Goethe. *Faust — Eine Tragödie von Goethe*. Tübingen: Cotta'sche Verlagsbuchhandlung, 1808. 1
- [Ham] Daniel Hammann. *doctorial thesis in preparation*. PhD thesis, Physics Institute, University Bonn. 5, 11
- [Ham08] Daniel Hammann. *Test und Inbetriebnahme der Prototyp-Driftkammer für das B1-Spektrometer*. PhD thesis, Physics Institute, University Bonn, 2008. 4, 10
- [IEC94] IEC — International Electrotechnical Commission. *IEC 60617-12 Ed. 3.0 — Graphical symbols for diagrams - Part 12: Binary logic elements*. URL: <http://www.iec.ch>, 1994. 15

- [Leo94] William R. Leo. *Techniques for Nuclear and Particle Physics Experiments*. Springer, 2nd edition, 1994. 19, 20
- [Mes] Francesco Messi. *doctorial thesis in preparation*. PhD thesis, Physics Institute, University Bonn. 4, 7
- [Mon09] J. Donald Monk. The mathematics of boolean algebra. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Spring 2009 edition, 2009. 14
- [Ram07] Alexander Ramseger. *Vorbereitung und Test des Flugzeitdetektors für das Crystal Barrel Experiment an ELSA*. PhD thesis, Physics Institute, University Bonn, 2007. 4
- [Sch10] Timothy Schwan. *Test und Inbetriebnahme der Driftkammern für das BGO-OD Spektrometer*. PhD thesis, Physics Institute, University Bonn, 2010. 4, 10
- [Sie10] Georg Siebke. *Design of the BGO-OD Tagging System and Test of a Detector Prototype*. PhD thesis, Physics Institute, University Bonn, 2010. 4, 7
- [Uni10a] University Bonn. URL <http://www1.cb.uni-bonn.de/>, 2010. 3
- [Uni10b] University Gießen. URL <http://pcweb.physik.uni-giessen.de/taps/index.htm>, 2010. 3
- [W-I08] W-IE-NE-R. *Product Catalog*, 2008. 20, 21
- [Wik06] Wikipedia URL: http://en.wikipedia.org/wiki/File:Switch_box.gif. *Switch matrix*, 2006. 23
- [Wik11] Wikipedia URL: <http://de.wikipedia.org/wiki/Logikgatter>. *Logic gates*, 2011. 63
- [Xil10a] Xilinx Inc. *Xilinx DS099 Spartan-3 FPGA Family data sheet*, 2010. 22

- [Xil10b] Xilinx Inc. *Xilinx UG331 Spartan-3 Generation FPGA User Guide*, 2010. 24
- [Zim] Thomas Zimmermann. *diploma thesis in preparation*. PhD thesis, Physics Institute, University Bonn. 5, 10