NTNU

Norwegian University of
Science and Technology

Event System Implementations for Microcontroller Circuits

Rune André Bjørnerud

Master of Science in Electronics
Submission date: June 2009
Supervisor: Kjetil Svarstad, IET
Co-supervisor: Morten W. Lund, Atmel Norway AS

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Problem Description

The objective of this project is to investigate possible improvements to the XMEGA AVR® Event System. This will entail a study of different implementation alternatives for the Event System, and the proposed solutions will be analyzed with respect to overall system and implementation costs. The study will also include an analysis of the proposed Event System architecture with regards to programmability versus cost.
A selected solution for the AVR® must be implemented using Verilog, and if time allows, the solution will be tested with a realistic system test.

Considering implementation and overall system costs, the following topics will be of primary focus:

- Low implementation costs, especially emphasizing area and power consumption.
- Cost effective design with respect to routing of signals.
- Reuse of existing peripherals, and if possible improve their integration with the proposed solution for the AVR® Event System.
- Cost effective design with respect to tool support.

Assignment given: 15. January 2009
Supervisor: Kjetil Svarstad, IET

Preface

During the last semester all academic work has been focused on my master's thesis to fulfill the requirements for the degree Master of Science, M. Sc, in Electronics. Atmel Norway presented me with an academically challenging task, and gave me the opportunity to work with experienced designers and researchers. Owing to these facts this master's thesis has been an inspiring and interesting academic work, and for that I am grateful to all involved participants.

Given as a relatively open assignment considering which aspects of the Event System that should be target for improvement investigation, I started off with a wide analysis and narrowed focus to the sections which revealed the largest potential for increased Event System functionality and also showed prospects for academic achievements. The result is a seemingly long report in order to give focus to the most important areas of this research, but it is narrowed down to only include the most fundamental theory needed to understand the presented work. Especially the vast field of asynchronous design methodologies is limited to only the most relevant subjects, not presenting a full overview in the scope of this thesis.

Considering the appendices appendix A contains important architectural drawings including area estimations for specific blocks, schematics for custom designed elements and additional simulations to cover handshake procedures.

Sensitive information regarding process related information to Atmel's 35k9 process is included as a restricted appendix B.1. The appendix contains essential background information for area estimations presented in chapter 12.
All Verilog code describing the selected Asynchronous Event System is included in a restricted appendix B.2, and will be referenced in the text only when circuit modules and elements are described. No Verilog related explanations or guidance will be given except in the commented code, assuming that the reader is familiar with basic circuit design at university level and with Verilog constructs and syntax. On an included CD all Verilog design files, workspaces, testbenches and test setups will be available for further inspection. In addition, schematics and address tables are included to define how the test model used for behavioral Asynchronous Event System verification is connected at a detailed level. The address tables and in-depth architectural drawings included on the CD are particularly useful when examining chapter 10.

Appendix B.3 includes a CPLD area estimation relying on restricted library information.

A scientific paper entitled "*Event Control and Programming for Microcontroller Peripheral Systems*" has been written based on the asynchronous research provided by this thesis and a preliminary paper version is included as an appendix C.

Last but not least I would like to give a big thanks to my supervisors professor Kjetil Svarstad (NTNU) and Morten W. Lund (Atmel Norway) for great tutoring during my research. I would also like to thank Vemund Lindås for making it possible for me to write this thesis for Atmel Norway, and PhD student Martin Simlastik for all feedback and comments.

_____ _____

Rune André Bjørnerud Place/ Date

Abstract

To ensure effective peripheral communication on their new AVR® XMEGA microcontroller platform, Atmel has included a peripheral resource known as an Event System. Through the submitted research from this thesis, new solutions for Event System implementations have been investigated and a selected solution has been has been designed in Verilog and synthesized for the Xilinx Spartan 3 XC3S1000 FPGA.

To find inspiration for developing new and cost effective Event System solutions with programmability features, an extensive literature study in the fields of FPGA related design and asynchronous circuit design was accompliced. After proposing two Event System solutions featuring programmable routing with a centralized I/O-processor and CPLD-elements, a decision was made to focus on an asynchronous Event System solution. Interesting features like a pipelined high-speed interconnect structure and possibilities for power saving due to natural power down capabilities where two of the reasons for this focus. Another factor was the academic value of researching if a programmable asynchronous routing structure was applicable for microcontroller implementation, and especially to see if such an implementation could be realistic and beneficial for the AVR® XMEGA or AVR®32. Increased asynchronous focus made it a natural choice to restrict the work towards physical FPGA implementation in order to provide a thorough research and develop a good Asynchronous Event System model.

The first focus for each of the proposed Event System solutions was to develop a new and cost effective routing facility, designing custom switch elements to fit the developed programmable FPGA inspired routing topology. After giving increased attention to the Asynchronous Event System, LUT elements implemented in an asynchronous handshake environment was included to perform logical computations on events. By reusing elements from existing peripherals to include and interconnect LUTs, a powerful computation environment was introduced.

To verify the Asynchronous Event System model and featured asynchronous principles, a Verilog model of the Asynchronous Event System Routing Network including global and local routing resources and a custom asynchronous LUT for event computation was designed. Simulation results verify the Asynchronous Event System model's behavior, and give notions on the system capacity. Synthesis towards the Xilinx Spartan 3 XC3S1000 FPGA was done to check physical implementation size, and make sure a synthesizable design was developed.

Estimated area results approximates an asynchronous Event System to about 5555 NAND gate equivalents without computational LUT elements, and from 6418 NAND equivalents with 2 interconnected 4-LUTs to 11413 NAND gates with a full range of 24 interconnected 4-LUTs. Size increase compared to the current Event System is 23% larger for a non-computational version, and from 43% to 154% larger for 2 and 24 included 4-LUTs respectively. Included in the size increase is programmable routing demanding 924 bits for global and local routing only, and 1724 bits for full programmability with 24 4-LUTs. Simulations show a possibility for transporting up to 32 events concurrently, which is an increase of four times the current Event System capacity. Performance evaluations put the developed Asynchronous Event System from 6.5 – 1.6 times faster than the current Event System estimated for 90nm – 350nm processes. Considering HERN area estimations, a simple I/O-processor alternative consumes 3150 NAND gates included programmable routing. A CERN solution featuring one CPLD MCB for each Port / TC peripheral requires 5308 NAND equivalents. All area estimations are based on developed architectural drawings, and are estimated according to the Atmel's 35k9 in-house process, thereby not representing a synthesized result.

Contents

List of Abbreviations

Abbreviation	Details
4-PDR	4- Phase Dual Rail
AESRN	Asynchronous Event System Routing Network
AGERN	Asynchronous Global Event Routing Network
ALERN	Asynchronous Local Event Routing Network
ALFERN	Asynchronous Local Functional Event Routing Network
ADC	Analog-to-Digital Converter
CAD	Computer Aided Design
CERN	CPLD Event Routing Network
CPLD	Complex Programmable Logic Device
CPU	Central Processing Unit
DAC	Digital-to-Analog Converter
DI	Delay-Insensitive
DMA	Direct Memory Access
EDA	Electronic Design Automation
ERN	Event Routing Network
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GUI	Graphical User Inteface
HERN	Hierarchical Event Routing Network
I/O	Input/Output
LAB	Logic Array Block
MCB	Macrocell Block
MTIO	Multifunctional Timer I/O
OBP	On-Board Peripheral
QDI	Quasi Delay-Insensitive
PCHB	Pre-Charge Half Buffer
PAL	Programmable Array Logic
PLA	Programmable Logic Array
RTC	Real-Time Counter
SI	Speed-Independent
SPLD	Simple Programmable Logic Device
SRAM	Static Random Access Memory
TCCB	Timer Counter Computational Block
TCFB	Timer Counter Functional Block
TCCFB	Timer Counter Combined Functional Block
TCT	Total Cycle Time
TVDM	Token Vector Delay Model
TV	Token Vector
WCHB	Weak-Condition Half Buffer

List of Figures

List of Tables

1 Introduction

1.1 About This Thesis

The demands for functionality integrated on a microcontroller have rapidly increased since the introduction of the first RISC architectures. The dominant factors pushing the development towards more and more aggressive processes are demands for smaller circuit size, faster computation speed and decreased power consumption. Demands in these areas become more prominent as more and more advanced portable devices are introduced, trying to accelerate performance while decreasing in size. As a response to these demands Atmel has developed the AVR® XMEGA microcontroller platform, introducing the Event System as a new invention in the microcontroller domain. Making it possible to route and process signals known as events from internal peripherals without the use of CPU, DMA or interrupt resources [6], the Event System leads on in a new era of power saving computation systems.

In order to make the AVR® XMEGA a powerful microcontroller when it comes to handling peripheral and I/O related resources, the Event System features a dedicated Event Routing Network with connections to peripherals and I/O-pins on the chip. Maintaining such an extensive communication using a minimum of CPU resources has proved to be a valuable addition to all systems relying on heavy I/O traffic and extensive peripheral communication.

Although innovative and in many ways groundbreaking, the XMEGA Event System is implemented with limited resources for logical event computations and mostly offer a routing facility for peripheral events and signals. This thesis aims at researching possible improvements to the original Event System [6] by developing a more flexible infrastructure for event transportation, and increase the functionality with regards to event computation while still maintaining low power consumption. Such a research is valuable, because if successful, more advanced event operations can be performed on the XMEGA microcontroller involving a minimum of additional resources from the CPU. Through an extensive literature study, different techniques to ensure a cost effective Event System solution both with regards to power consumption, programmability and routing of signals will be investigated. Focused research areas include FPGA topologies for routing, programmable computational elements like Look-Up-Tables and asynchronous circuit design. With some of these elements as part of a new Event System solution, the goal is to develop a new benchmark for flexible routing and programmable computational power seen in a commercial microcontroller Event System.

To ensure cost effectiveness of the selected implementations, a cost analysis of all proposed Event System solutions is an important part of this study. Of special interest is the influence of a new Event System design on the existing tool chain, and a brief overview of suggested alterations and additions to the existing tools will be given along with the cost analysis. The analysis itself will contain cost factors like area, power consumption, ease of use, programmability and testability to ensure a complete evaluation of all Event System solutions.

To prove the concept of the new Event System, a selected solution should be designed in Verilog and synthesized towards a target FPGA platform. Main design focus is given to a new routing infrastructure, but elements offering logical computations for events are also interesting candidates from a design perspective. Simulations will be used for behavioral verification of a selected Event System solution. Conclusions based on both simulation results and cost- and architectural analysis will clarify the future of the selected solution as a realistic Event System alternative for the XMEGA microcontroller. Researched aspects of all proposed Event System solutions will be examined at the lower abstraction levels, mostly concerning hardware.
Higher abstraction levels including development of software is not a part of this thesis, and the suggested tool improvements will be the only analysis superficially concerning this level of abstraction.

1.2 Thesis Contributions and Layout

The following points will summarize the contributions from this thesis:

- Three novel concept architectures for a new Event System solution with programmable interconnects.
- Custom designed programmable switch blocks for full routing flexibility operating in an FPGA inspired environment.
- An asynchronous Event System solution with full asynchronous event distribution in a pipelined hierarchical routing topology designed for microcontroller implementation.
- Asynchronous event computation with LUTs included in existing Port / TC peripherals.
- Performance and area analysis of all proposed solutions.

Considering layout, this thesis is partitioned into four main sections consisting of several subsections. Section 1 will give necessary background theory and put the presented work into the context of other research in the same area. Section 2 presents all proposed Event System solutions, and section 3 explains aspects related to simulation results of a selected Event System solution and summarizes synthesis results. Section 4 concludes the report by showing performance and area related cost factor estimations for all solutions. Aspects regarding market related cost factors and tool improvements are also included in section 4, along with the final conclusion and discussion.

Section I –

Background Theory and Related Work

2 The AVR® XMEGA Event System

The Event System is a unique distribution system for handling peripherals interconnected on a microcontroller. This section will provide the background needed to understand the purpose of the Event System, how it works in practice and why it is a useful addition to the AVR® XMEGA microcontroller family where it is implemented. The last subsections will give a brief overview of why an increased functionality and flexibility considering the Event System is desirable, and how this master thesis will contribute in that area of research.

2.1 Event System Functionality and Construction

The continuous fight for lower power consumption, increased functionality per cost and flexible means of inter peripheral communication on a microcontroller drives the development of more subtle and advanced microcontroller subsystems. With the introduction of the Event System the AVR® XMEGA responds to these demands. Basically the Event System is a communication facility for inter peripheral communication, also offering a set of features which can be distributed by the same communication infrastructure. An event is issued when a peripheral changes its current state, and the different features available are controlled by the peripherals receiving the event. Feature examples include possibilities for event decoding and filtering. The routing of events is handled by the Event Routing Network (ERN), connecting to most of the available peripherals. Figure 2.1 shows which peripherals that can take part in event transactions. For more detailed descriptions on each peripheral, the reader is referred to the XMEGA A manual [6].

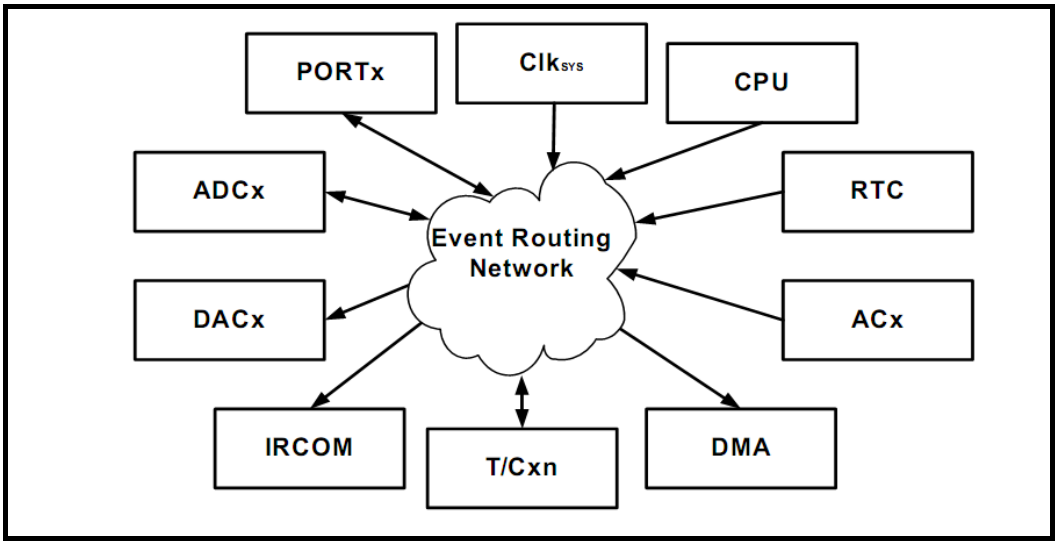


Figure 2.1: All peripherals connecting to the ERN [6]

By using events to trigger actions in the peripherals, functionality can be achieved without the use of CPU, DMA or interrupt resources. This achievement is one of the chief motivations for the Event System, along with the fact that the ERN is available for routing arbitrary signals from all connected peripherals. Connecting the Ports together with dedicated Timer/Counter (TC) modules provides processing of signals sent by I/O-pins, with the possibility of utilizing the ERN as a distributor of external clock- and decoded data signals. A note to the presented event notation considers the strobe register mentioned in the XMEGA manual [6]. The strobe register represents the functional equivalent of the *clock* bit used for event description in this thesis. This means that in the XMEGA manual, an event channel will not be described in terms of *ev_d* and *ev_c* wires, as this represents a more practical model. The functionality is however equivalent for both notations.

2.1.1 Events and the Event Channel

As mentioned in the previous section an event is referred to as a state change in a peripheral observable as either a signaling event or a data event. A signaling event holds no information, and merely informs selected peripherals that an event has occurred in some other peripheral.
A data event contains some additional information, for instance a decoded sequence of bits or a clock signal [6]. Each event is described by a combination of two bits, called the *event clock bit* and the *event data bit*. Each bit is distributed on one wire, and the Event Channel is therefore constructed of the *ev_c* (clock) and *ev_d* (data) wires.

When an event is received at a peripheral, it is interpreted in accordance with the applied register settings and by the event bit combination. The interpretation is peripheral specific, but an example from [6] considering the TC peripheral is included in table 2-1. A note to the table is that the strobe bit notation is used, instead of the clock bit notation.

STROBE	DATA	Data Event User	Signaling Event User
0	0	No Event	No Event
0	1	Index/Reset	No Event
1	0	Count Down	Signaling Event
1	1	Count Up	Signaling Event

Table 2-1: TC event interpretation [6]

It is important to emphasize that the event notation presented specifically for the Event System is developed by Atmel, and must not be confused with the notation used for general events presented in scientific literature. In this thesis events distributed on the Event System will follow the Atmel notation presented in this chapter.

2.1.2 Event Routing Network

All event distribution is maintained by the ERN. To determine which event should be received in which peripheral, a set of eight MUXes are connected to eight global event distribution channels as described in section 2.1.1. All possible Event channels from all peripherals are connected to all MUXes, providing full routability between peripherals. Figure 2.2 shows the ERN.

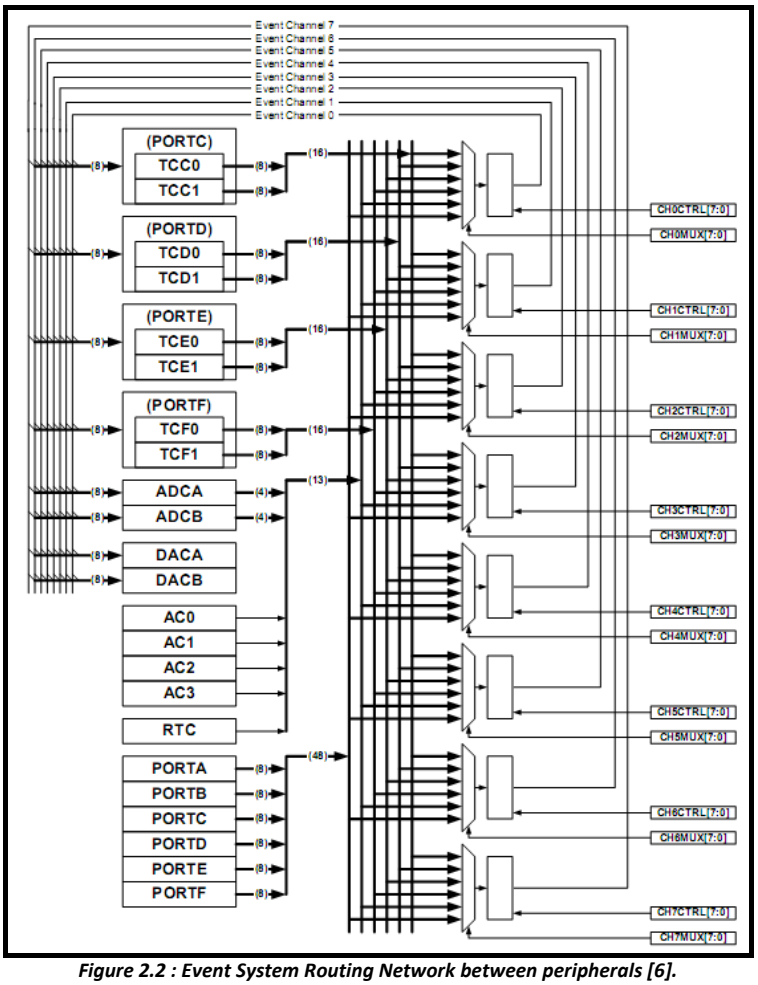


Figure 2.2 : Event System Routing Network between peripherals [6].

Each MUX is controlled by a channel select register $CH_xMUX[7:0]$, and the global event channels connect to all peripherals available for the specific XMEGA. It is important to note that not all XMEGAs contain all peripherals.

2.2 Why Use an Event System?

It is a fact that with the introduction of the Event System to the XMEGA microcontroller family, new costs as well as new functionality was introduced. A consideration of these factors is therefore important when determining additions to the existing Event System.
The biggest achievement of the Event System technology is independency of CPU, DMA and interrupts during event distribution. This independency means less power consumption during transfer, and makes the XMEGA a powerful microcontroller when handling peripheral data transfer and control. Another benefit is that the CPU can be used to execute other instructions while the Event System is operating. Connecting to all available peripherals the ERN can also be used as a routing network for external I/O- signals on the microcontroller, and distribute these at a lower cost than using data- or I/O bus transmission.

The current Event System described in [6] uses the ERN merely as a facility for routing a signal from a transmission point A to a destination point B. No logical operations can be done on events during transfer, and when an event is issued it will remain in its original form until it reaches the target destination. As can be concluded, the current ERN is a rather area expensive routing facility which offers a limited flexibility when it comes to event channel routability and connectivity.

This fact is a result of limiting the ERN to only global event distribution, and the locality of peripherals with more extensive need for local communication is not considered. With these benefits and drawbacks as basis, a more specific outline of the research contributed by this master thesis can be given.

2.2.1 Improvements to the Event System

As mentioned in the introduction the focus of this thesis will mainly be to make the Event System more flexible and powerful regarding handling of events during transfer, while maintaining minimal power consumption. The challenge is to achieve these goals while still maintaining a reasonable complexity of the system to limit new additions to the existing tool chain, and also keep the associated cost profitable compared to the offered functionality.

To ensure that events can be routed arbitrarily between peripherals at both a global- and a more local level between neighboring peripherals, a new routing facility will be introduced. Inspired by FPGA topologies, this routing facility will feature fully programmable routing of event channels and represent a hierarchical solution for increased functionality. Custom designed switches will be introduced for the routing system to make it intuitive and simple for the user.
Another FPGA feature that will be investigated is the use of Look-Up-Tables (LUTs) for logical event computation. In this way the Event System can not only distribute events, but also compute all logical operations on the events. Such functionality allows more advanced patterns of events to exist, enabling possibilities for more sophisticated peripheral operation. While still maintaining the low power demands of the original Event System, these are powerful additions.
The proposed solutions are also developed for easy tool chain adaption.

An important comment to the developed Event System solutions is that the XMEGA A manual [6] defines the architecture used as basis for the new Event System designs. This means that smaller XMEGA devices will not contain all peripherals described in the presented research, but this imposes no limitations on adding the equivalent functionality.

3 Asynchronous Design Techniques: Implementations and Challenges

Asynchronous circuit design is a domain not usually applied when constructing microcontroller circuits. Some of the proposed research for the Event System includes a wide variety of asynchronous elements and design techniques not earlier used in the presented context. This section will provide the fundamentals for understanding how asynchronous implementations can be beneficial, present common challenges of implementation and explain the basic circuit elements. The first subsections will consider asynchronous design in general and focus on benefits and challenges using presented design techniques. Protocols and encodings used to provide asynchronous handshake facilities are discussed next, before this chapter is completed by explaining basic asynchronous circuit elements and a simple asynchronous pipeline.

3.1 Asynchronous Circuit Models

With no clock driving the signaling procedure in asynchronous circuits, notations of how to handle delays in different circuit elements like wires and gates need to be determined. The different assumptions made about these delays characterize each circuit model based on their corresponding delay model [29]. Here only the most common models and the ones relevant for this thesis will be presented.

Delay-Insensitive (DI) circuit models
DI circuits impose no timing assumptions, and therefore allow arbitrary gate and wire delays [29]. If arbitrary delaying a circuit's inputs and outputs does not produce an output hazard or alter the sequence of involved tokens, the circuit can be classified as delay insensitive [13]. Another definition used in [22] classifies a circuit as delay-insensitive if it can function correctly with arbitrary gate and wire delays. The delay is considered unbounded with the delay bond of $0 < t_{delay} < \infty$ in both wires and gates [22]. This assumption requires completion detection for all signals in a DI-circuit, normally meaning two-phased handshaking with *request/acknowledge* signals. Using DI-models could derive timing benefits in datapath circuits, but the needed redundancy to ensure correct data transfer leaves DI-circuits little used in practice. Also mentioned in [22] is that only a small class of circuits, mostly constructed by Müller C-elements, can be classified as DI-circuits.

Speed-Independent (SI) circuit models
This model assume that all wire delays are negligible compared to gate delays which are considered unbounded [13] [29]. SI-models are impractical in designs where wire delay dominates the logic delay, as is often the case for FPGAs. The SI assumption could also be difficult when considering I/O and off-chip signals with long transmission wires. These drawbacks are a result of the SI-assumption, where operating with ideal zero-delay wires are not realistic in practical circuits [22]. The SI-model is however more practical than pure DI-models, mostly because of the reduced need of costly redundancy.

Quasi-Delay-Insensitive (QDI) circuit models
QDI- models depend on DI-assumptions considering wire and gate delay [13] [22], but partitions wire delays into critical and non-critical delay. Critical wire sections arise if a signal is forked into several branches, and the QDI assumption is that the difference in arrival time between different branches are smaller than the minimum gate delay. Delay in non-critical wire sections are considered unbounded. The forks forming the critical wire segments are called *isochronic forks* [13] [22] [29]. The QDI circuit model will be used for all of the asynchronous designs considered in this thesis. Figure 3.1 provides an example of a small QDI circuit.

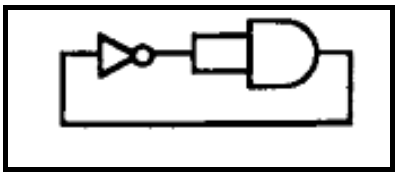


Figure 3.1 : Simple QDI example [13].

This circuit is considered QDI because even if the AND-gate will begin processing when the first input arrives, the second input will arrive before the AND-gate commits a transition. This is the important property of QDI assumptions, and accomplished because the fork is isochronic. Another observation with this property in mind is that the QDI assumption is not valid if the AND-gate is replaced with an XOR-gate. This is because the output is different when one input change compared to when neither or both inputs change, creating an output hazard.

3.2 Hazards Associated with Asynchronous Designs

Many benefits can be associated with an asynchronous design, but there are also some problems present that will not occur if a synchronous design is used instead. Problems related to glitches in signals due to asynchronous behavior are addressed as hazards, and a good overview is presented in [13] and [22].Examples of a typical hazard encounter is when a state machine is implemented asynchronously. Due to internal delays in logic elements, false glitches with non-valid values can be produced during a signal transition. An important remark from [22] is that in a stable state an asynchronous circuit does not produce any glitches. Only in the dynamic state can glitches be produced.
A modified example from [13] can be seen in figure 3.2, illustrating multiple hazards. All logic elements are assumed to have a delay of one unit for signals.

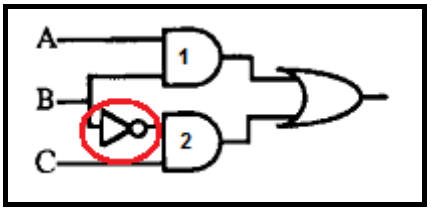


Figure 3.2: Circuit with hazards [13].

Static Hazards
The first hazard problem is created by an input-wire with fan-out, where one of the fan-out wires is connected to an inverter and the other directly to another logic gate constructing a non-isochronic fork. The state (A,B,C) = (1,1,1) gives an output value of “1”, and a change to the state (A,B,C) = (1,0,1) should leave the output value unchanged. However, due to the delay in the inverter (marked red), the AND-gate marked 1 will consider the input as false before the AND-gate marked 2 becomes true. Therefore the value “0” will propagate to the output instead of the correct “1”, producing a temporary glitch. A glitch like this is categorized as a *static-1 hazard*. A similar glitch when the output should be “0”, but produces a momentary “1”, is called a *static-0 hazard*. These hazards are relatively simple to remove for small circuits, and logic transformations with a Karnaugh map minimizing overlapping terms are often sufficient. An example on how to use such techniques in practice is provided in [30], during the design of a hazard free Müller C-element.

Dynamic Hazards
Dynamic-hazards are created when a single transition i.e. 1→0 should take place, but instead multiple transitions 1→0→1→0 occur due to glitches. Such hazards are usually encountered if multiple inputs change to produce a new output, and the delay to some gate is greater for one of the input wires. Dynamic-hazards can be removed in the same fashion as static-hazards [13].
Figure 3.3 shows a dynamic-hazard example presented in [22].

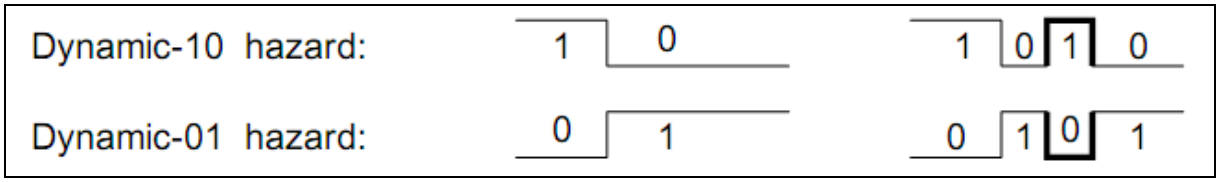


Figure 3.3: Examples of dynamic hazards. Desired transition (left).Hazardous transition (right) [22]

The Karnaugh map optimization technique is only valid and useful when one input is allowed to change at a time. If multiple input changes, a hazard free operation is no longer guaranteed. This fact makes dynamic hazards troublesome, and the only safe way of dealing with such hazards is to enforce that only one input change at a time.

3.2.1 Metastability

While most of the static- and dynamic hazards encountered by asynchronous signaling can be removed with optimization techniques and smart design, the hazards categorized as meatastability-hazards can never be fully removed and will always be present in the interface between synchronous and asynchronous logic. The definition used in [23] describes metastability as the state encountered when an asynchronous signal arrives such that the receiving synchronizer latch is not allowed enough time to stabilize before the output value is clocked. In [25] it is stated that a synchronization failure occurs when a Flip-Flop (FF) with logically undefined outputs is sampled. Practically this means violating the setup and hold constraints provided by the latch. An incorrect output may be produced and introduce faults to the circuit, in worst case participating in circuit failure. If the synchronizer latch receives the asynchronous signal within the time it takes for the latch to stabilize its output, then no fault is produced. An example from [25] illustrates some important principles of metastable behavior.

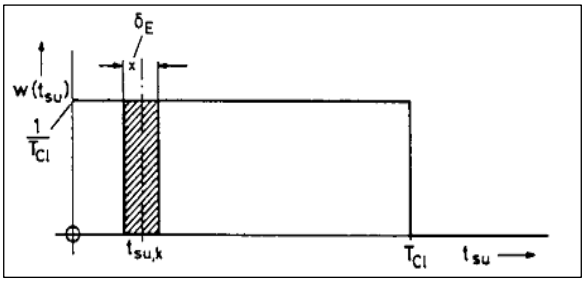


Figure 3.4: Critical trigger window [25]

In figure 3.4 δ_E denoted the critical window, and $t_{su,k}$ is the critical D-FF setup time. If the timing constraint $t_{su,k}$ is violated during the critical window δ_E, the D-FF output will be unresolved, or in a metastable state, after a given decision time t_E. Expressing δ_E as a function $f(t_E)$, a distribution for metastability can be derived. Parameters included in t_E are FF dependent.
See [25] for details.

Clearly the biggest problem with metastability is the potential for a normally functional circuit to suddenly produce a faulty output. Metastability-hazards are also very hard to find during testing, therefore synchronizers must be produced in such a way that the probability of introducing a meta-stable state to a circuit element is satisfyingly low. The hazard is avoided if the setup- and hold periods for synchronizers are never violated [23].

Several papers [24] [25] [26] introduce different techniques to encounter metastability issues. In [24] Ran Ginosaur analyzes several synchronizers proposed in recent years, and emphasizes their problems and how to solve them. In addition a secure two-stage D-Flip-Flop is presented to synchronize data transferred between two domains using a bundled data protocol with *request/ackowledge*. A custom synchronizer is constructed in [26], not relying on Flip-Flops but relying on filter stages to filtrate out the metastability issue. This construction is designed to operate under QDI assumptions, and synchronizes a single-rail signal with an associated control signal to a

dual-rail output. In [25] a double D-Flip-Flop formation is used for synchronization, making this one of the most accepted synchronizers in literature.

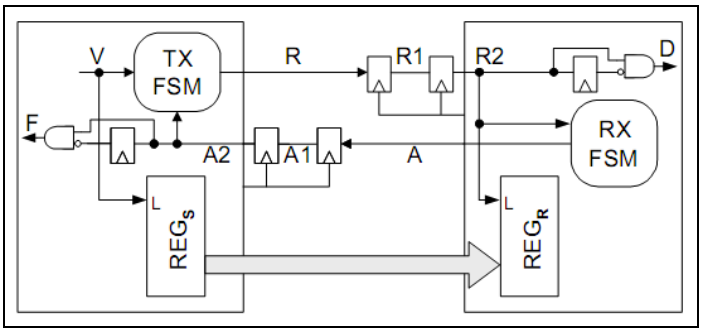


Figure 3.5: Two-Flop synchronizer for a Four-phase Bundled data protocol [24]

Figure 3.5 shows a principle sketch of a two-flop synchronizer considered in [24]. The bundled data request and acknowledge procedures are controlled by two simple state machines implemented on the transmitting and receiver side. For a four-phase dual-rail protocol a different interface would have to be implemented. An overview of asynchronous transfer protocols are given in section 3.4.

For the two-flop solution presented in [24] a MTFB (Mean Time Between Failures) are calculated to be 10^{204} years using conservative numbers in a System-on-Chip setting with two different clock domains. Dealing with totally asynchronous domains distributed with a uniform probability function, where the phase of the system clock and arriving data is uncorrelated, the chance of encountering a metastable state is higher than for synchronous domains where it is possible to sample a signal without encountering metastable behavior [53]. A two-flop synchronizer experiment with asynchronous domains conducted in [25] reveals an MTFB of 9.5 years in the worst case scenario. Both of the above examples show prospects of construction good synchronizer circuits based on the two-flop design philosophy.

3.3 Benefits and Challenges Considering Asynchronous Designs

In modern circuit construction synchronous design- and layout methodologies dominate the market, leaving several aspects of asynchronous circuit design more an area of research than an area of manufacturing. There are several reasons for this, but the most accepted factors are poor Electronic Design Automation (EDA) - and Computer Aided Design (CAD) - tools support together with completely different design methodologies than synchronous designs [13] [29]. However, the benefits could be potentially large considering an asynchronous implementation for specific systems. In [13] Scott Hauck presents a general overview which will be partly presented in the next subsections.

3.3.1 General Benefits

Timing and performance

As presented in section 3.1 asynchronous circuits have no assumptions of clock driven timing. This factor eliminates the need for a clock tree, and the problem of clock skew between signals arriving in different parts of a circuit. Synchronous systems must take clock skew under consideration, and often slow down the circuit performance to meet clock skew demands. Synchronous circuits also need to wait for all computations to finish before latching a result, leading to worst-case timing performance. Asynchronous systems can sense when a computation is finished, and therefore allows average-case timing assumptions. Timing issues considering global timing is not present in asynchronous systems, in contradiction to synchronous systems where the critical path determines the highest clock speed achievable. This fact is beneficial in systems where rarely used portions of the design need to be optimized in order to meet the demands created by the critical path for a

synchronous design. In [29] it is emphasized that asynchronous systems utilize new and aggressive process technologies better than synchronous designs, because of direct adaption to new delay conditions. This enhances performance compared to synchronous circuits, where variable factors as a result of production inaccuracy must be taken into account when determining the clock frequency. This is also an example of asynchronous circuits` adaptability properties.

Power consumption and environment adaption
Synchronous designs need to toggle the clock lines in all parts of the circuit, not only the parts used in a current computation. With applied CMOS technology this leads to a lot of switching in unused transistors, resulting in unnecessary power consumption. An asynchronous design involves more activity during a computation due to applied handshaking, but unused parts of the circuit will not consume energy because of toggling. In [1] this is referred to as event-driven energy consumption.

Changes in the surrounding environment are critical for synchronous circuits, where a worst case assumption in the changing of factors must be taken into account. The robustness represented by asynchronous circuits is gained by automatic adaptability to the changes occurred, and thereby utilizing the new conditions as much as possible [1]. For a purely asynchronous computation of mutual exclusion metastability is not an issue, since a response from a circuit part in a metastable state can delay for an arbitrary length of time without jeopardizing the reception at the receiving element. In a synchronous system guaranteed mutual exclusion is subject to a bounded delay, and can fail if a metastable state occurs. The same factor also favors asynchronous circuits when dealing with inputs from the outside world, which by nature can be considered asynchronous.

3.3.2 Drawbacks and Challenges

The benefits mentioned in section 3.3.1 do not come unconditionally, or asynchronous systems would probably have a larger share of the commercial market for electronic circuits. As mentioned the lack of good EDA- and CAD tools support are a major contributor to the synchronous dominance. Current tools do not support an asynchronous design flow at all, or yields unoptimized designs [13]. Especially for FPGA systems it is difficult to map an asynchronous description using conventional tools to a clocked FPGA architecture, without altering the mapping software [4]. The work presented in [30] emphasize how ordinary place and route tools are not applicable for creating hazard free designs, even if an optimized and hazard free rtl model are targeted for FPGA implementation. One problem is that isochronic forks must be balanced by the synthesis tool for QDI-circuits, a feature not supported by tools designed for synchronous layouts. The proposed solution in [30] is therefore to design and hand map a library of asynchronous cells for implementation, and interface this library towards the existing place and route tools.

In practice this means that asynchronous designs are hard to do in ad hoc fashion, and that the designers must pay a lot of attention to the dynamic behavior of the system in order to avoid the hazards presented in section 3.2 [13]. In the context of the Event System hazards involving metastability can become prominent because of the interface towards synchronous peripherals. This fact leads to the issue of testing and verification of asynchronous systems, which usually cannot be done with conventional Scan Testing. Some comments on testing can be found in section 4.2.

3.4 Asynchronous Handshake Protocols

Considering FPGA implementations, the need for a powerful routing facility is always one of the most prominent issues.
If asynchronous routing facilities are to be successfully implemented, some mechanism for assuring that the asynchronous signals reach their appointed destination must be taken into consideration. This section will give an overview of the most common protocols used to implement such facilities, and also compare the different protocols when it comes to power consumption, signal overhead and ease of implementation. According to [2] there are two groups of handshake protocols that are usually considered for asynchronous circuits; *4-phased protocols* and *2-phased protocols*. These protocols are often used as either 2- or 4-phased bundled data (2/4-PBD) protocols or 4-phased dual-rail (4-PDR) protocols [22].

3.4.1 Two- and Four-Phase Bundled Data Protocols

Bundled-data protocols use an additional two wires per signal to ensure a correct handshake procedure in addition to the data wires. As depicted in figure 3.6 (left) the extra wires perform *request* and *acknowledge* services. If a 4-PBD protocol is used, signal levels on the *request(req)* and *acknowledge(ack)* wires during a transfer are used to secure correct handshaking. Figure 3.6 (center) shows the transition pattern for a four-phase protocol, with two *req/ack* comparisons for each data value transmitted on the channel.

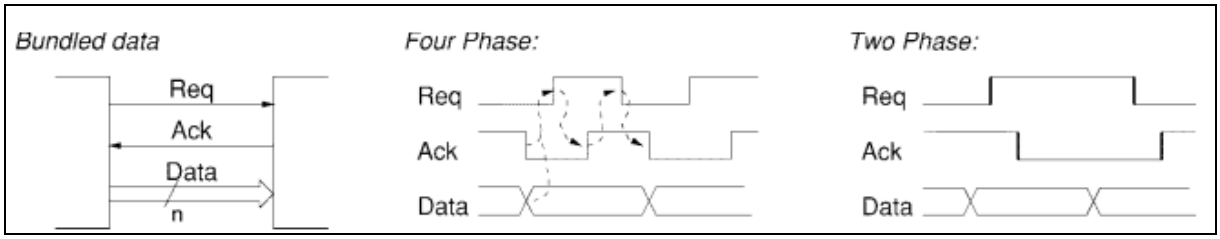


Figure 3.6: Signal transitions: Bundled-data channel (left). Four-phase protocol (center), Two-phase protocol (right) [20]

The 2-PBD protocol uses a similar but simpler handshaking scheme. Here only one *req/ack* is accomplished for each data transfer, making this protocol similar to that used in synchronous designs. The wire overhead is the same as for a four-phase protocol, and a description of the signal transitions is given in figure 3.6 (right).

As mentioned in [2] and [16] both protocols suffer from worst case timing behavior due the timing assumption that the data must be valid when a request is issued. Due to different delays in the *req/ack* and *data* wires the timing violation "*data before request*" can occur if the delay differences are not taken into consideration. To always make the timing assumption valid, the *req* signal should use a matched delay that is larger than the senders computation delay for data plus some margin [16].

One of the benefits of bundled-data protocols are only a small logic- and wire overhead when implemented. As figure 3.6 shows, the original data lines are used for data transfer, adding only two additional wires per channel for *req/ack* procedures. Thus the total wire overhead is n + 2 for *n* data wires per channel. Another benefit described in [2] is the possibility of utilizing reduced signal transition activity on the data wires, thereby reducing the associated energy consumption.

3.4.2 Four-Phase Dual-Rail Protocol

The 4-PDR protocol combine the encoding of data and request by using two wires to represent one data bit. Since the request for data is encoded in the signaling process on the data lines, the only extra wire needed to represent the handshake mechanism is an *acknowledge* wire [2] .
Figure 3.7 illustrates the signals needed in a transfer. In [1] an *enable* wire is used instead of an *acknowledge* wire to provide the handshaking. *Enable* is the inverse of *acknowledge*, and the implementation needs less logic to provide handshake facilities.

Comment	d1	d0
Empty	0	0
Valid “0” bit	0	1
Valid “1” bit	1	0
Not used	1	1

Table 3-1: Four-phase dual-rail signal encoding

That the protocol is four-phased indicates that two signal transitions of *req/ack* are performed during one data transfer. To obtain a unique encoding of the data signals, similar bit values for both data wires at the same time does not encode a valid bit value. In practice this means that only one data wire can change value at the same time, in order for the transition to be considered legal. Table 3-1 indicates the bit encoding used in 4-PDR protocols. Figure 3.7 (left) shows the Dual-Rail channel with an *acknowledge* wire providing handshake, while figure 3.7 (right) shows the signal transitions during a handshake procedure.

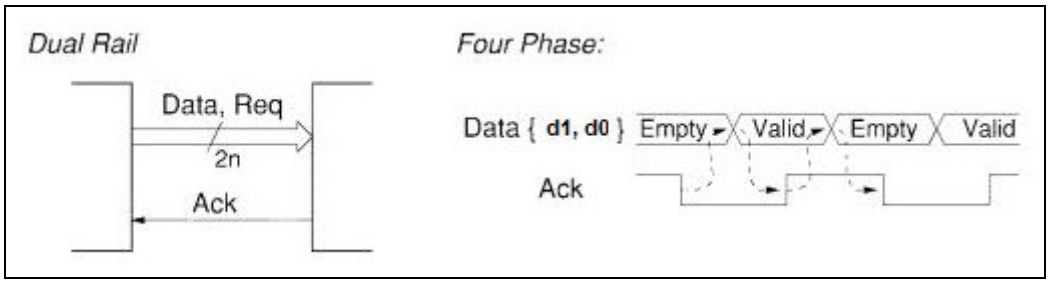


Figure 3.7: Dual-Rail handshaking. Dual-rail channel (left), 4-PDR protocol (right)

One clear drawback of this method is the addition of one extra wire for each original wire sending data bits in a channel, and also the extra *ack* wire. Compared to the n+2 extra wires needed by a 4-PBD protocol, the 4-PDR protocol needs 2n+1 wires, representing a significant wire overhead. However, due to the encoding of request and data by two wires, the 4-PDR protocol is insensitive to delay differences between *data* and *request* wires. This is an advantage over the 4-PBD protocol, and represents an increased robustness against timing errors caused by different delays in the circuit [22]. More complex handshaking circuits for the 4-PDR protocol requires more logic gates than implementations using 4-PBD, and since one of the data wires always will represent a signal transition, reduced switching activity cannot be exploited. The effect is generally higher power consumption than bundled-data protocols [2] .

To indicate that a handshake is completed a non-code word known as a *spacer* must be transmitted to the receiving handshake element, before the next valid data token is transmitted. Typically the value {00} on both data rails is used as a spacer, and the signaling protocol for dual-rail circuits is in this case denoted Return-To-Zero handshake protocol [1] [33]. A point emphasized by [29] is the assumed condition that two consecutive data tokens are separated with a spacer. This is important because data validity by examining the data wires can only be certain if spacers are inserted.

Transmission pattern using the 4-PDR protocol

To show how the protocol is executed in the constructed FPGA pipeline from [1] with the usage of WCHB-elements in the switch blocks, a simple example can be illustrated in figure 3.8.

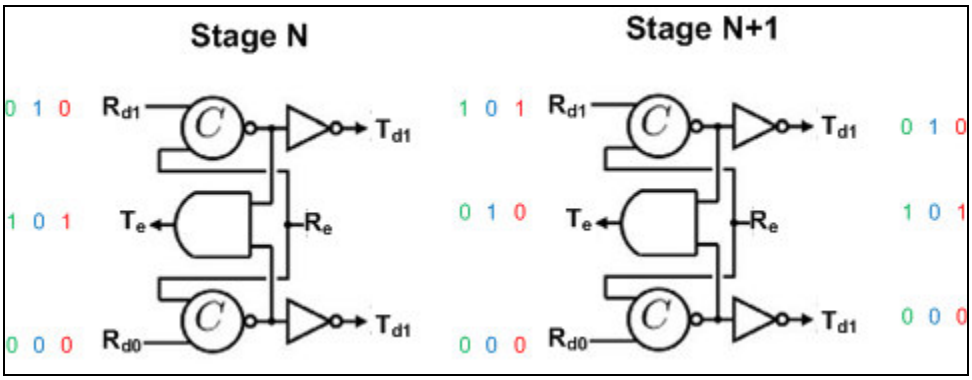


Figure 3.8: Simple 4-PDR transition example

The red sequence marks the initial sequence, when stage N receives the reset token, or spacer, from stage N-1. Stage N+1 has the data token from stage N on its inputs when stage N receives the reset. Because T_e from stage N+1 is set to "0" when data is received, T_e is set to "1" when the reset token arrives. The pipeline state indicate that stage N is free for a new data token arrival on one of its rails, while stage N+1 is busy until it receives the reset token from stage N. The blue and green sequences indicate another token transaction. The protocol applies to both dual-rail lines, not only R_{d1} as used in the example. The WCHB element used to illustrate the handshake procedure is described in section 3.5.2. Instead of the conventional *ack* signal usually applied in 4-PDR protocols, an *enable* signal representing the logical inverse of an *ack* is used in the pipeline. As in the pipeline from [1] an *enable* signal will be used in the proposed Asynchronous Event System solution.

3.5 Asynchronous Circuit Elements

Asynchronous circuits contain certain elements that are not found in synchronous circuit design. In this section synchronization elements for asynchronous signals will be discussed, emphasizing areas of use and different implementation alternatives for each element. The descriptions given in subsequent sections will provide the basis for further argumentations as to choosing the right synchronization architecture for different Event System solutions. First the most basic of all asynchronous elements will be described: the Müller C-element. Constructed by Müller C-elements buffer elements for asynchronous pipelines, copy-elements and a basic Micropipeline structure will be presented in the subsequent sections.

3.5.1 The Müller C-element

Much research has been done on the C-element and its design in order to achieve the most efficient implementation layout when it comes to power consumption and hazard avoidance. Several papers address this matter, among them [22] [30] [31], proposing different solutions for handling these challenges.
Figure 3.9 and figure 3.10 shows two implementation variants proposed by [30] and [31].

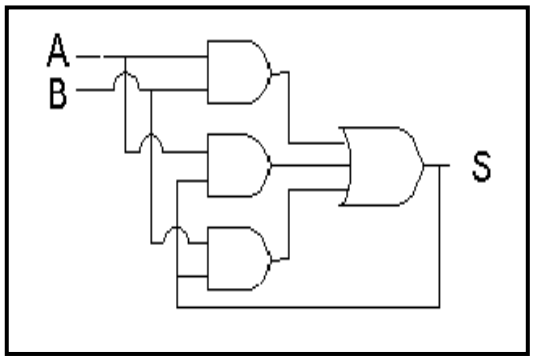


Figure 3.9: Hazard free Müller C-element [30].

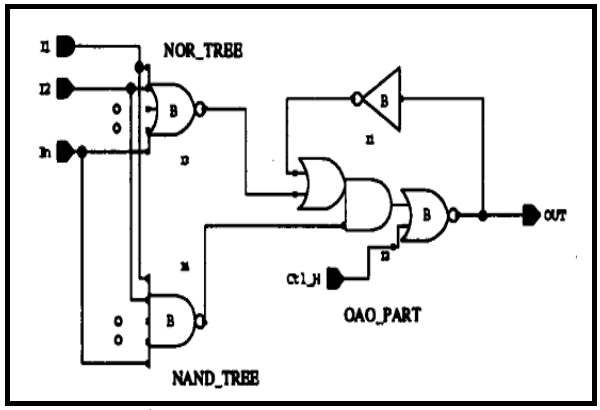


Figure 3.10: AND / NOR- tree implementation of Müller C-element [31].

Basically a Müller C-element is either used to join signal transitions or for completion detection in asynchronous circuits. This functionality includes stateholding in pipelined circuits, for instance encountered in asynchronous FPGAs awaiting handshake signals from one or multiple sources [1] [13] [27] [28] [30] [31]. C-elements are also the fundamental circuit element in Sutherland's Micropipeline [40]. Functionally the C-element is equivalent to the SR-latch, but it can handle the unbounded delay assumptions used in QDI circuits and other asynchronous circuit models, a trait not achievable by an SR-latch. When both inputs are "1" the output of a C-element is "1". If both inputs are "0" the output is "0". For input combinations "10" or 01" for a two input C-element the previous

output value will be transmitted. It is this property that makes the C-element unique when joining multiple *acknowledge* signals after a completed transition [28].

3.5.2 Asynchronous Buffer Elements – WCHBs and PCHBs

Weak-Condition Half-Buffer (WCHB)

To provide correct handshaking for each pipeline stage a WCHB is one possible solution, and is used as the preferred handshake and pipeline element in [1].
The implementation considered is designed for use together with the 4-PDR protocol described in section 3.4.2, and the logic elements constructing the WCHB-element are illustrated in figure 3.11. The rail notation for R_{d1} and R_{d0} indicates that both rails are data rails received by the WCHB, where R_{d1} indicates the valid "1"-rail and R_{d0} indicates the valid "0"-rail. T_d-rails behave accordingly, only on the transmitting side of the WCHB. T_e and R_e are transmitted and received *enable* respectively.

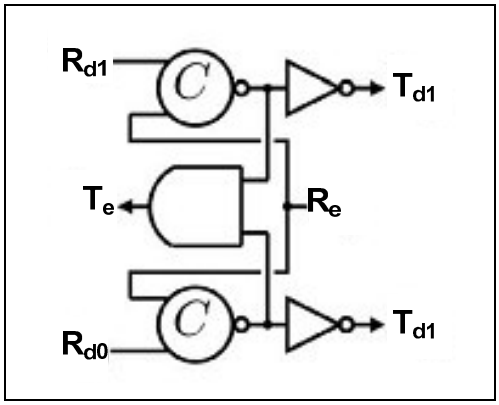


Figure 3.11: WCHB pipeline element detail [1]

The term half-buffer means that a handshake must be completed on the transmitting side of the buffer, before it can begin on the receiver side. This fact can also be seen in figure 3.11 since the output of each C-element is dependent on the signal R_e, which is the received *enable* signal from the instance receiving data transmitted by the WCHB. This is in perfect accord with the transfer protocol described in [1]. The benefit of such a handshake structure is that a token traveling between the buffer elements can never be overwritten by proceeding tokens. A half-buffer implementation means that every consecutive data token travelling in the pipeline is separated by a *spacer*, as explained in section 3.4.2. Applying the Return-To-Zero (RTZ) protocol with 4-phased signaling, one data token is allowed for every second buffer element separated by a {00} spacer [29]. Half-buffers yield small implementation area compared to a full-buffer pipeline stage, where handshakes at the transmitting and receiving end can be overlapped for increased signal speed [1]. The benefit of using a WCHB as a buffer-element is due the few signal transitions to complete the handshake, making it the fastest QDI-buffer because of the short cycle time [16].
WCHB-elements are not optimized for logical computations, and are therefore typically used in the setting of a pipeline buffer stage or a copy-element.

PreCharge Half-Buffer (PCHB)

The PCHB is another pipeline buffer element, but with other functional properties than the WCHB. PCHBs are built on pull-down stacks of transistors used for token computations, and are therefore more optimized than WCHBs to be used in circuits where more complex operations than buffering are required. Typical examples from [1] and [27] use PCHBs in logical computation elements like asynchronous LUTs. In addition to the pull-down computation stack a similar handshake facility as in a WCHB is included, using the same dual-rail interface for all handshakes. This fact makes mixing of WCHBs and PCHBs trivial, allowing for more optimized element construction. Yahya et Al. have some conclusions on this subject in [35], showing potential of increased circuit speed with an optimal mixture of WCHBs and PCHBs in the computational pipeline. Figure 3.12 illustrates the PCHB used in [1].

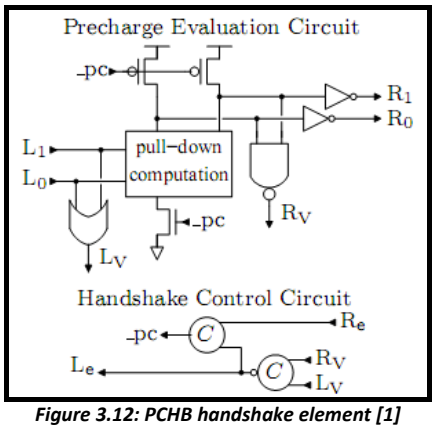
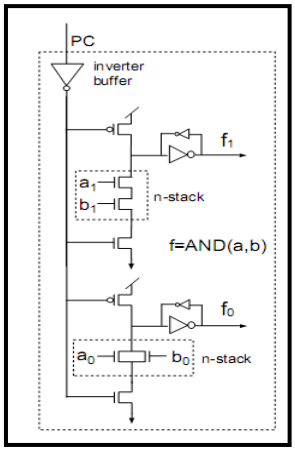


Figure 3.12: PCHB handshake element [1]

One possible problem occurs when constructing larger computation elements in PCHB fashion: the pull-down stack of transistors gets to complex, and the created noise affects other parts of the system. Alternative solutions are presented in literature, both in [1] and [27], dealing with the potential noise problem. The principle of precharge circuits connected to pipelines is explained in [38]. For a simple illustration, a dual-rail AND gate implemented as a PCHB element is shown in figure 3.13.



The PC-signal at a chosen stage N is the precharge signal, and it is received from stage N+1 in the pipeline. When stage N+1 has completed its computation, the completion detection signal is used to precharge stage N via the PC-signal. Precharging ensures that stage N cannot compute data tokens until stage N+1 is finished, implementing the same half-buffer functionality as a WCHB element. The stacking problem is due to the n-stacks which can be depicted in figure 3.13. As the computational block becomes more advanced, the stack size increases rapidly. This problem is partly solved in [27], and examined in chapter 9.2.2.

A more in depth study of different PCHB implementations can be found in [39].

Figure 3.13: Simple AND gate as PCHB implementation [38]

3.5.3 The Copy-element

Section 3.1 described some of the most common models used when dealing with asynchronous design and how a circuit is characterized by its associated delay-model. When it comes to forking of signals, asynchronous designs differ significantly from synchronous designs due to the handshake mechanisms employed in each channel. It is important to consider that for an asynchronous system the splitting of a wire containing a token must maintain handshake mechanisms with all destinations. Another consideration is that to preserve the delay-insensitivity property of a QDI system, the handshake signals must be synchronized in some way [28]. Practically this means that a wire cannot be split in an asynchronous design, as it can when dealing with wires in synchronous systems, without introducing an extra element known as the copy-element.
Figure 3.14 (left) shows a basic sketch of the copy scenario for a two-input copy element, and figure 3.14 (right) gives a more detailed view of a basic copy-element [28]. A copy-element for an Asynchronous Event System will be designed according to figure 3.14 (right) because of its simplicity.

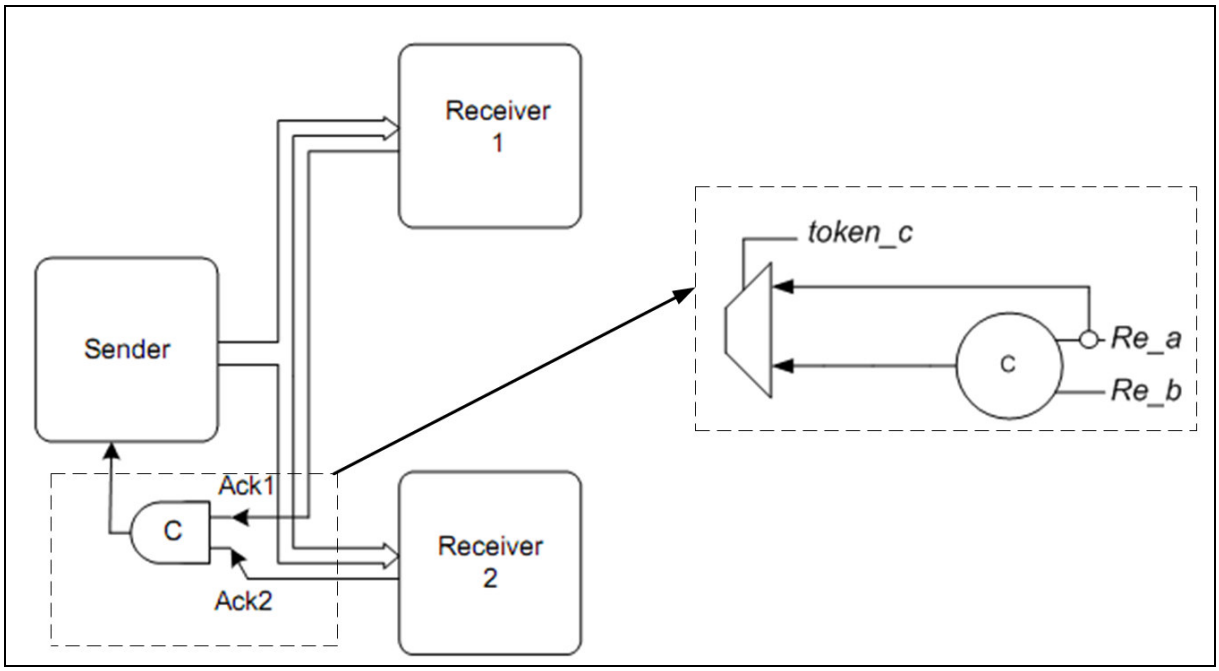


Figure 3.14: Copy scenario for two receivers (left), Copy element detail (right) [28]

A different copy element design is used in [1], where WCHB-elements are used together with a network of C-elements to wait for all received *enable* signals after the forking of a signal. This solution is specially designed to be used in custom logic blocks for asynchronous FPGA structures, with a switch element as destination. This makes the solution rather area consuming, and will not be considered for implementation in an asynchronous Event System.
The implementation in figure 3.14 from [28] uses C-elements combined with MUXes, and a select signal to make it optional if one or several destinations are desired in the forking procedure. This implementation is simple, and suited for use in circuits where the data token only needs to be forked to a limited number of destinations.
Both the implementations considered relay on the QDI circuit model, not constructing any isochronic forks of importance for the circuit delay.

3.6 Asynchronous Pipelines

Many asynchronous interconnect structures, especially those constructed for FPGA implementations [1] [27], rely on pipelining of various granularities to construct high-speed circuits with high throughput. These properties of asynchronous pipelining are interesting for an asynchronous Event System implementation, and an overview is provided, emphasizing terminology and some implementation considerations.

3.6.1 Properties of Asynchronous Pipelines

Asynchronous pipelines are defined in literature as a collection of concurrent hardware processes that communicate through message passing channels [1]. These channels are also known as handshake channels, and the performance of the pipeline is highly dependent on the efficiency offered by the applied handshake protocol [36]. Utilizing this factor has led to the development of pipelines with higher throughput [38], culminating with Ultra-High-Speed pipelines [37]. These types of pipelines anticipates the arrival of certain events to speed up the handshake, but is not considered as an alternative in this thesis. Therefore the focus will be considering pipelines using a 4-PDR protocol with QDI delay assumptions.

Messages sent in the pipeline channels are addressed as tokens. Applying the definition from [36], the tokens containing data are called *evaluation tokens*, while the tokens resetting the handshake are called *reset tokens*. Tokens are passed between pipeline stages in accordance with the applied handshake protocol. Pipeline granularity controls the performance in terms of the speed tokens can propagate through the pipeline. This property is called *slack-elasticity*, and means that changing the pipeline depth to increase speed in any channel will not alter the correctness of the original system. Clocked systems are not slack-elastic since changing local pipeline depths arises the need for retiming of the whole system [1].

Another terminology which is used for asynchronous pipelines is *linear* and *non-linear* pipelines. A linear pipeline is defined as the simplest pipeline structure, containing no split or merge of token channels [36]. Non-linear pipelines on the other hand can contain elements for split and merge. In this thesis only linear-pipelines will be considered.

3.6.2 Sutherland's Micropipeline

Ivan Sutherland introduced the Micropipeline in his Turing Award lecture in 1989 [40], but the descriptions given in this section is derived from [13]. The basic architecture can be depicted in figure 3.15. The control circuit features interconnected C-elements, where the propagation of a signal entering the pipeline replicates that of a FIFO-queue. If the pipeline is waiting for a transmission to commence out of the pipeline, signals already in the pipeline will be held at the stages before the output by the C-elements. Adding registers controlled by the control signals, the micropipeline datapath is constructed. If blocks for logical computation are added between the registers, the pipeline can be utilized for pipelined logical computation. The delay blocks in figure 3.15 are added to specify the delay encountered by the computational blocks. All delay blocks reflect the worst-case delay of the slowest computational block. In this context the micropipeline delivers worst-case performance.

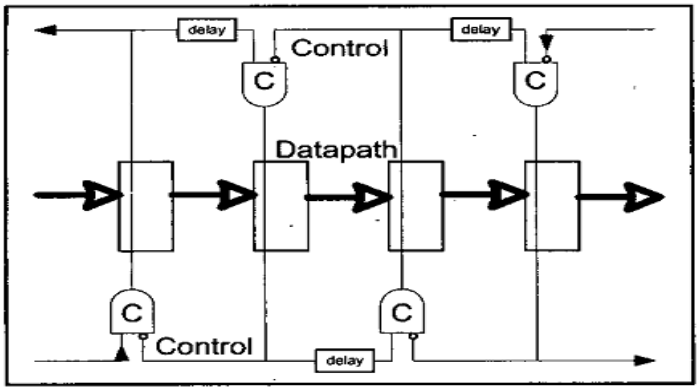


Figure 3.15: Micropipeline architecture [41]

One acknowledged benefit of the micropipeline structure is the elasticity which can be accomplished. Being non-clock dependent, data can arrive in the pipeline at arbitrary times for computation. This is one of the reasons for the popularity of the micropipeline design structure. Problems include the performance and the testing of the pipeline. One stuck-at fault in the pipeline will halt the forward progress of the entire circuit, revealing asynchronous testing as a challenge even in simple circuits. [41].

4 Analysis and Testing of Asynchronous Linear-Pipelines

In addition to the introduction of new circuit elements and a need for new design- and test methodologies combined with specialized synthesis tools, asynchronous circuits also differ from synchronous circuits considering performance and delay analysis. This is still a relatively new area of research, and a proposed model will be used for estimated performance evaluation of an asynchronous Event System solution. The asynchronous solution presented for the Event System uses a linear-pipeline structure, and this chapter will provide the necessary background to understand the challenges and principles behind performance measurements in these structures. A small section considering testing of asynchronous systems will be included to put focus on techniques applicable for testing an asynchronous Event System solution.

4.1 Performance Analysis in Asynchronous Linear-Pipelines

Analyzing the performance of an asynchronous system, for instance a pipeline, is complicated by two factors: There exist no easy way of partitioning the system, and the fact that the system is self-timed [35]. Self-timed behavior complicates performance evaluation by making the performance dependent on relative delay values in the different pipeline stages, values which can be different for each stage if computational elements exist in each stage [35]. The result is that performance analysis based on average delays is not accurate, even though asynchronous systems can compute in average time instead of worst-case time [13].

A method called Token Vector Delay Model (TVDM), developed by Yahya et. Al in [35] [36] , use vectors of latency pairs in each pipeline-stage to represent the delay properties introduced by computational elements in the pipeline. The vector pairs represent delays formed by an arbitrary mathematical function for some chosen statistical distribution.

The proposed asynchronous solution for the Event System use WCHB buffers in the pipeline as described in section 3.5.2. To understand the analysis principles of the TVDM, the token pattern of a WCHB element must be analyzed. Valid tokens in the pipeline consist of an *Evaluation Token* (Eval) containing data or information and a *Reset* (Rest) token, also referred to as a spacer. Using the 4-PDR protocol for sending data the token pattern is as described in section 3.4.2 , meaning that for a WCHB the total delay consist of a Token Vector (TV) pair $[T_{eval},T_{reset}]$ [35]. This token pair represents differences in the computation time for the *eval* and *reset* tokens, along with latencies in the pipeline and other fixed and variable delay factors. All latencies specify TV pairs for each pipeline element, building the fundamentals for analytical modeling when combined with the actual circuit model of the pipeline.

To present the right TV equations, the TVDM model rely on dependency graphs modeling the pipeline behavior. Additional background for such analysis is provided by [36] and [22]. The model used to describe an asynchronous pipeline is depicted in figure 4.1 (left), while the dependency graph used to provide the analytical model is given in figure 4.1 (right) . The pipeline is modeled with a functional block used for pipeline computation, while the register acts as a buffer element for the tokens. The time it takes for a buffer stage to completely process a data pattern consisting of both an *eval* and a *reset* token is denoted Total Cycle Time (TCT). According to [22] determining the TCT is the same as determining the longest simple cycle in the dependency graph, or the cycle with the longest accumulated circuit delay without containing any sub-cycles. Equations for TCT in each pipeline stage can be derived from the dependency graph in figure 4.1 (right). Even though asynchronous processes can trigger concurrently, there is always a main synchronizer event.

For a WCHB element this event comes from each C-element output, and is denoted C↑ and C↓ for an *eval* transition and a *reset* transition respectively. For the next equations C↑ is considered the main synchronizer event. See [36] for more details.

The maximum latency traveling from C↑ to C↓ is given by the equation:
Max [(F↑,C↑,Ack↓, C↓), (Ack↓,C↓,F↓,C↓)] according to the associated dependency graph.
From C↓ and back to C↑ the delay equation is: Max [(F↓,C↓,Ack↑, C↑), (Ack↑,C↑,F↑,C↑)] according to the dependency graph. The total cycle time associated with proceeding and preceding pipeline stages for a WCHB is given by:

$$TCTn_{WCHB} = MAX[\ (T_{F(n+1)\uparrow}+T_{C(n+1)\uparrow}+T_{Ack(n+1)\downarrow}+T_{C(n)\downarrow}),\ (T_{Ack(n)\downarrow}+T_{C(n-1)\downarrow}+T_{F(n)\downarrow}+T_{C(n)\downarrow})] + \quad \text{(Eq. 4.1) [36]}$$
$$MAX[\ (T_{F(n+1)\downarrow}+T_{C(n+1)\downarrow}+T_{Ack(n+1)\uparrow}+T_{C(n)\uparrow}),\ (T_{Ack(n)\uparrow}+T_{C(n-1)\uparrow}+T_{F(n)\uparrow}+T_{C(n)\uparrow})].$$

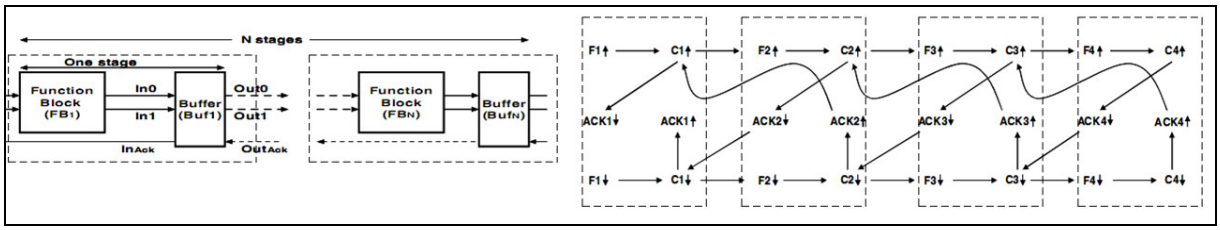


Figure 4.1: Pipeline model (left), WCHB dependency graph (right). [35]

Other approaches to performance analysis use statistical methods to obtain performance parameters for asynchronous circuit structures. All papers considering statistical methods include more process variable parameters in their computations than the TVDM method, like gate variability for adjacent transistors [45], variable threshold voltages and channel lengths [44] and spatial correlation effects in the silicon die [43]. All performance and delay models used in the three mentioned papers rely on computationally heavy Monte Carlo analysis, and require good knowledge of the implemented circuit. Such analyses are therefore not considered an alternative in this thesis, and the simpler dependency graph analysis will be considered for the Asynchronous Event System. A more general analysis using Markov Chains is provided by McGee et Al. in [46].

4.2 Test Generation for asynchronous pipelines

Testing asynchronous pipelines and other asynchronous computational structures is perhaps the most pressing research area for asynchronous logic. As accurate testing of an Asynchronous Event System solution is beyond the scope of this thesis and will be considered an area of future research, only some references towards research on asynchronous testing will be provided.

The work presented by King et Al. in [41] proposes simple testing techniques for Sutherland's micropipeline mentioned in [40], and C-element tree structures. Asynchronous elements are treated as atomic FSMs, so that test patterns can be conducted for each FSM. Using a scan latch in each pipeline stage, scan testing closely related to synchronous testing is managed with high fault coverage and low area overhead.
In [37] methods for stuck-at fault- and handshake testing in Ultra-High-Speed pipelines are proposed.

5 Description of FPGA and CPLD Architectures

The asynchronous elements and design methodologies presented in section 3 bring some challenging design aspects into the microcontroller domain. To fully utilize the different techniques a new routing structure is proposed for the Event System, building on an FPGA inspired topology.
This section will provide background information on both synchronous and asynchronous FPGA designs, and highlight some interesting FPGA features for the Event System. The first subsections will present some common FPGA topologies and notions, including some custom circuit elements specially developed for asynchronous FPGA architectures. To show similarities and differences, synchronous techniques are compared to asynchronous techniques in some of the settings.
One of the proposed solutions for the Event System considers the use of CPLD architectures. A brief overview especially emphasizing CPLD design developed by Atmel is therefore provided, also introducing some general CPLD design considerations.

5.1 FPGA Topologies

Whether synchronous or asynchronous FPGA topologies are under consideration, the basic structure regarding routing resources and connectivity are basically the same. The topologies considered in this section concerns SRAM FPGAs because of its common use and well investigated architectures.

Segmented Routing
This is an Island-style routing topology, meaning that each logical computation block in the FPGA is surrounded by routing resources. Another name also used for such topologies is symmetrical [15]. Routing segments are connected with wires of different lengths, where short wires accommodate local communication and longer wires connects to fewer switch blocks to provide global routing resources with less delay. Local wires connect to more global structures using switch blocks to emulate global connections, giving large programmability for the overall system [34]. Figure 5.1 (left) indicates segmented routing.

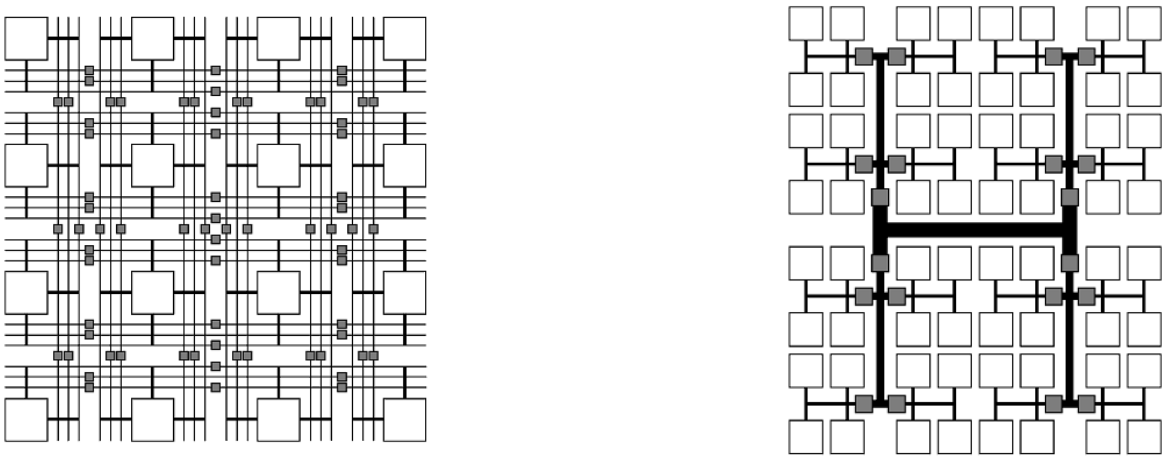


Figure 5.1: Segmented Routing (left) and hierarchical routing (right) [34]

Hierarchical Routing
This routing method considered in [15] [34] routes within a cluster of logical elements at the lowest hierarchical level, with direct connections within this cluster. At the boundaries of each cluster are switch blocks connecting the local segments to the next hierarchical level, making the local clusters accessible to each other. Applying a hierarchical routing topology could be beneficial for circuits where the locality of certain logic groups are dense, achieving better density and performance [14]. The method is however relying on a good placement to utilize the local communication. Figure 5.1

(right) shows the principle of hierarchical routing. Research committed by Aggarwal et. Al [15] indicates that compared to symmetrical FPGAs, hierarchical solutions can give a reduced cost with regards to area because of reduced switch count.

5.2 FPGA Switch Block Design

Much of the area consumed by an FPGA, regardless if it is an asynchronous or a synchronous FPGA, is used for routing resources. According to [34] only 10% of the total FPGA chip area is used for logic cells, the rest for routing resources. This fact makes the switch element one of the most crucial components in an FPGA design [3] [17] [19] and therefore optimizing the design of switch elements are important if high throughput, small area and short delays are to be obtained [18]. The switch is defined as the element connecting horizontal and vertical channels on the routing network, and usually has a uniform number of input/output channels connected to each side. Another important aspect of the switch element design is the routability aspect. Research on this area has according to [3] shown that if three switches are connected to one input/output terminal, a near maximum of routability is achieved. Practically this means that a terminal connects to three other terminals for signal distribution.
Figure 5.2 (left) shows a typical switch block consisting of eight switch points, denoted S, while figure 5.2(right) describes in more detail a design model of a multi-directional switch point where each pass transistor is controlled by an SRAM cell.

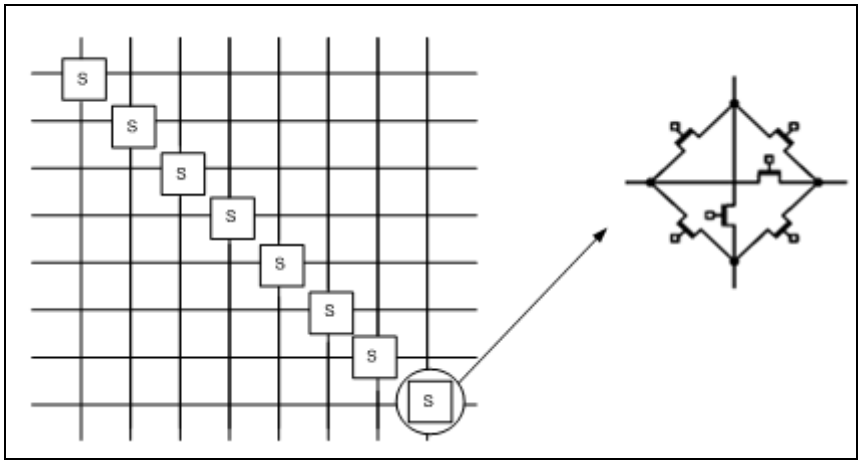


Figure 5.2: Switch block with eight switch points (left), Switch point model (right) [3]

There are two switch types which are mainly considered for different FPGA architectures; subset switch blocks [3] [17] [18] and universal switch blocks [19] [20]. These architectures will be more carefully considered during the subsequent sections.
The principle of construction is the same for synchronous or asynchronous designs considering switch blocks, except that asynchronous switch blocks must contain logic elements to provide handshaking if used in pipelined applications. This is described in [1].
Optimizing switch blocks is a rather complex layout procedure, and therefore this chapter is mostly considered for orientation, meaning that the techniques presented will not be applied to the switch blocks constructed for the Event System. It is however important to be aware of different optimization techniques, and the fact that a simple switch design not necessarily reflects an optimal one. Compared to a either a synchronous or asynchronous FPGA, a routing architecture for the Event System will not contain many switch blocks, hence careful optimization can be considered too resource consuming for a prototype version.

5.2.1 Switch block design for synchronous FPGAs

Since most FPGAs developed today are synchronous, papers addressing switch block design [3] [17] [18] [19] [20] are considering synchronous FPGAs as the fundamental structure. This section will give a brief overview of the most important aspects of effective switch design, and the most commonly used switch types. Only subset switchblocks will be considered for Event System implementation for presented solutions relying on FPGA inspired routing topologies.

Subset switch blocks

The papers [3] and [17] by Schmit et Al. investigate effective layout of subset switch blocks, and some heuristics are proposed in order to compute the placements of switch points in order to create the best possible layout for a given number of input/output terminals. The switch block illustrated in figure 5.2 represents one of the least efficient layouts, because all switch points are stationed on a diagonal with reference to each other. Usually this layout presents an unoptimized Euclidean distance between two switch points; hence optimization heuristics can be applied. The optimization technique presented is based on permutation matrices satisfying certain equations using the modulus operator. A superficial example for 8 switch points is illustrated by the optimal permutation matrix in figure 5.3 (left) directly translated to an initial placement with optimal Euclidian distance between switch points figure 5.3 (centre). figure 5.3 (right) is the final and optimized switch block layout for eight switch points, showing significant differences from the diagonal layout in figure 5.2. Note that the position of each “1” in the permutation matrix indicates an initial switch point placement on the grid.

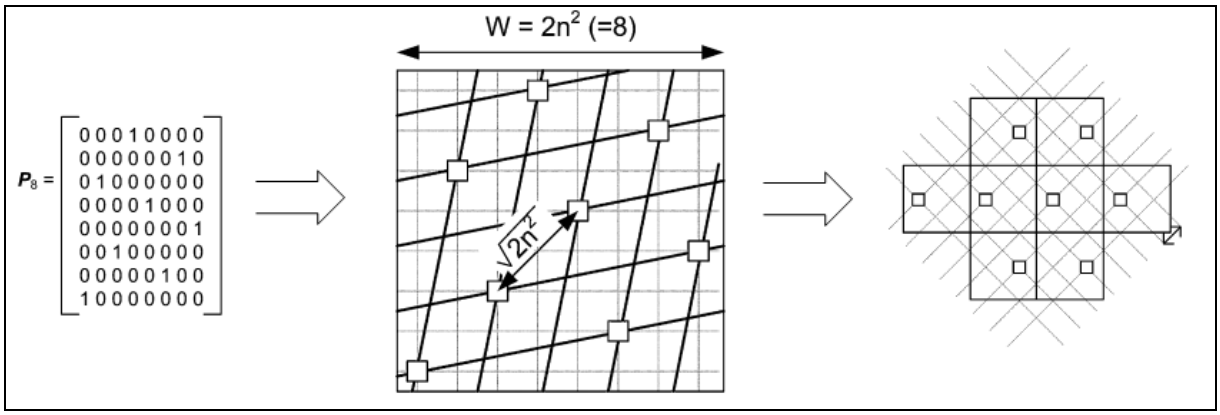


Figure 5.3: Optimal permutation matrix (left), initial placement (center), optimal switch block (right) [3]

Universal Switch Blocks

This switch block class represents switch blocks with inherited isomorphic properties and symmetric wiring topology [20]. The isomorphism property refers to identical routing capacity for two isomorphic switch blocks, meaning that if the terminals of a switch block M can be relabled to the terminals of another switch block M’, they can be considered isomorphic. An example presented in [20] uses a symmetric switch module as basis for constructing isomorphic alternatives of this block. Figure 5.4 presents this example. All these switch blocks will be universal by the presented definitions.

Experimental results from [19] and [20] show considerable savings in area compared to for instance subset switch blocks, when 100% routability is demanded.

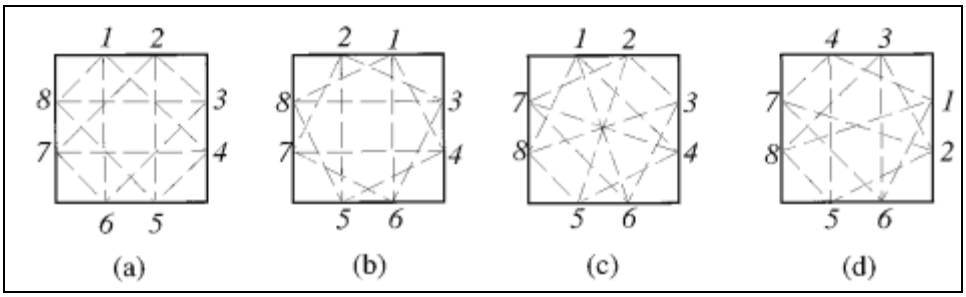


Figure 5.4: Symmetric switch block (a), isomorphic blocks of a (b-d). [20]

5.2.2 Switch Block Design for Asynchronous FPGAs

Most of the developed commercial FPGA designs consider synchronous FPGAs, and some designs like MONTAGE [4] [21] implements asynchronous architectures by porting clocked FPGA architectures to an asynchronous circuit implementation. The works presented in [1] [32] investigates pipelined FPGAs built from pipelined asynchronous circuits, not considering a clocked environment.

The main difference in switch block design compared to synchronous circuits is that wires for handshake must be provided by each switch point in the switch block. An experiment conducted in [1] states that pipelined switch blocks deliver superior performance when the routing distance is long, mainly because of the series resistance encountered by the signal when passing many pass-transistors in an unpipelined environment. The proposed switch block design is therefore constructed of two WCHBs for each switch point, providing a fine-grained pipeline structure with high throughput. Figure 5.5 shows a fine grained pipelined switch point.

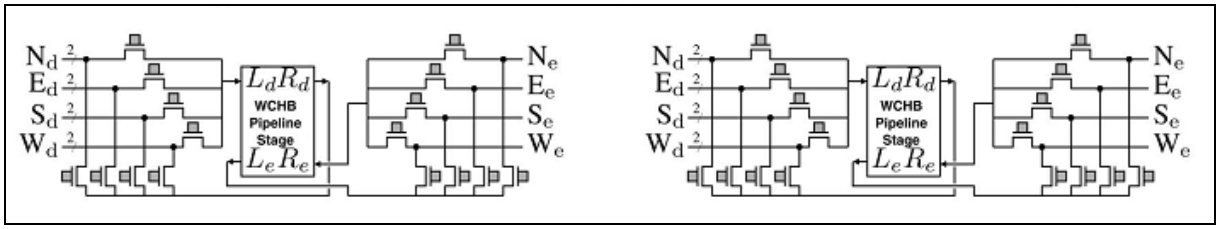


Figure 5.5: Fine grained pipelined switch point, with two WCHBs for each switch point [1]

A switch design inspired by this layout will be developed for an asynchronous Event System solution.

5.3 Asynchronous LUT and Logic Block Design

In most conventional FPGAs the Look-Up Table (LUT) is the basic block when it comes to implementing programmable logic functionality. According to [34] the LUT has proved to be the best alternative for supplying programmable logic, and it is programmed by the configuration bitstream provided by the synthesis tool. This chapter will only consider SRAM-programmable LUTs for asynchronous architectures, which are applied in the Asynchronous Event System. The benefits of using SRAM technology in this context is large programmability, with the drawback of larger area when implementing the memory cells, and the need for some RAM-device to hold the configuration bits [14]. The LUT is often part of a larger structure known as logic block, where output selection, carry logic and bypass functions are included. For orientation a good overview of different synchronous logic block implementations spanning a variety of vendors is given in [14].

In order to utilize the potential of asynchronous circuits, custom LUT designs have been presented in different papers focusing on how the LUT can be interfaced to proper handshake logic. The design of the entire logic block includes the handshake logic needed to operate in asynchronous environments.

A solution from [1] uses highly pipelined logic blocks, and the LUT is implemented using a PCHB computation- and handshake environment. A sketch of the logic block can be seen in figure 5.6.

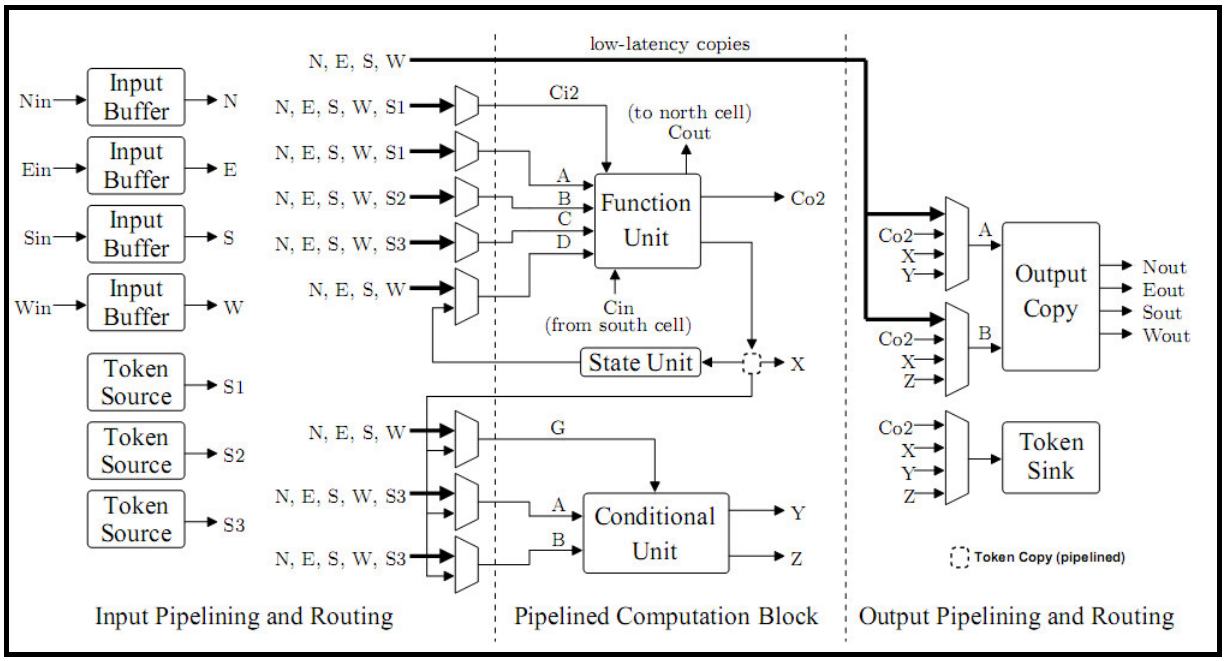


Figure 5.6: Asynchronous Logic Block [1]

The differences from synchronous implementations feature the output copy element, conditional unit and token source / sink blocks. Output copy elements are needed to ensure correct handshake with all elements receiving the logic block output. The conditional unit offers split and merge functionality for tokens, while the source and sink produce and consume tokens to generate tokens at reset and consume undesignated tokens. All facilities within the logic block are different from synchronous implementations, making this a custom design process. The functional unit block in figure 5.6 contain the LUT and carry logic similar to that found in synchronous implementations. The difference is that each functional unit is implemented with either a PCHB or WCHB handshake environment dependent on the level of logic computation needed. The logic block from [1] can be seen in figure 5.7.

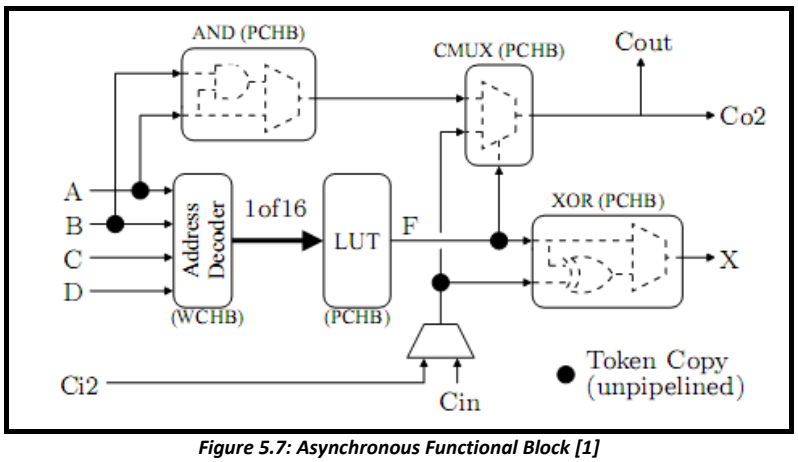


Figure 5.7: Asynchronous Functional Block [1]

Traditional PCHB implementations can suffer from noise problems because of a rather complex pull-down stack of transistors, and also size overhead for the associated implementations [1] [27].

A solution to this problem is presented by Mahram et Al. in [27] where the complicated transistor stack used for dual-rail computation is simplified by making the computation inside the functional block in a single rail environment. Handshake signals are still considered dual-rail. Figure 5.8 illustrates the building blocks of the computation block, with handshake elements.

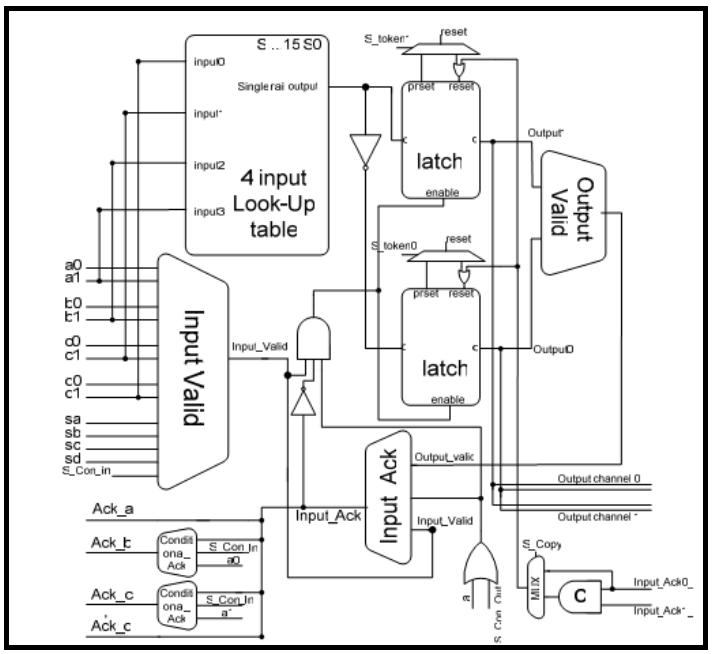


Figure 5.8: Modified PCHB computation circuit [27]

A more in depth analysis of this block is presented in chapter 9.3.1, where a modified version is used for logical computation in the Asynchronous Event System solution. Another point to emphasize is that the PCHB handshake environment can be used in many logical computation elements and not only with 4-LUTs as in the example.

5.4 CPLD Architectures

As an alternative to logical computation using FPGAs, Complex Programmable Logic Devices (CPLDs) are another good option. CPLD architectures are known for their ability to outperform FPGAs when it comes to speed [5] and for its capability to implement glue-logic [42]. According to [42] circuits which can exploit AND/OR gates and not consume large numbers of flip-flops are good candidates for CPLD implementation. Finite State Machines (FSMs) are examples of such circuits.
Building on the basic logic blocks from SPLDs (Simple PLD), CPLDs offer a high level of predictability when implementing logic. Partitioned into several SPLD-like blocks the speed-performance of the total implementation becomes more predictable.

SPLDs form the basic building blocks for CPLD architectures. The SPLD architecture can be implemented as either PLA (Programmable Logic Array) or PAL (Programmable Array Logic). Both methods use planes of AND- and OR-logic, whereas the PAL structures only allow programmability of the AND-plane, PAL structures allow programmability for both the AND- and OR-planes. The structures considered in this chapter only feature FLASH programmable solutions.

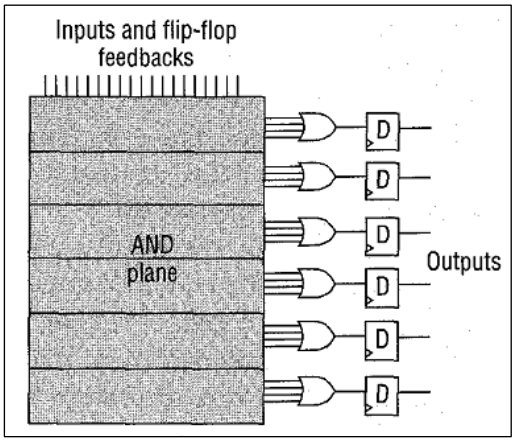


Figure 5.9 shows a basic PAL programmable architecture. To show how a CPLD is constructed, the Altera Max 7000 explained in [42] will be used as an example. The basic architecture of the Max 7000 series feature a Programmable Interconnect Array (PIA), which can route any of the Logic Array Blocks (LABs) to a selected I/O-pin or other LABs. The basic architecture can be depicted in figure 5.10 (1), showing the top level connection of LABs. Each LAB can be viewed as a complex SPLD structure, consisting of interconnected macrocells as shown in figure 5.10 (2).

Figure 5.9: PAL architecture [42]

The macrocells in figure 5.10 (3) can connect to all wires distributed on the PIA, and consists of programmable product terms constructed by an AND-plane. A product term select matrix feeds the desired term into an OR gate connected to flip-flops. Selecting the appropriate product term, any logical expression featuring the selected input wires can be realized.

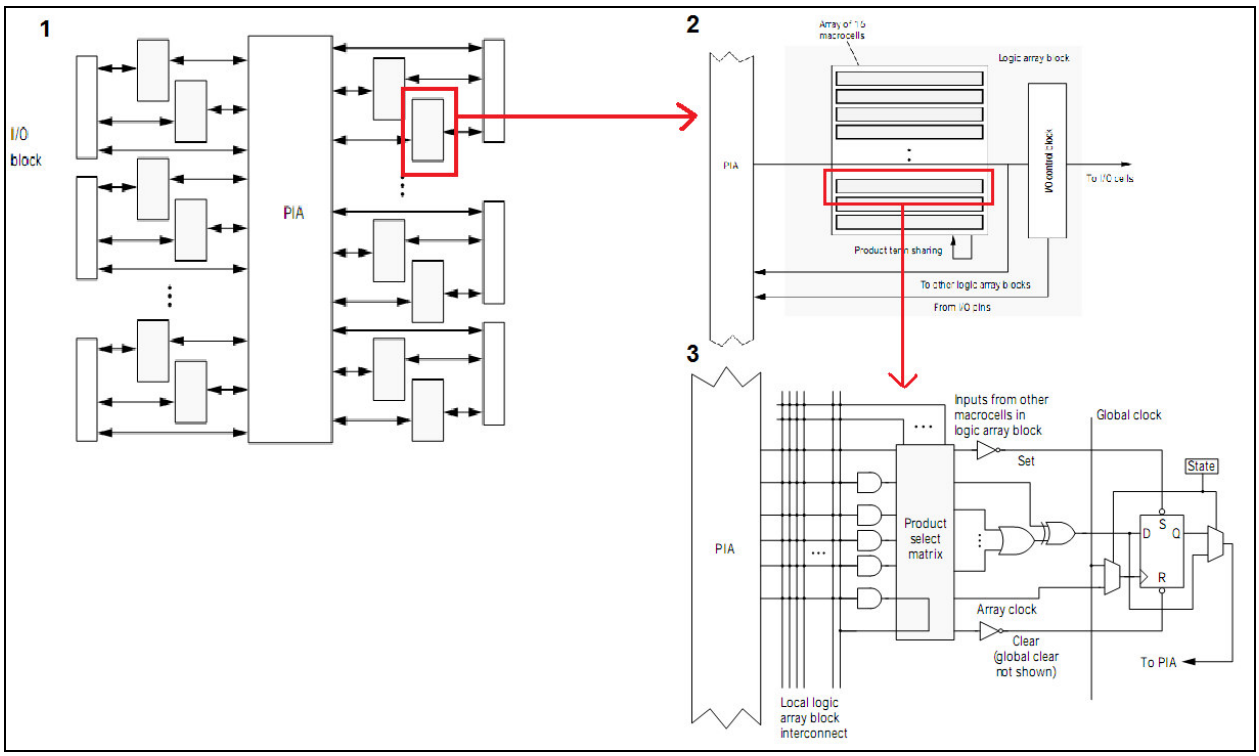


Figure 5.10: Altera Flex 7000 architecture (assembled from [42])

5.4.1 Atmel CPLD Architecture

Applying some of the benefits mentioned in the previous section to a CPLD based Event System architecture could enhance the flexibility and computational power of the system. Atmel delivers several high performance CPLDs, and reusing elements of these architectures in an Event System context can produce a powerful implementation alternative for the Event System, also providing great time-to-market potential. The architecture under consideration is the ATF1508RE CPLD from Atmel [7]. Providing large routing flexibility, small pin-to-pin propagation delay and up to 40 product terms handled for each macrocell, the architecture delivers state of the art CPLD performance. Another useful feature is direct interconnecting between Logic Block neighbors.

Architectural details can be depicted in figure 5.11, concerning both the top level architecture and macrocell details. The Bus wires feeding into the AND-structure are selected from the Global Bus, and 40 wires connect to each macrocell using a switch matrix. The Global Bus includes foldback signals from all Logic Cells as well as all I/O signals. Having the possibility for cascaded coupling, up to 40 logical terms can be computed by connecting all 8 Logic Blocks in cascade.
Compared to figure 5.10 there are some similarities in the fundamental architectural buildup of the macrocells. The AND-plane uses the same connectivity in both implementations, feeding into an OR-gate and flip-flops. Although slightly differently implemented, possibilities for interconnecting macrocells and distributing foldback signals are also present in both architectures.

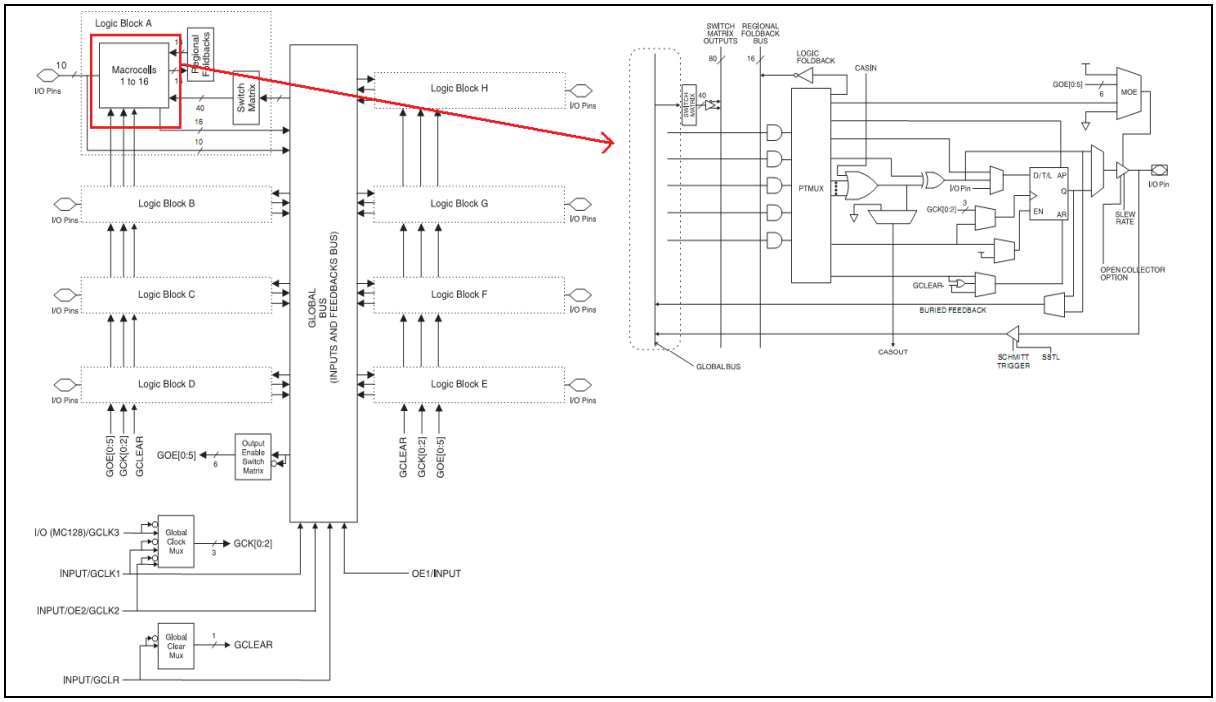


Figure 5.11: Atmel ATF1508RE architecture and macrocell detail [7]

6 Asynchronous VS Synchronous Event System Design

The previous sections have provided the background for considering different solutions for the Event System, and also discussed some of the differences between synchronous and asynchronous implementations. This section considers the two design paradigms as alternatives for the Event System, revealing the gains and losses of applying the different design methods in the AVR® XMEGA. Factors which will be evaluated at a superficial level in this chapter, but considered in depth in chapter 11 and 12, include system costs, circuit complexity, performance and size, tools support and functionality.

6.1 Synchronous Design Considerations

6.1.1 Drawbacks with a Synchronous Event System Design

Most microcontroller systems rely on a synchronous design flow, omitting the problems for EDA- and CAD tools support often experience by asynchronous systems. However, as the processes used in current circuit technology become increasingly aggressive passing 65nm and beyond, the uncertainties in the production makes it difficult to calculate clock frequency with sufficient precision. To compensate for this inaccuracy more variable factors must be taken into consideration when calculating the clock frequency, slowing down the clock speed significantly. An example from [29] shown in figure 6.1 illustrates this problem.

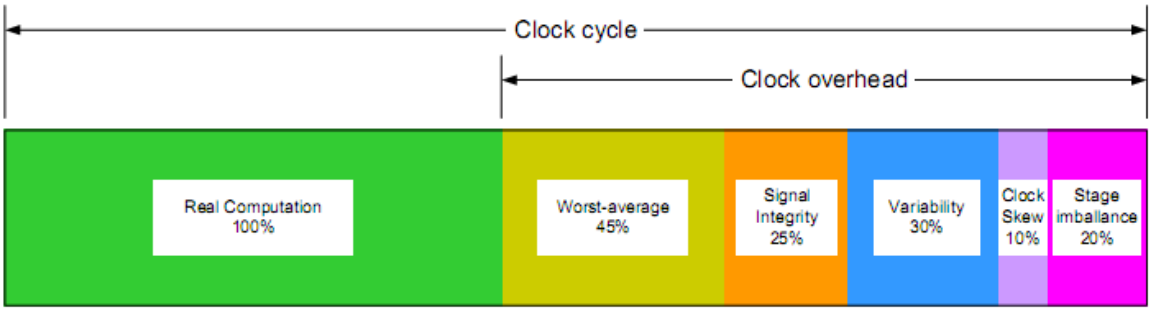


Figure 6.1: Parameters for determining a safe clock cycle duration [29]

As can be seen, variable factors constitute about 120% of the total clock cycle compared to the actual computation time. To compensate for the variable factors encountered in more aggressive processes, more accurate analytical timing models combined with statistical models is a possible solution. Papers considering variable factors in the manufacturing process includes [43] [44] [45], all emphasizing different aspects of predictability difficulties. Although this illustrates a general scenario, the Event System will be part of this development as well and suffer from the same unfortunate constraints.

Another design consideration related to the clock is distribution of the clock tree to meet timing requirements, setup- and hold timing constraints and minimize the clock skew. Normally these challenges are met by the design tools employed by each manufacturer, but as the process size decreases and new processes are introduced the tools must be updated to cope with new timing requirements. This complicates the process of fully utilizing the new technology instantly.
Possibly penalizing the achievable circuit speed, a weakness in robustness and adaptability considering synchronous circuits is reveled at the same time.
Synchronous systems are not robust against delay variations, and are therefore considered Delay-Sensitive or bounded-delay systems [13].

DS systems do not have the same adaptability as asynchronous systems towards changes in external conditions, and must therefore always expect worst case behavior. Handling events which are by nature asynchronous, increased performance could be achieved by utilizing these properties.

When considering low power, synchronous designs have a disadvantage compared to its asynchronous counterpart. All components connected to clock lines will toggle their transistors for each cycle, even in parts of the circuit not currently used in a computation. Although the Event System operates with minimal use of CPU resources, it is subject to unwanted energy consumption due to clock toggling even in idle operation.

6.1.2 Benefits of a synchronous Event System design

Even with some commonly accepted drawbacks, the benefits of designing a system in synchronous fashion could balance the scales. Most manufacturers have a well developed tool flow ranging from complicated design tools to synthesis- and layout tools, utilizing the available process technology as much as possible. Synchronous circuits are efficiently designed using well defined methodologies approved by all designers. Standard cell libraries developed for each process is available, yielding smaller designs and limits the need for extensive custom design. A direct consequence is a reduced time-to-market because of fast design development.

Most of the commercially available microcontroller- and peripheral systems are synchronous, which makes interfacing easy without having to deal with the metastability issue present when interfacing synchronous and asynchronous systems. In the Event System context this would mean that interfacing a new design into the existing AVR® XMEGA design would be much more trivial. Well developed test environments ensure verification and full compatibility with scan testing.

6.2 Asynchronous Design Considerations

6.2.1 Why Consider an Asynchronous Implementation?

Section 3.3 gave an overall review of possible benefits and drawbacks of asynchronous design. In the context of the Event System it is clear that some benefit could be gained by an asynchronous implementation.

First of all the tendency of decrease in process size continues, yielding more aggressive processes. With no need for considering the clock limitations depicted in figure 6.1, utilizing a new process would be much easier in the asynchronous domain. Increased size to due to handshake elements would also be masked because of smaller gate size, giving synchronous designs less advantage in this area as well. Increased signal speed means increased processing, and with the additional functionality developed for the Event System during this thesis the computational power as well as the distribution power could be increased with an asynchronous implementation. Considering highly pipelined solutions, the gap in signal speed compared to a synchronous version could increase even more. According to scaling theory a size decrease of 50% compared to the previous technology is possible for digital logic [49].

Events are defined as asynchronous happenings at some instant in time, and could be distributed naturally with an asynchronous routing network. Using such a facility would inherit all the robustness of QDI circuits, adapting to changing circuit conditions with ease. The most interesting asynchronous trait for the Event System is natural power down capabilities, which should decrease the associated power consumption compared to the current Event System. An exact decrease factor is hard to derive without a prototype or explicit implementation knowledge, but according to papers, among them [1] and [13], a decrease should be expected.

6.2.2 Drawbacks of an Asynchronous Implementation

The lacking support of a good design flow and tools optimized for constructing asynchronous designs are one of the chief reasons for postponing the thought of an asynchronous Event System. In a marked where rapid development is necessary the asynchronous design flow would slow down design efficiency, resulting in longer time-to-market than a synchronous alternative. Developing a custom set of design tools and standard cell libraries for asynchronous logic would make the design cost to high compared to the immediate gain. To support new features the tools produced by software designers must also be altered, adding another cost factor to the total tool cost. A related design cost include re-training of engineers for an asynchronous designflow.

It is a common conception that testing and verification makes up about 70% of the total design time. Considering the challenges associated with asynchronous testing, the time spent on testing and verification would be further increased.

Section II –

Proposed Event System Designs

7 Hierarchical Event Routing Network with I/O Processor

The first design attempt for a new Event System implementation emphasizes changing the routing structure fundamentally, introducing an FPGA-inspired routing topology. To control the execution of events an Event I/O processor is considered, connecting to all peripheral instances.
In the following sections the design will be presented, along with an overviews of costs, size and its feasibility as a solution for the AVR® XMEGA.

7.1 Features

- Programmable routing featuring FPGA switch blocks
- Pseudo-hierarchical routing topology to exploit local communication
- Programmable I/O-processor for flexibility and processing power
- Logical operations on events through the I/O- processor

7.2 System Architecture

The Hierarchical Event Routing Network (HERN) architecture consist of a three level hierarchical routing principle inspired by hierarchical FPGA routing as presented in section 5.1. Figure 7.1 illustrates the partitioning of hierarchical levels, where level 0 consists of one Port/TC and its associated peripherals, level 1 of two neighboring Port/TC blocks connected by routing or an ADC, DAC, DMA peripheral and level 2 is level 1 blocks connected via the I/O processor.

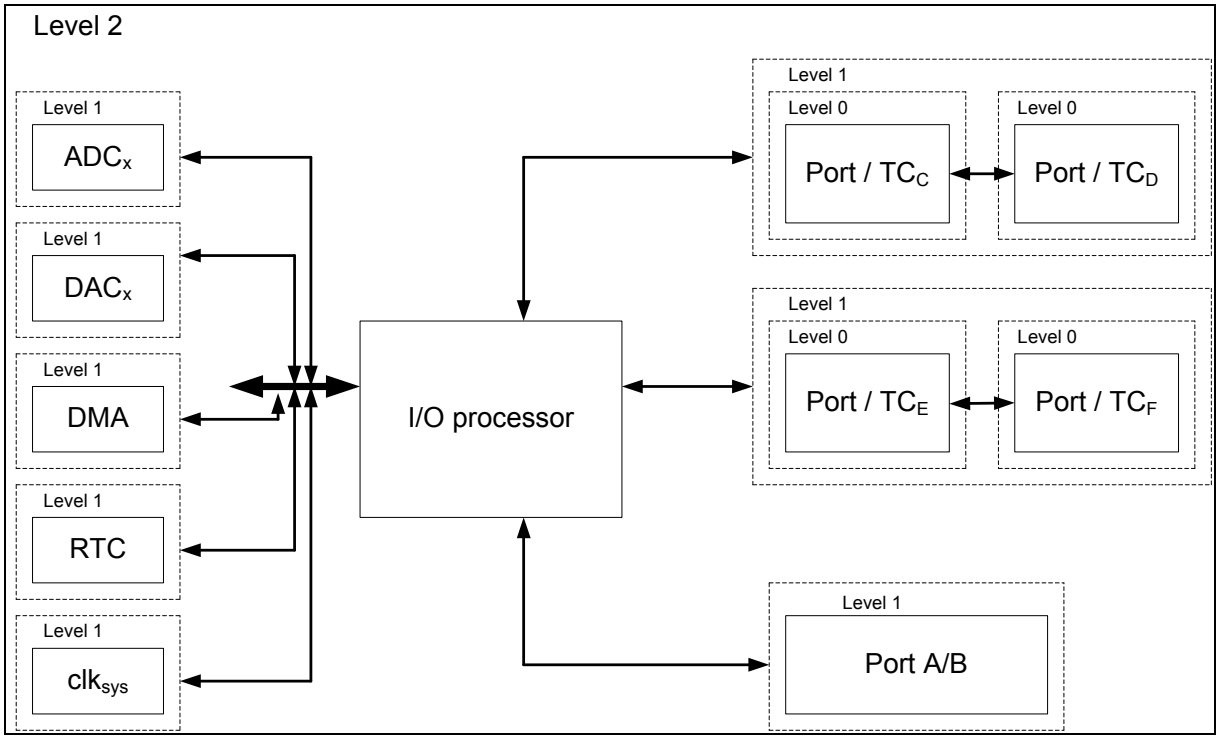


Figure 7.1: Hierarchical level partitioning for the HERN

Communication on level 0 is denoted local, communication on level 1 is denoted as regional and communication on level 2 is considered global. To ensure routing flexibility, programmable switches make up the interface between different hierarchical levels. This principle is more explained in detail in figure 7.2. Port A/B is not included in figure 7.2, but would be coupled directly to the I/O-processor. The global wires connect to all of the global peripherals like in the original Event System, and the wires are directly interconnected to the I/O-processor.

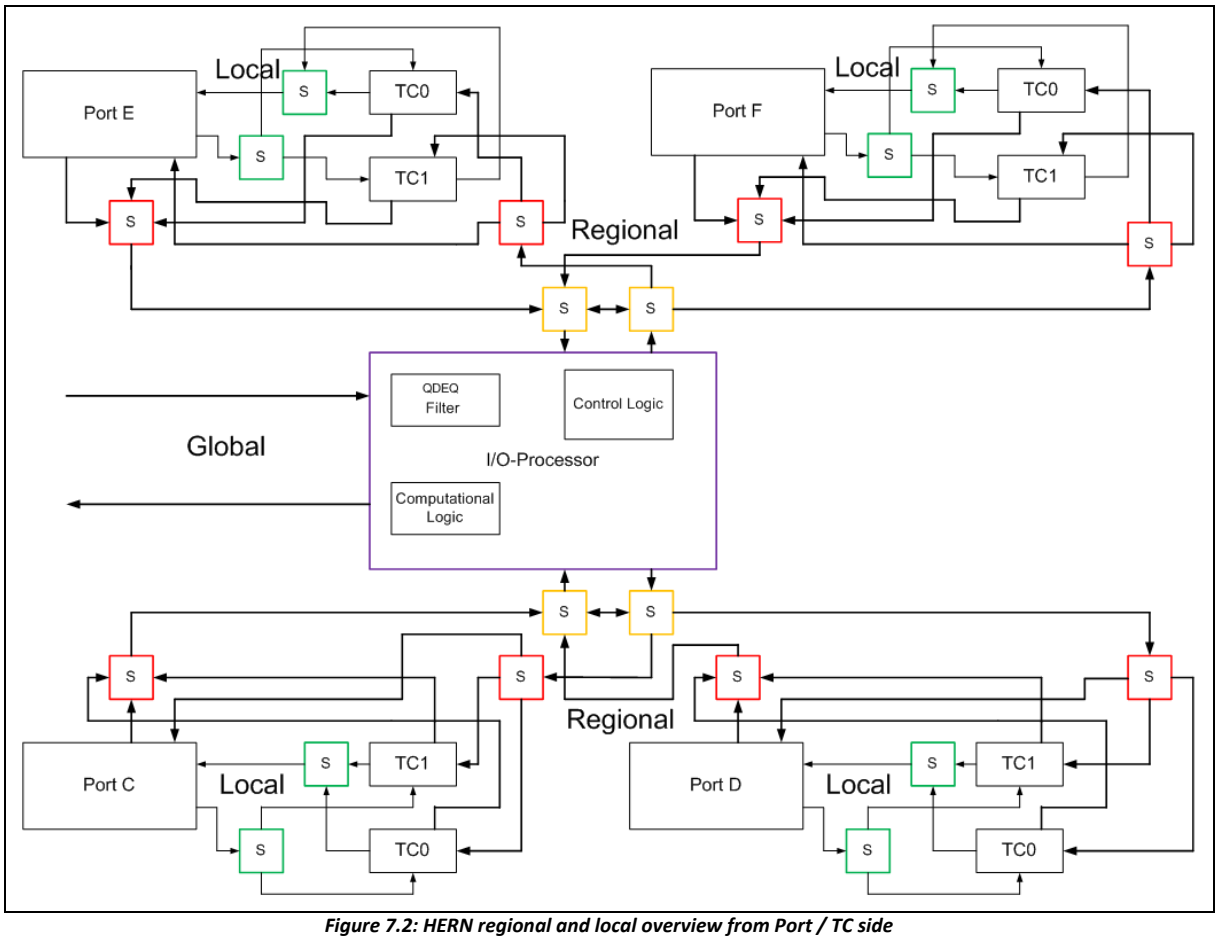


Figure 7.2: HERN regional and local overview from Port / TC side

With an I/O-processor in the center containing Event System functional blocks like QDECs [6] and filters along with added computational logic for event processing, the HERNs regional and local hierarchical levels provide good routing capacity. Main routing directions are split in two, using bi-directional switches instead of the multidirectional switches usually applied in FPGAs. This is for simpler switch design, and easier control and determination of the event flow. Each switch is custom made for its purpose and hierarchical level. Local switches have a capacity for 4 event channels in each direction, while regional switches have a capacity for 8 event channels. At a global scope between the I/O-processor and ADC / DAC peripherals 8 event channels are used in each direction, providing double the global capacity of the original Event System. All event channels use the event notation from section 2, and will consist of the same *ev_c* and *ev_d* wires.

The idea of applying the hierarchical routing structure to the Event System has its background in the system concept called Multifunctional Timer I/O unit (MTIO) [8], preceding the Event System. As the name MTIO indicates, I/O-pins and TC modules communicate locally to enhance peripheral functionality. Applying the same principle of communication between Ports and TC-modules gives more resources for local signal routing, making the HERN suitable for extensive handling of I/O-signals and events at the local- and regional communication level. The switch block colors indicate different block types, and equivalent NAND gate count for each switch block can be viewed in appendix A 16.2-1 - A 16.2-4. The appendixes focus on the asynchronous switch block which will be introduced in section 9, but a synchronous alternate gate count is also given because both the synchronous and asynchronous switch block designs rely on the same template.
Figure 7.2 with an associated gate count is provided by appendix A 16.1-1.

7.2.1 Architectural Considerations

Switch Blocks based on SRAM technology

When constructing the basic routing facility of the HERN, the decision of choosing programmable switch blocks instead of the conventional MUX solutions for routing was the first major design decision to be made. The switch block solution was chosen to be consistent with the pseudo-hierarchical FPGA topology applied, and to import the concept of programmability to the routing network in addition to the I/O-processor. Another trait is that SRAM-cells are size effective if implemented from the Atmel standard-cell library [10], enabling design of area efficient switch blocks. According to [11] an SRAM based solution gives increased flexibility over a MUX-based solution, although the solution is slightly more area consuming. The decision to use SRAM-based switch blocks required a lot of custom designed switch elements for different numbers of input/output event channels, depending on the hierarchical level of operation for the switch block. The connection block mentioned in [11] used to connect input/output pins to the routing channel in standard FPGAs are considered a peripheral dependent implementation, and is therefore not mention as a part of the HERN routing architecture. Keep in mind that the HERN is FPGA inspired, and not a general FPGA topology.

Interconnections

The decision to be made at this level is considering hardwired direct connections in some portions of the system, at the cost of a reduced number of programmable connections. Although [11] states that careful placement of hardwired interconnects can increase performance, the HERN is constituted only of programmable interconnects. The reason is a probable increase in area and less flexibility considering the overall routing [11]. Wanting to maximize programmability and flexibility with the HERN solution, a decision involving only programmable interconnections was applied.

7.3 The I/O Processor – Computational Power and Flexibility

The computational- and processing power of the HERN is centered in a custom designed I/O processor. Some of the benefits gained by such a centralized processing solution include programmability by applying a unique instruction set for event distribution, centralized processing logic yields smaller area for these elements compared to if implemented in each peripheral and logical operations on events can be delivered by the I/O-processor. Implementing state-machines for operation on event-chains is also possible with some associated memory, adding a powerful feature for processing events in a longer sequence without interference from the CPU. Implementing FSM functionality would continue the FSM heritage suggested by the MTIO system [8].

However this amount of flexibility and computational power comes with a high cost. Even a simple I/O-processor inflicts a significant increase in circuit size and power consumption. Evaluating the fact that the AVR® XMEGA is an 8/16 bit microcontroller, this will not be cost effective compared to the performance offered. This fact is strengthened by previous research performed by Atmel, drawing the conclusion that using existing process technology, such an implementation is not cost effective [9].

An obvious reason for considering a processor based alternative for the Event System is chiefly motivated by the available degree of programmability. Given the proper instruction set, the processor could manage advanced computations on transmitted events allowing for advanced peripheral operations. Of course the associated power consumption would increase with a dedicated processor, especially regarding dynamic consumption, and the consumed amount would further increase if more advanced functionality was added to the I/O-processor.

7.3.1 Area Estimations and Programmable Bit Count

This section will provide some estimation on expected HERN area in terms of NAND equivalents, chiefly based on appendixes A 16.2-1 - A 16.2-4 for switch block area, and appendix A 16.1-1 for total area. Considering estimated size for an I/O-processor, previous research by Atmel [9] puts a basic I/O-processor at 2200 NAND gates including routing and shared memory with the main CPU. An advanced I/O-processor cost approximately 6000 NAND gates, also with shared memory. These numbers are released with the courtesy of the Atmel Corporation. Have in mind that the estimated area does not consider routing area in terms of wires and additional memory to provide programming bits to each switch block. These factors will be included in section 12 where all proposed Event System solutions are compared. **The NAND gate count is for logic area only**. Table 7-1 provides a gate count overview for different HERN alternatives.

Domain	Switch type $\#CH_{In}/\#CH_{Out}$	# SRAM transistors switch block	**Gate count per element [NAND]**	Number of switch blocks	**Total gate count [NAND]**
Local	8x2/8x1	64	7.5	8	60
Regional	8x3/8x1	192	22	8	176
	8x2/8x2	256	29	4	116
Total					**352**
Simple I/O			**2200**		**2552**
Medium I/O			**3500**		**3852**
Advanced I/O			**6000**		**6352**

Table 7-1: Measured NAND equivalents for HERN routing and I/O-processor

Because of increased programmability for an applied HERN solution, an amount of bits must be provided by memory to configure the routing network. Included in the same appendixes as gave the area estimations are also the bits required to program each switch block. Table 7-2 summarizes the results.

Domain	Switch type $\#CH_{In}/\#CH_{Out}$	Required programming bits	Number of switch blocks	**Total bit amount**
Local	8x2/8x1	4	8	32
Regional	8x3/8x1	24	8	192
	8x2/8x2	32	4	128
Total				**416**

Table 7-2: HERN required programming bits

A note to table 7-2 is that extra registers for the I/O-processor is not considered, only bits connected to routing.

7.4 Conclusions regarding the HERN

Because of the size and resource issues regarding the I/O-processor, a full solution for the HERN was not developed or considered at a more detailed level than in figure 7.2. The total implementation was regarded too resource consuming with the available process technologies based on previously mentioned Atmel research. Attempting this kind of design for the Event System reveals some potential, and also the need for further research in the area of using a dedicated processor to control microcontroller sub-systems. Such a research is considered beyond the scope of this thesis. A note to the area and power issue is that with subtle designing, the solution could be a candidate for the AVR®32 Event System which represents a more advanced microcontroller structure.

Despite the resource issue, some of the proposed routing architectures and solutions were considered worthy of further development. The hierarchical topology is applied to the Event System design proposal in section 9, but in a slightly refined version. Ideas considering programmable switches using SRAM technology are also further developed, and presented in section 9.2.5 along with appendix 16.2. Although not the first choice for an XMEGA Event System implementation, the HERN can be considered a good first design attempt for a routing infrastructure.

8 A CPLD and Bus Based Solution for Event System Routing

This section will present the architecture and design of a CPLD based I/O - and event-processing solution. To form a new solution for the Event System the hierarchical routing legacy of the HERN was further developed, but replacing the I/O-processor with blocks of CPLD logic in each Port / TC module. A discussion on benefits of using CPLD architectures as an alternative to classic FPGA architectures are included, along with a description on how the solution can be integrated with the existing peripherals. Finally some conclusions on the proposed design are presented.

8.1 Features

- Bus based routing for fast and deterministic event transfer
- Utilization of local connectivity between peripherals through cascaded CPLD Macrocell Blocks.
- Integrated and connected CPLD-blocks in each Port / TC – peripheral for logical processing of incoming events.
- Using Atmel`s original CPLD design allows for simpler integration with existing peripherals
- Extensive handling of I/O signals through the CIOBus

8.2 System architecture

The routing structure used for the CERN (CPLD Event Routing Network) adopts the hierarchical description used for the HERN in section 7.2. However, the programmable FPGA routing solution is substituted by a bus solution similar to the one used to connect Logic Blocks in [7].

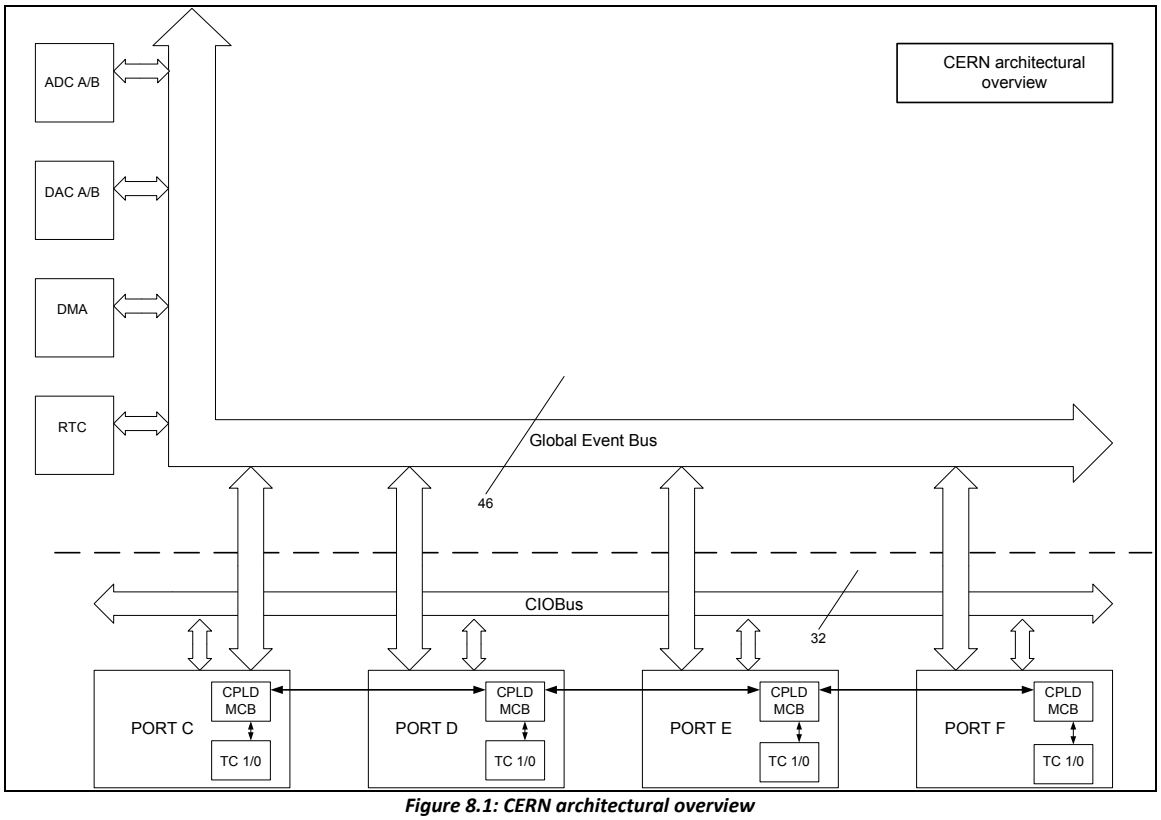


Figure 8.1: CERN architectural overview

Figure 8.1 shows the connectivity between different peripherals using a Global Event Bus (GEB). The GEB is considered the primary routing facility for events between all peripherals. To enhance

performance between I/O Ports and TC modules, the CIO (CPLD I/O) Bus is introduced as an extended version of the currently implemented I/O bus [6].

The CIOBus consists of 32 event channels for specific I/O usage, and is also used to connect the CPLD MCBs (CPLD Macrocell Blocks) forming a CPLD network for logical computation between the Port / TC modules. A more detailed connection scheme can be seen in figure 8.2, where a regional CERN view is provided. The regional level of the hierarchy features Port / TC modules connected to the CIOBus, and connections from different CPLD MCBs. Local level connectivity features connections between a Port and its associated TC modules and an included CPLD MCB. Also considered a local connection is the cascade link provided between neighboring CPLD MCBs. This connection is included to secure a fast transfer of progressed events at a local level, without needing access to the CIOBus.

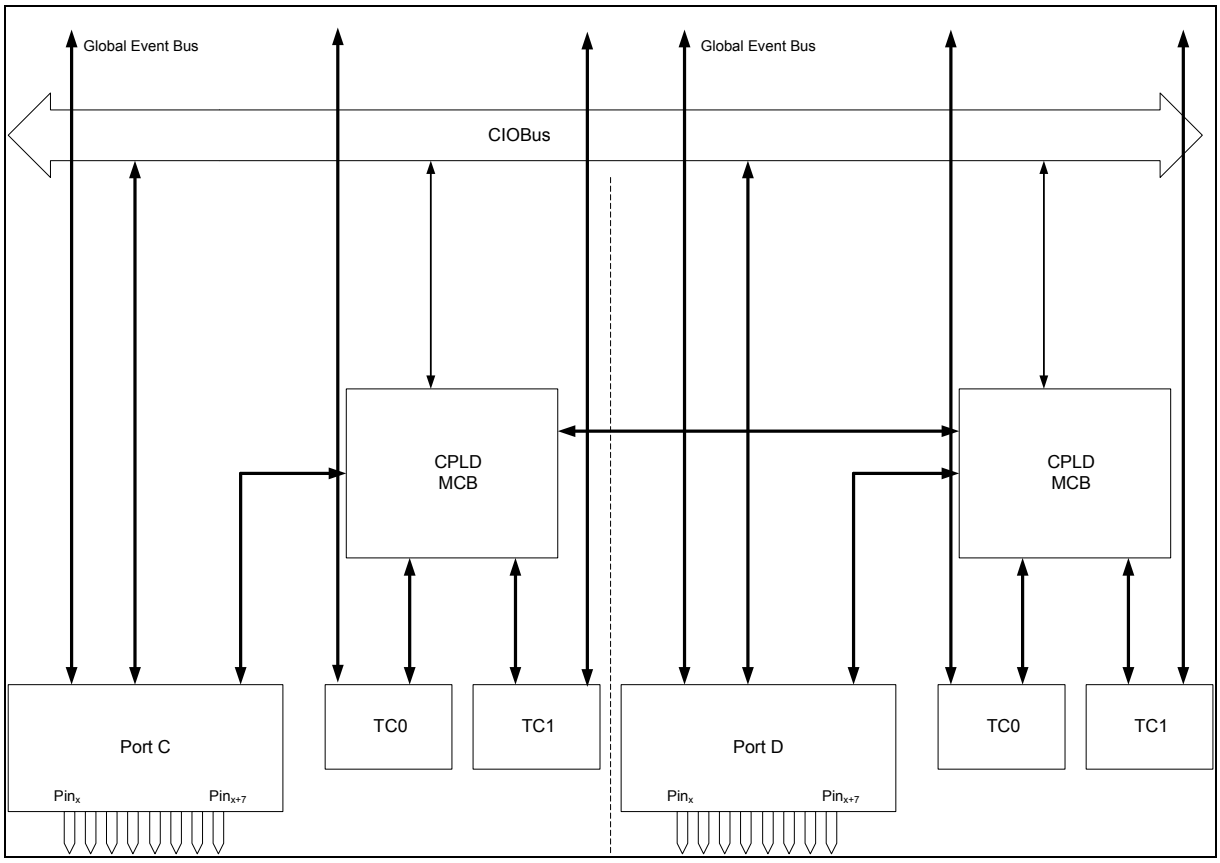


Figure 8.2: CERN architecture regional view

Each Port provides connectivity for eight I/O-pins accessible for the CPLD MCB either through direct connection with the Port or as selected channels from the CIOBus.

8.2.1 CERN Local Level Architectural Description

In the ATF1508RE 16 macrocells are provided for each logic block, as shown in figure 5.11. At the local hierarchy in the CERN one modified macrocell represents the basic CPLD MCB. Each MCB can be expanded with additional interconnected macrocells if increased computational power is needed. The construction is illustrated in figure 8.3, also providing details on the interconnect structure of each macrocell. Not illustrated are the changes to internal connections for the macrocells computational logic and connection to the I/O pins.

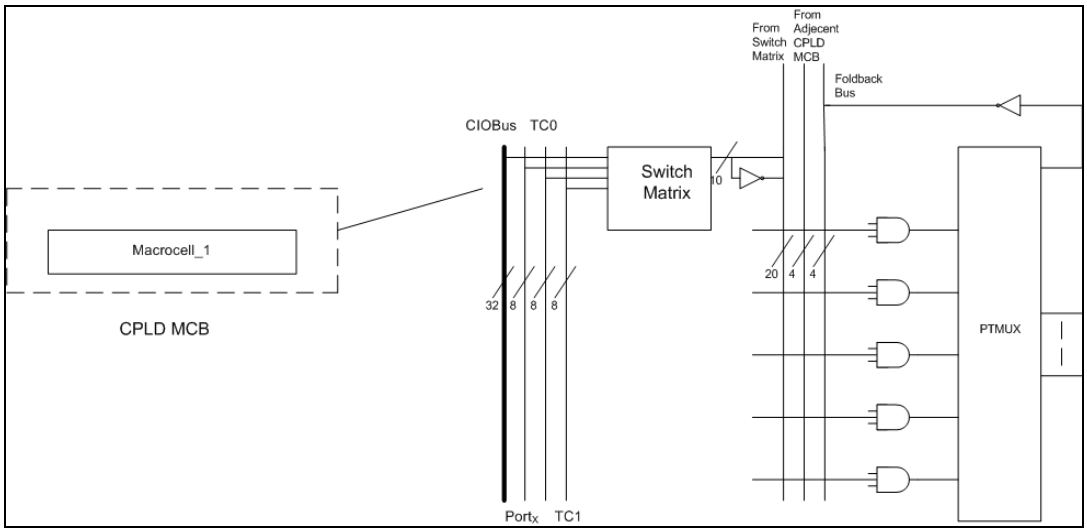


Figure 8.3: CPLD MCB detail showing internal connectivity

Figure 8.3 provides information on differences in input wires compared to the ATF1508RE. Wires from the original global bus described in [7] are substituted with wires from the CIOBus and Port / TC wires. A total of 56 wires connect to the switch matrix, which selects ten of these and produces the complement value in addition. Connecting to the PTMUX is the original AND-structure from ATF1508RE, with the addition of a set of wires from the adjacent CPLD MCB constellation. These four wires are included to connect CPLD MCBs during Event Computation, avoiding access to the CIOBus. With a total of 4 macrocells the CERN offers powerful capacity when it comes to handling I/O-signals as well as events. Because of the size issues discussed in section 8.5 the basic CERN architecture will only include one CPLD MCB with one included macrocell in each Port / TC peripheral. With a rich access to cascaded and bus distributed signals, a wide variety of product terms can be realized making this architecture powerful despite the modest number of computational cells.

8.3 Why Consider CPLD Based Processing?

Using architectural elements from the ATF1508RE one obvious reason to use CPLD based technology instead of FPGA technology is possible reuse of existing library elements from Atmel. This will greatly reduce implementation time compared to constructing new libraries of custom made FPGA elements. Reducing time-to-market for a proposed Event System solution, this factor may weigh heavy considering the economical aspect of system development. Some re-design must be considered in order to interface existing peripheral connections, but the amount of re-design is potentially smaller than for a custom FPGA design.

Considering functionality, there are also benefits using a CPLD based interconnect structure to form a routing and computational environment for the Event System. As mentioned in section 5.4 CPLDs can offer greater performance in terms of computational speed compared to FPGA structures. Connecting Logical Blocks with fast busses, the CERN could outperform an FPGA based routing topology. As mentioned in background theory the macrocells in the Logic Blocks consists of an

AND/OR-plane especially suitable for implementing FSMs [42] and logical expressions in the Sum-of-Products form. This architectural layout is perfect in terms of the adaptability needed to maintain flexibility with regards to routing for the Event System, while at the same time offer powerful facilities for logic computations. Implementing chains of events similar to the FSMs used in the MTIO system [8] would enable advanced peripheral operation with the possibility for reduced power consumption compared to the current Event System.

In terms of area and logic density the FPGA has a benefit over the CPLD using multiple logic levels with a smaller number of fan-in gates, to construct more compact logic than the two-level AND/OR plane used by most CPLDs [14]. However, the flexibility offered by FPGA style implementations is not demanded by the Event System, making a CPLD solution very competitive in this area as well.

8.4 Changes to Existing Peripherals

One benefit of using CPLD based processing for events and I/O signals, is the reduced need for changes to existing peripherals. A CPLD MCB block is added as an independent addition to a Port / TC constellation, and can almost be considered as an extra peripheral for these modules. The concept is illustrated in figure 8.2. In this way each CPLD MCB is modified to fit into the existing peripheral context, and not vice versa. Of course some extra synchronization and signaling procedures must be included in the existing modules, but in general these changes can be regarded as non-complex and small.

Although the peripheral blocks in themselves will feel only slight changes considering CPLD additions, there is a need for more memory to feature the programmability aspect of the implementation, and provide configuration memory for implemented FSMs. The same memory could also be used to alter CPLD parts of the Event System by reprogramming the running configuration. More details on this are mentioned [7].

8.5 Area Estimations and Programmable Bit Count

To compare the CERN to other proposed Event System solutions, estimations on both area and the amount of bits needed to configure the architecture is important. Because the CERN is not the chosen architecture for further development, the methods used to obtain area estimations are included in appendix 16.4. In this section only the logic area expressed in NAND equivalents will be presented. Total chip area including additional logic for interfacing and routing wires is included in section 12. Table 8-1 gives the required overview of logic area consumed if one CPLD MCB is included in each Port / TC peripheral.

Domain	# Pass transistors	# SRAM cells	**Gate count per element [NAND]**	Number of elements	**Total gate count [NAND]**
Input Switch Matrix (sparse)	224	224	175	4	700
Product Terms (full)	700	700	586	4	2344
Macrocell logic			130	4	520
Total					**3564**

Table 8-1: CERN estimated gate count for one CPLD MCB in each Port / TC peripheral

Table 8-2 shows the total amount of configuration bits needed to program the basic CPLD MCB architecture, with one macrocell block in each Port / TC peripheral.

Domain	# SRAM cells	# configuration bits	Number of elements	**# Total configuration bits**
Input Switch Matrix (sparse)	224	224	4	896
Product Terms (full)	700	700	4	2224
Macrocell logic				0
Total				**3120**

Table 8-2: Amount of configuration bits for one CPLD MCB in each Port / TC peripheral

A note to table 8-2 is that bits needed to program the internal macrocell logic with MUXes and the Product Term Allocation MUX are supported by registers, and does not influence the rewritable FLASH bit count.

8.6 Conclusions for a CPLD Based Solution

Relying on well known and developed bus technology, the CERN offers great prospects of fast interfacing with the existing Event System. However the results obtained from table 8-1 and table 8-2 outline some limitations to a CPLD based solution. Requiring a rather large amount of programming bits, for instance 7 times the amount required of the HERN, it is clear that routing resources consume much area and will require a considerable amount of FLASH memory for more advanced solutions. Such a solution is not cost effective and further analysis on the switch matrix and product term allocation crossbar design is recommended to decrease the configuration bit overhead.

Relying on technology already incorporated in the Atmel library, modifications should yield design-time benefits over developing new FPGA inspired solutions. Also acknowledging the benefits of easy FSM implementation and seamless integration with existing peripherals, the CERN architecture stands out as a potential direction for future Event System development.

9 The Asynchronous Event System

The following sections will describe important features of the proposed solution for an asynchronous routing network for the Event System. The main features of the routing network will be listed, along with a detailed description of the components used as building blocks for network services. A new encoding scheme for events is also proposed in order to seamlessly integrate events into an asynchronous distribution model. To include LUTs for logical event computation, both new peripheral functionality and how this functionality should be implemented into existing peripherals is discussed. The last sections will include some remarks on cost associated with an asynchronous implementation, and also give some conclusions on the feasibility of the proposed solution.

9.1 Features

- Fully asynchronous routing of events and other signals using the Asynchronous Event System Routing Network.
- Pipelined switches for fast transfer of data tokens.
- Local Port/TC event wires for fast event generation and minimal delay for local events.
- Potential savings in power consumption due to event driven consummation.
- Programmable routing and LUTs represents large flexibility and possibilities for self-programming of the system functionality.
- 4-Phased Dual-Rail encoding for hazard free routing operation and delay-insensitivity.
- Dedicated inter-peripheral wires for connecting LUTs effectively

9.2 Routing Network Description

9.2.1 General description

In order to secure a flexible and powerful routing facility, a hierarchical solution built on an FPGA inspired routing topology is proposed. The solution is inspired by the HERN explained in section 7.2, but with a different hierarchical solution. In this case hierarchical means parting the routing resources in a global- and local routing network, omitting the regional level, and FPGA inspired points to the fact that programmable SRAM based switches perform the actual routing. This structure is applied in an effort to utilize the locality of certain modules and the fact that these modules tend to have a significant need for local event communication.

The Asynchronous Event System Routing Network (AESRN) considers all communication between peripherals like ADC, DMA and Port/TC peripherals as global communication. Global connections are performed by the Asynchronous Global Event Routing Network (AGERN). Local communication is defined as event communication between different Port/TC constellations, or inside one constellation, and is provided by the Asynchronous Local Event Routing Network (ALERN). Figure 9.1 illustrates the hierarchical structure in more detail.
As depicted in the figure the local transition path for events makes it easy for a selected event to quickly be transported at a local level, with the possibility to be routed globally at the same time. This feature enables easy duplication of events, triggering different actions in different peripherals.

The AESRN routing facility consist of uni- and bi-directional switches instead of the multi- directional switches usually applied in FPGA systems. The benefits of this structure are switch modules of lesser complexity than switches applying multi directional routing schemes, and an easier determination of the event flow itself. Each switch is pipelined for fast event flow and applies a handshake protocol based on 4-PDR logic for encoding of the event signals.

The protocol is enforced by multiple buffers in each switch, also providing fine-grained pipeline possibilities. A similar switch structure is used in [1]. Details considering switch blocks and WCHB token-buffers are presented in section 5.2 and 3.5.2.

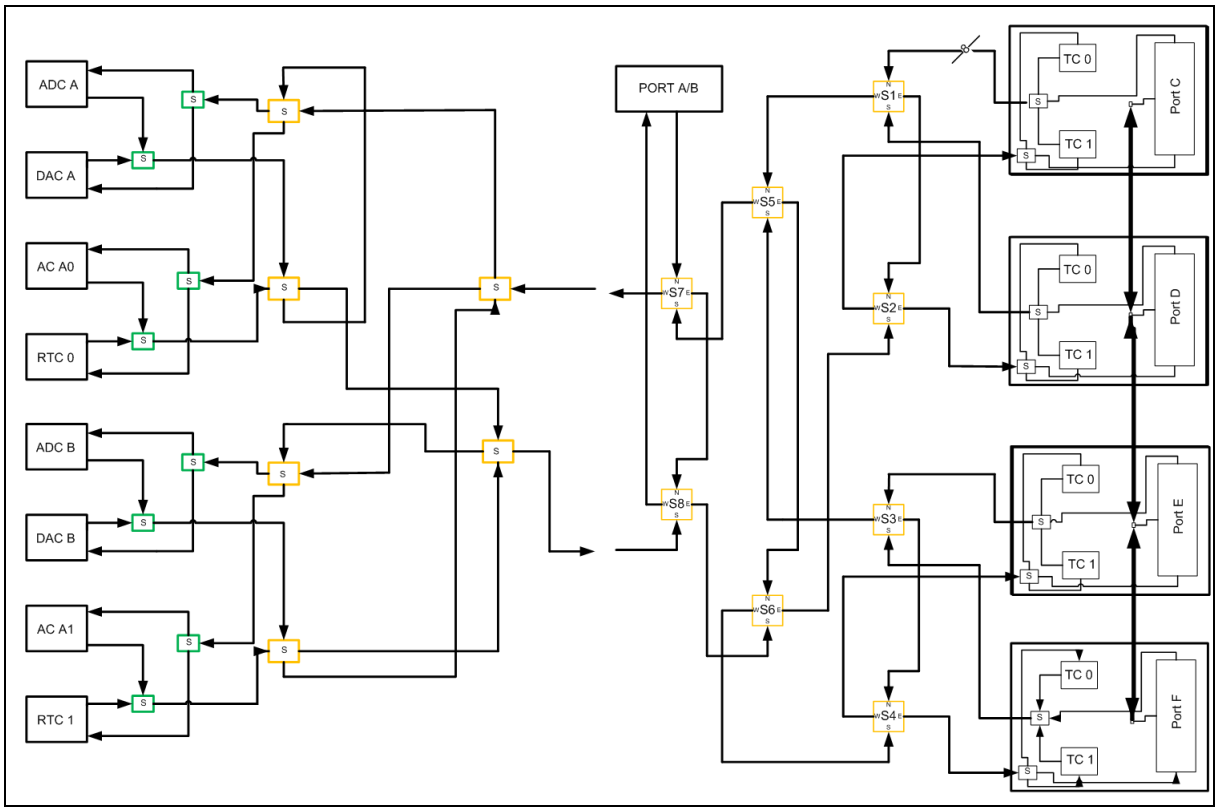


Figure 9.1: Illustration of the AESRN employing a hierarchical routing structure connected by switch blocks.

Due to the 4-PDR encoding, each bit transferred on an arbitrary routing channel is represented by two wires, and one additional wire is used to provide the correct handshake procedure in the form of an enable signal.

The architecture shown in figure 9.1 has a capacity of 8 event transactions concurrently in each direction at the global level using the AGERN. At the local level 12 channels can be routed arbitrarily to all ports or TCs via the ALERN, while also maintaining the sending and receiving of events on 8 global channels. The Asynchronous Functional Event Routing Network (ALFERN) is an alternative to the ALERN structure. It is designed for more flexible LUT integration and can route 8 unique event channels, but in a more flexible manner than the ALERN. ALFERN and ALERN design considerations are handled in the following sections.

Because of the bi-directional routing scheme, a peripheral can easily both receive and send an event at the same time, and switching technology ensures that each event channel can connect to all peripherals, regardless of which peripheral the event is issued from.

9.2.2 The Asynchronous Local Event Routing Network

The idea of implementing a dedicated ALERN is inspired by the system preceding the current Event System, the MTIO unit [8] and the proposed HERN solution. As mentioned in section 7.2 a hierarchical structure is also implemented to support fast event transportation between different Port/TC constellations, in addition to maintain fast connectivity between LUTs provided by TC buffer registers when this option is selected. This LUT-feature is explained in section 9.3.

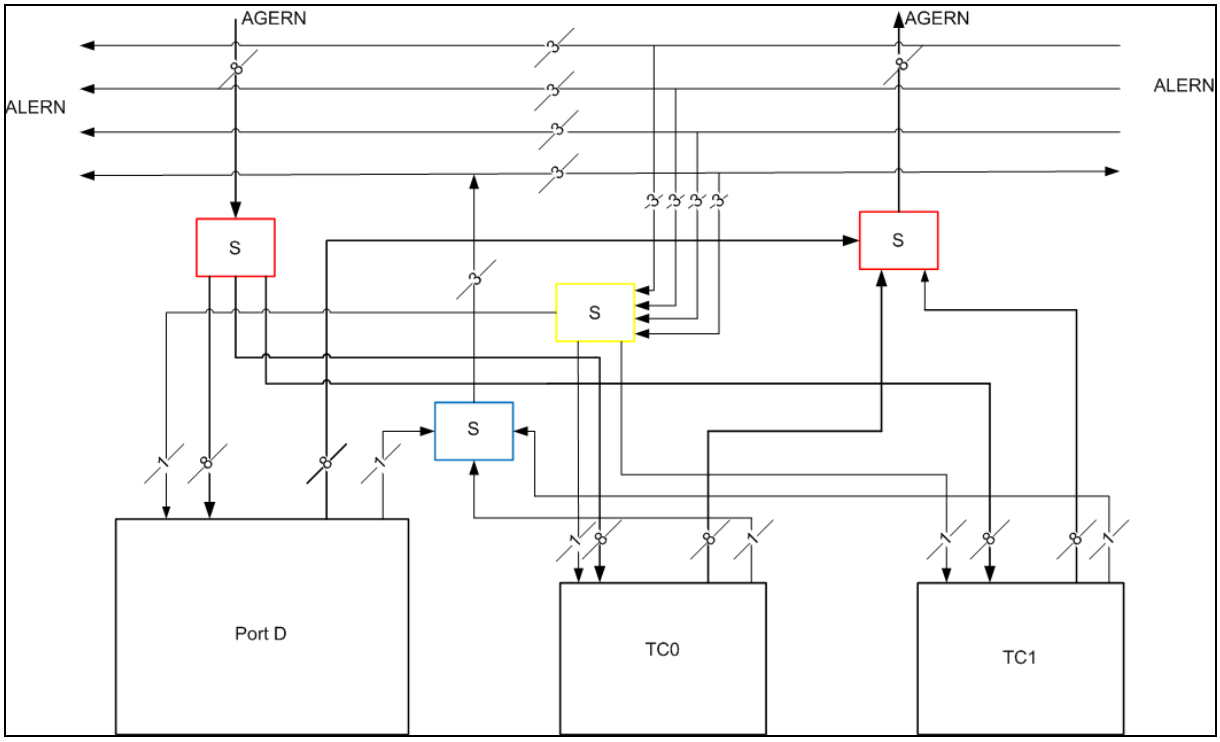


Figure 9.2: ALERN routing structure. Numbers indicate Event Channels.

The ALERN structure is identical for all featured Port/TC blocks on the specific XMEGA. Distribution of local events on the ALERN is maintained by the yellow central switch. Each of the twelve local routing channels connects to the switch, distributing all locally generated events to all Port and TC peripherals. Each Port/TC block also contains a switch like the one marked in blue on figure 9.2 , were each peripheral is allowed to issue one event on the local channel for that particular Port or TC block. This also implies that each of the port and TC peripherals has its own unique local distribution channel. In practice this means that the blue switch can be considered a solid connection to the ALERN, but controlled by programmable pass transistors and WCHBs to ensure correct handshaking. For details considering each switch type, the reader is referred to appendix 16.2.

The extra resources used to implement the ALERN, both with regards to programming bits and gates, are the most critical arguments when discussing if the ALERN should be implemented as a part of the event system, or if the AGERN is sufficient to provide all event services. However, the flexibility offered by the ALERN concerning functionality between ports and TC blocks should not be underestimated, and was one of the important aspects considered in the MTIO-system [8]. Apart from the offered flexibility, the ALERN has some disadvantages as well. The amount of local events from each peripheral is limited to only one, and the local distribution system breaks with the partitioned two-way routing employed in the AGERN. In addition, connecting LUTs are not trivial as they will reside in different TC-modules, with only limited accessibility through the local distribution network. Therefore a more functionally advanced local routing facility would be beneficial, with a simpler interface than the ALERN.

9.2.3 The Asynchronous Local Functional Event Routing Network

The new local routing network is called Asynchronous Local Functional Event Routing Network (ALFERN), developed specifically for increased local communication, allowing a Port/TC constellation to have eight local channels for event reception, and two channels for sending of events. The ALFERN is also developed in order to communicate with a Timer Counter Combined Functional Block (TCCFB), instead of two separate TC modules. In this way internal communication within the TCCFB can connect the registers as a small LUT network, while the ALFERN connects all Port/TCCFB blocks into a larger LUT network.
Communication between two neighboring Port/TCCFB blocks is favored with four direct communication channels on the ALFERN. Figure 9.3 illustrates the new routing network, and further details can be viewed in appendix A 16.1-2 and A 16.1-3.

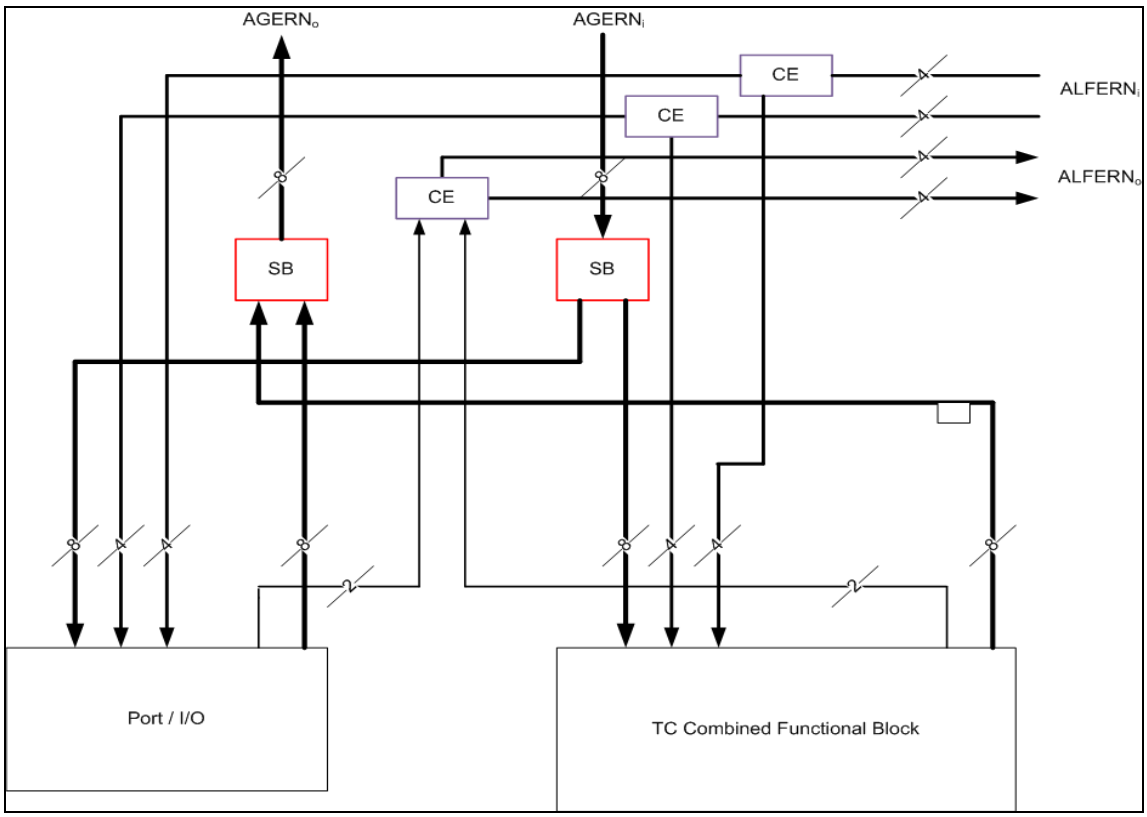


Figure 9.3: ALFERN overview, supporting the addition of TCCFBs

One featured change from the previous ALERN is a more prominent use of copy-elements to fork desired signals to multiple destinations, making duplication of events easier. Another feature is that the ALFERN is parted in two separate directions like the rest of the AESRN, not uni-directional as the ALERN, providing better architectural consistency.
Section 9.3.2 will examine connections and internal Event Channel distribution inside TCCFBs more closely.

9.2.4 Choice of handshake mechanism

To support a completely asynchronous transfer protocol there is a need for mechanisms preserving correct handshaking and hazard avoidance during signal transfer. As described in section 3.4 different protocols can be adapted in order to preserve handshaking consistency. According to [2] only the three protocols mentioned in section 3.4 are worth implementing. The choice for the 4-PDR protocol is mostly due to the signals *ev_d* and *ev_c* forming the event channel in the original Event System. By using the original wires with a slightly modified encoding scheme to fit dual-rail encoding, overhead due to extra wires are minimized.

Since the original event is encoded by a combination of the bit pattern on these two wires, a direct adoption to the 4-PDR protocol can be made with the only addition of an *enable* wire for each channel. This wire provides the needed handshaking facilities. 2- and 4- phased bundled protocols were considered, but the delay discussion featured in section 3.4 tilted the scales in favor of a 4-PDR protocol implementation.

Another factor favoring a 4-PDR solution is its common use in asynchronous FPGA architectures, especially in [1] [27], with scientifically proved results. Analysis methods for handshake element performance using the featured protocol is considered in [36], enabling the possibility for using the same methods to analyze the proposed asynchronous solution [35].

9.2.5 Asynchronous Event Channel and Pipelined Switches

The Event Channel used in the current Event System consists of the signals *ev_d* and *ev_c*, as described in section 2. Due to the introduction of an asynchronous transfer protocol which must be supported by each channel, there is a need for additional signals to ensure correct handshaking. As mentioned in the previous section, the 4-PDR approach is used to utilize the fact that the event system operates with the two wires *ev_d* and *ev_c* to encode an event.
Thus the only overhead represented by the proposed event channel is an additional enable wire for each channel, and some extra logic to provide the handshaking mechanism.
Since the *request* signal is encoded in the data signal itself, there is no need for a conventional *req/ack* protocol. This handshake mechanism is substituted by a more subtle *enable* signal telling the sender and receiver when the sending of data tokens can take place. The differences between the original- and the proposed event channel can be seen in figure 9.4.

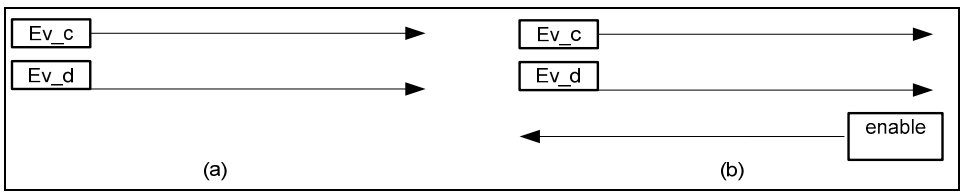


Figure 9.4: (a): Signals in the original event channel, (b): Signals in the proposed event channel using 4-PDR encoding

To connect channels distributing the events and handshake signals, the AESRN uses pipelined switches. Featuring one pipeline element for each switch point, the event channel in the AESRN represents a fine-grained pipeline. According to [1] and [15] fine grained pipelining yields a very high throughput, making rapid distribution of events possible.
A similar pipelined structure is used in the total asynchronous FPGA architecture presented in [1], but the AESRN implements these facilities in a new context using a totally different FPGA based routing topology.

As shown in figure 5.2 each switch block consist of a number of switch points, depending on how many channels which are distributed through that particular switch block. Because of the bi- or uni-directional routing schemes used in the AESRN, a custom switch layout is presented for all the different switch types introduced to the network. The proposed switch blocks can be studied in appendix 16.2. Each switch point consists of a WHCB-element, as described in section 3.5.2. This element provides all the needed handshaking facilities and the buffer capacity for one token needed by the pipeline at that switch point. Figure 9.5 (left) shows the design of a custom switch point including one WHCB-element, for illustration purposes notated as Synch Buffer because of its functionality in the circuit.

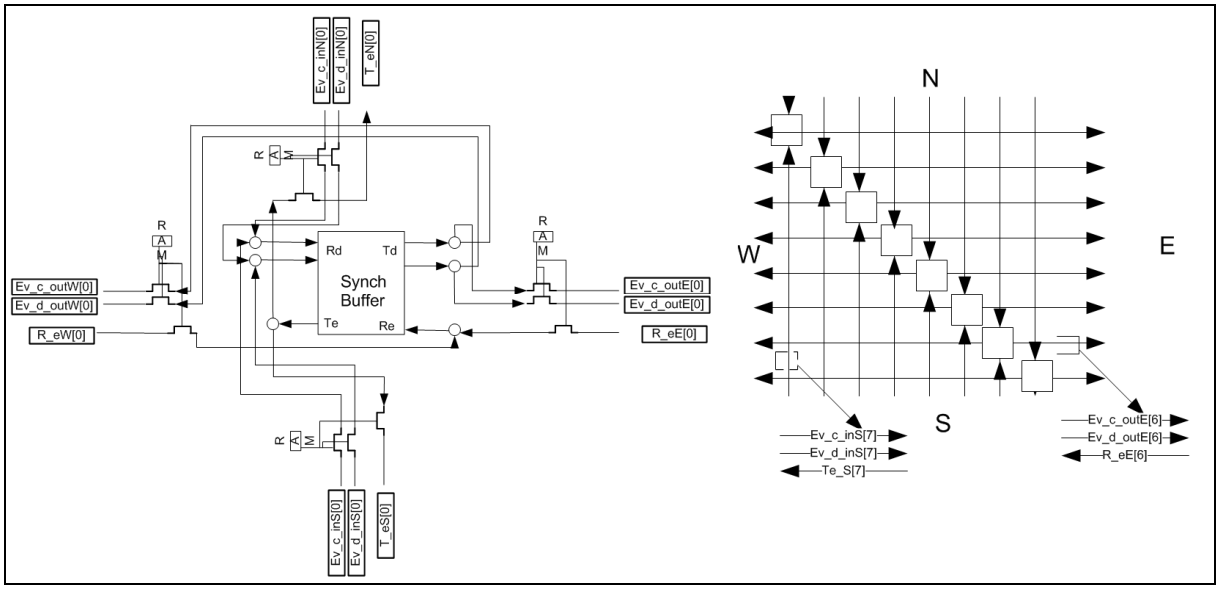


Figure 9.5: Bi-directional WCHB switch point (left). Switch block input / output directions (right)

A typical bi-directional switch block used in the AESRN consists of eight channels in each direction, and therefore needs to implement eight WHCB-elements, one for each switch point, to preserve the pipelined structure. The handshake procedure between receiving and transmitting WHCB-elements must be completed for one event token, before the next can arrive at the buffer. Combined with the 4-PDR protocol the result is a robust transfer scheme for events. This is an important ability of the proposed solution considering some of the vital services provided by the event system. The comparison of WCHB and PCHB elements is covered in section 3.5.2. Being smaller in logic area and optimized for less advance token computations than PCHBs, WCHBs are the preferred elements for pipeline switch blocks in the AESRN.

Using the applied uni- or bi-directional switches instead of the usual multi-directional switches implemented in most FPGA architectures shows its simplicity in the context of figure 9.5 (left). Since each WHCB-element are only allowed to provide handshaking service with one particular source per data token transfer, only one event channel can be operated by one switch point at the same time. The consequence is that one SRAM cell can control all three pass transistors in one channel, saving significant area concerning RAM cell implementation and introducing a simpler programming interface due to fewer programmable bits.
If forking of signals are practiced at some sources, with a switch point as destination for each branch, a copy-element act as one signal source for each WCHB-element and combines the handshake signals into a single signal. In this way one WCHB-element seemingly never communicates with more than one element directly, although the multiple elements are part of the transaction. For instance copy-elements are used actively in the ALFERN distribution network described in figure 9.3.

9.2.6 Event Decoding

Due to the usage of 4-PDR encoding, an event must be encoded in a different way than in the original event system. As presented in chapter 2 each peripheral interpret an event in accordance with the combined bit pattern presented by the *ev_c* and *ev_d* wires. In order to provide consistency with the 4-PDR protocol, both wires are now interpreted as one event bit instead of representing two separate bits. In the proposed encoding scheme two proceeding event bits encode either a positive event or a negative event, or signals that a special event has occurred. A longer sequence of event bits represents different types of special events. Usually this pattern should start with the event bit combination signaling a special event, in order for a peripheral to uniquely identify what event type it will receive. If a peripheral receives "1" as the first event bit, the event type is either a positive or negative event. If the first event bit is "0" a special event of some sort can be expected, normally represented by a longer bit sequence. This method varies slightly from the representation used in [6], but contains the same functionality and flexibility considering different event types.

Table 9-1 presents the new event encoding scheme, and how the different bit combinations of the wires *ev_c* and *ev_d* is interpreted as an event bit.

ev_c	ev_d	Event bit	Comment
0	0	None	Not valid for encoding
0	1	0	Special Event
1	0	1	Event Occurred
1	1	None	Not valid for encoding

Table 9-1: 4-Phased Dual-Rail protocol and Event bit coding

Figure 9.6 gives a more detailed description of the event bit and how it is interpreted by a peripheral.

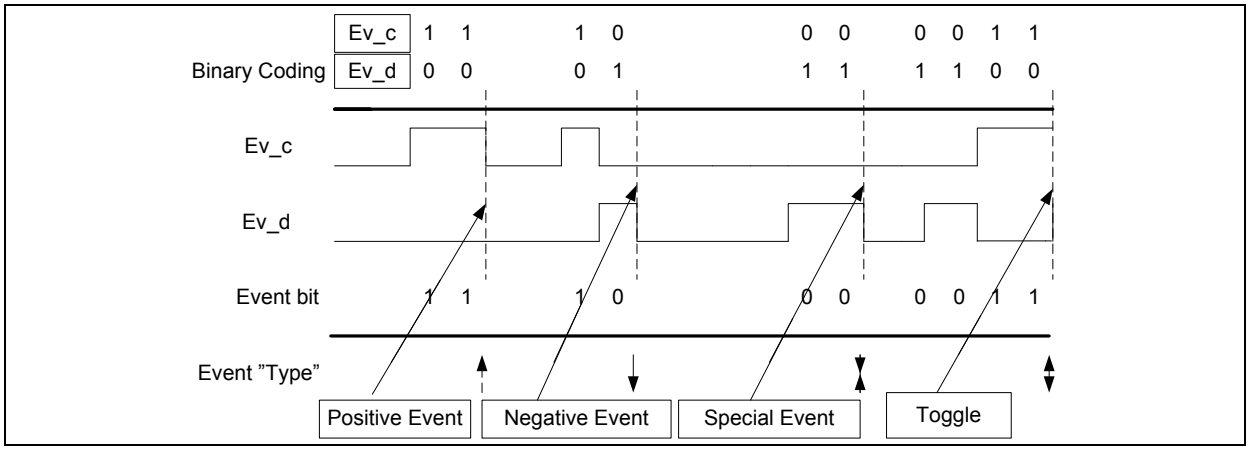


Figure 9.6: Encoding of different event types

As the figure indicates, a positive event is encoded by the event bit sequence 11, a negative event is encoded 10 and a special event is encoded 00 with usually some proceeding bits. The toggle event explained in [6] and [8] is supported by some peripherals, and is encoded by a longer sequence of Event bits. One possible encoding is the one used in the figure, "0011".

9.3 Changes to Existing Peripherals

In order for each peripheral to communicate and operate effectively with the AESRN, some changes must be made to the existing Event System interface. The most prominent changes that must be applied to all peripherals are the implementation of a handshaking interface towards the AESRN. This section will also describe some of the increased functionality characteristics for some peripherals, and the logic needed to support these functions. TC-blocks will be especially considered, explaining how they can be improved to support new functionality like implemented LUTs.

9.3.1 Changes to original Timer/Counter peripherals

One of the basic principles of the MTIO-system [8] was to give extra functionality to the TC blocks in order to efficiently process I/O signals at a local level, and use the results to control operations in other peripherals via the Global Event Routing Network (GERN). Some of this functionality was sacrificed in the development of the event system [6] but the flexibility provided by programmable switches in the AESRN provides the fundamentals for improving the functionality currently implemented and adding new features.
One of the proposed features is to use internal registers in each TC peripheral to implement 3- or 4-LUTs, and use the ALERN as a routing facility for communication between the LUTs. An important remark to this solution is that the TC modules operate independently, and are not considered TCCFBs which are only supported by the ALFERN and will be explained in section 9.3.2.

The four TC-blocks used in the original event system have the ability of double buffering associated with each Capture and Compare (CC)-register. These registers are denoted CC_XBUF, and for each of the maximum four CC-channels in a TC module a buffer register is present for use when buffering new capture- or compare- values. TC0 has four CC-channels and includes four CC_XBUF registers, while TC1 includes two buffer registers of 16 bits each. Since all CC_XBUF registers can be used independently of its corresponding CC register, and be accessed via the I/O bus, each register can implement unique 16bit functionality when unused for buffering. The register can be programmed to hold the functionality of a 4-LUT or two 3-LUTs only implementing some additional handshake computation logic and a 16:1 MUX. Figure 9.8 shows the proposed solution, in the form of a modified PCHB handshake environment. The combined computation environment and register will be denoted as a Timer Counter Computational Block (TCCB). If all the needed routing facilities are considered together with the TCCB, the combined block is denoted Timer Counter Functional Block (TCFB) which is described in section 9.3.2. The implementation is based on some of the principles presented in [27], described in background theory section 5.3.

TCCB Functional description:
The extra logic used for the LUT implementation is to provide a valid and hazard free handshake mechanism and computation environment. Four selected channels routed by the ALERN connects to an Input Valid logic block , containing four 2-input XOR elements connected a tree of C-elements in order to make sure that all inputs have arrived with a valid dual-rail value before any further logic processing is allowed. The proposed implementation of the Input Valid block is illustrated in figure 9.7. As described in table 9-1 the signal *ev_c* is deciding whether the arriving event bit is encoded as a “0” or “1” by the dual-rail wires. Therefore the *ev_c* wire for each channel controls the input to the MUX selecting LUT outputs. A possible problem arising is that a non-valid {00}-spacer value selects an output from the LUT, but because of the Input Valid block the LUT output will never be transmitted if this occurs. This is because a “00” value fed into an XOR will generate a “0”, causing the C-element to hold and not set the *input_v* signal in figure 9.8 to “1”.

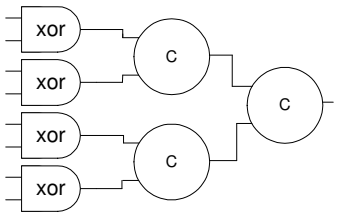


Figure 9.7: The input valid block

Since the implemented dual-rail protocol always returns to the “00”-input position after a handshake procedure, the Input Valid block will always be reset for each transition.

To create a dual-rail output from the 16:1 MUX, the output signal is split in two rails using an inverter on one of the rails. In order to secure the handshake procedure with the receiving and transmitting circuit, asynchronous latches are used, triggered by an enable signal controlled by handshake elements. One of the modifications from [27] is the addition of inverters on the *ack* signals in order to create the *enable* signal used in the AESRN. The latches will only fire when all input values are valid, and the LUT has finished its computation. The latch is reset by the handshake mechanism connected to the receiving buffer, or by a global reset. The TCCB in figure 9.8 forks its LUT output in two dual-rail branches, and an appropriate two-input copy-element is placed to receive the R_e_A and R_e_B signals from receiving elements.

To control the trigging of each CC_XBUF register, the “*write enable*” signal used in [6] to trigger the register are now combined with a new *LUT_enable* bit which can be set for each register independently. These bits are XORed so that the conflict of both bits set at the same time cannot occur.All registers can bypass the LUT circuitry, and operate directly with the associated CC register as normal, when buffer functionality is required.

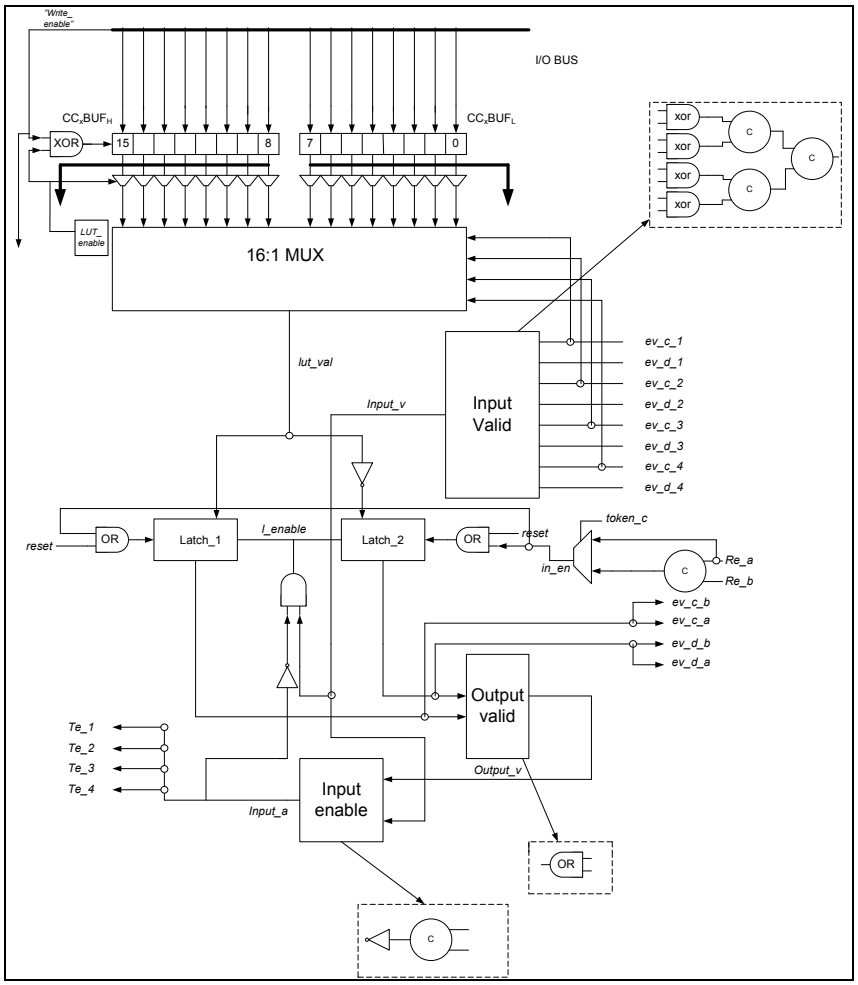


Figure 9.8: Register implementing 4-LUT fuctionality and handshake logic as a TCCB

Since *enable*-triggered latches are used, there are some timing issues involved. The timing conditions are presented in [27], and will only briefly be reviewed here. The first timing constraint involves the two latches and their respective setup-times. From an input arrives at the Input Valid block, there

must be a delay compensation between the computation of the LUT and the Input Enable result. A difference here can make the latches trigger an invalid result.

Suggested measures to prevent hazards are appropriate transistor sizing, or adding a small buffer after the inverter connected to Latch_2 in the figure. The hold constraints of the latches are met because of the handshake mechanisms between sending and receiving pipeline stages, which means that the received *enable* signal only will represent an acknowledge when the output of the latches are deemed valid. For other timing constraints the reader is referred to [27].

9.3.2 The Timer Counter Combined Functional Block

In order to offer a higher rate of flexibility, an alternative to keep the original TC1 / TC0 structure is to merge the TC-blocks and create a Timer Counter Combined Functional Block (TCCFB). The TCCFB includes all previous TC functionalities, but also provides the possibility for being interconnected as a LUT network allowing logic computation of events at a local block level. Introducing an internal DIRECT signal for each TCCFB block, fast logical computations based on event bits are possible, and the result can be routed on both the ALFERN and the AGERN. All signals available on the ALFERN and AGERN can be used as inputs and outputs to the TCCFB. In this section the TCCB from section 9.3.1 will be modified with some additional routing resources, and hence be denoted TCFB in this section.

General TCCFB Description

Figure 9.9 shows a block schematic overview of the overall TCCFB system, while figure 9.10 depicts a more detailed view of one TCFB connected to a buffer register, and the internal connection of different TCFBs. The original CC_XBUF buffer registers for TC0 and TC1 explained in [6] are used, with extra logic to handle computation, routing and handshake. For improved LUT functionality an internal signal called DIRECT is added for each TCCFB, only valid inside that block, but with the possibility to be routed both on the ALFERN and AGERN networks. There exists one DIRECT signal for each TCFB inside the TCCFB as can be seen in figure 9.10. $TCFB_A$ to $TCFB_D$ are formed based on registers from TC0, while $TCFB_E$ and $TCFB_F$ are constructed of registers provided by TC1.

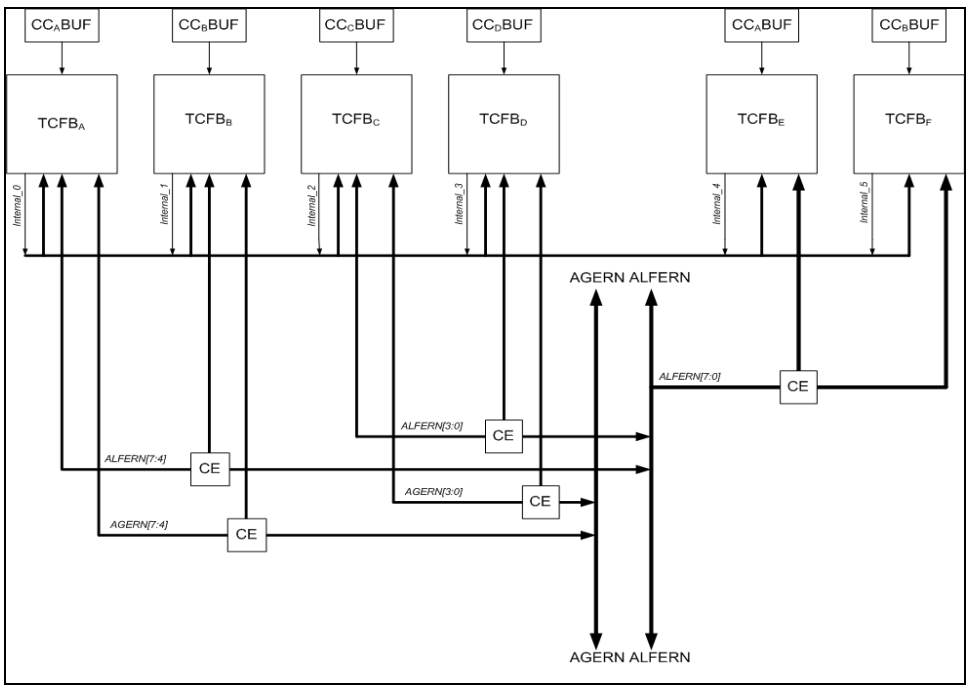


Figure 9.9: Overview of the TCCFB, showing involved TCFBs and signal connections

Not all TCFBs can be addressed by all AGERN, ALFERN and DIRECT event wires. For the special registers $CC_CBUF_{H/L}$ and $CC_DBUF_{H/L}$ only found in the original TC0-block, AGERN[3:0] and ALFERN[3:0] can be used as input to the Logic Block out of the eight possible AGERN and ALFERN wires. The registers $CC_ABUF_{H/L}$ and $CC_BBUF_{H/L}$ are accessed with AGERN[7:4] and ALFERN [7:4]. All four registers can be accessed by all DIRECT signals. For TCLBs supported by TC1 registers, only ALFERN and DIRECT wires can be used as LUT input. These restrictions are maintained in order to limit the amount of routing required, while still making almost any logic combination of events possible.

To prevent handshake violations in places where signals are forked, copy-elements are used in all such junctions.

Description of the Timer Counter Functional Block

The handshake and input/output blocks are the same as in the solution presented section in 9.3.1, but now supporting DIRECT signals between TCFBs and a more sophisticated routing scheme for ALFERN and AGERN signals. The new internal routing structure is maintained to make a more complete network of internal LUTs, interfaceable to all input and output signals from the AESRN. Each TCLB has its own Input Valid block as shown in figure 9.10, using programmable switches to transfer signals correctly both to the LUT-inputs and the Input Valid block. These switches are un-pipelined, since the handshake mechanism is provided by the TCFB itself. As described in section 9.3.1 only the *ev_c* wires are valid for LUT input, while both e*v_c* and its corresponding *ev_d* wire are needed as input for the Input Valid block to validate all input values. This fact employs the same SRAM-cells to manage both the *ev_c* switch, the *ev_d* switch destined for the Input Valid-block and the t_e-switch transmitting enable signals to the correct receiver.

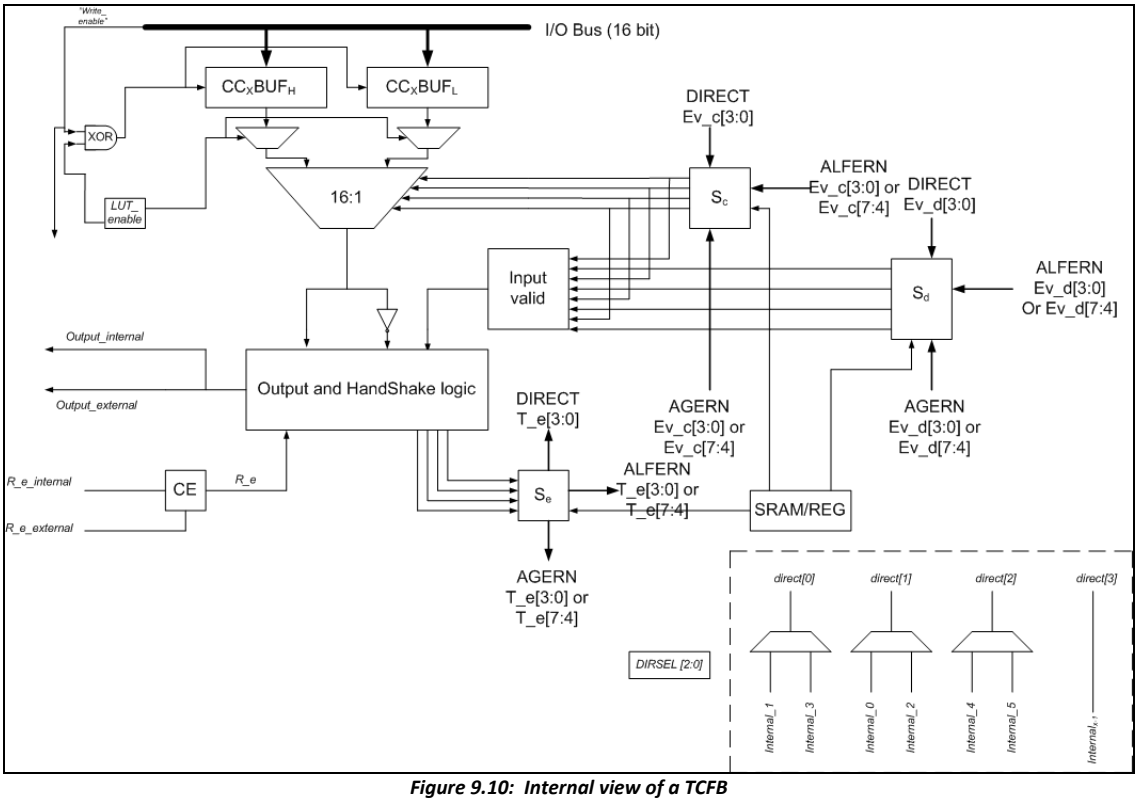


Figure 9.10: Internal view of a TCFB

Considering the output, each TCFB has its own dedicated DIRECT wire and can also connect to a selected AGERN or ALFERN wire.
The receiving handshake mechanism for *Re_internal* and *R_e_external* is the same as the one presented in figure 9.8.

Scenario description

First it is assumed that the ALFERN is used to route events to the LUT, for instance implemented in the $CC_xBUF_{H/L}$ register. Four different event bits are issued from different peripherals (which practically means four different events), or from four IO-pins on one of the ports. The event bits are routed to the correct TC peripheral using the AGERN, and the bits are processed by the Input Valid block. For illustration purposes the LUT is implemented to only issue a "1" if all input bits are "1". This is equivalent to the fact that four events must occur before the output will give a valid "1". In all other cases the output will be "0". When the condition occurs, the latches will receive the single rail output from the LUT as a dual-rail input because of the splitting of the single rail signal by en inverter. Latch_1 will send a "1" to the output valid, while latch_2 will send a "0". This output is valid, and together with the input enable signal an *enable* will be produced to trigger the latches and route the values to channel outputs. The Input_ack circuit will produce a "1" to verify the ack, and the inverter makes this a valid *enable* signal after the convention used in the AESRN. The "0" sent to the transmitting buffers signals that the received data is under processing. The transmitting buffers set their outputs to "0", issuing a reset token or spacer. This causes the Input_valid block to issue a "0", resetting the enable signal for the LUT environment and, a "1" will be transmitted as the *enable* signal to receiving buffers elements telling that more data can be sent.

9.4 Hazards

Section 3.2 gives an overview over the most common hazards in the asynchronous design domain, and will form the fundamental knowledge for dealing with the hazards in the AESRN.
Being a completely asynchronous system, the AESRN can in theory suffer from all the hazards mentioned in section 3.2, but in practice the only real hazard problem is concerning metastability. By using hazard-free Müller-C elements for the handshaking buffers, static and dynamic hazards concerning signal transitions are eliminated. The Müller-C element is the most important component in the AESRN considering routing resources, because of its prominent role in the WCHB pipeline elements. The proposed design for a hazard-free Müller-C element is presented in section 3.5.

The other hazard directly concerned with the routing and handshake procedure is the "*data before request"*- hazard described in section 3.2. The applied 4-PDR encoding of the events introducing the new event bit makes sure that the timing hazard will never happen in the AESRN.

Since the AESRN in the context of the XMEGA will be implemented as an asynchronous distribution system with an interface to the peripherals synchronous domains, the metastability hazard will be an important implementation issue. For prototype and concept purposes the two-flop synchronizer mentioned in section 3.2.1 should provide a safe, simple and reliable architecture for controlling hazards connected to metastability. Since the AESRN pipeline implements WCHB buffers enforcing the 4-PDR protocol, the two-flop solution from figure 3.5 can be modified in terms of state machine implementation. Only on the synchronous receiver side are state machines needed, and they must enforce the same 4-PDR protocol as used in the buffers. Sending data into the pipeline can occur at arbitrary times because of the QDI dependent construction. Because each WCHB element implements a 4-PDR protocol, synchronizers are only needed in the interface between event channels transmitting data to the synchronous XMEGA peripheral environment.

9.5 AESRN Measured Gate Count and Programmability Cost

The architectural measurements presented in this section are derived from the architectural concept drawings, and do not represent a synthesized gate count. Synthesis results are reviewed in section 10.5, and are mostly considered for orientation. Appendix 16.2 provides the detailed sketches for each custom switch block, and a measured gate count for each individual block.

Appendix A 16.1-2 - A 16.1-5 shows the complete setup of switch blocks in the AESRN, and an equivalent NAND-gate count for each section. Table 9-2 and table 9-3 summarize routing resources based on these appendixes. The gate count results for a full Asynchronous Event System implementation, including full range of computational TCCFB functionality are presented in Table 9-4 - Table 9-6. Architectural drawings for these tables are featured in appendix A 16.1-6 - A 16.1-8. In section 12 the total area will be compared to the HERN and CERN solutions, and area associated with routing and additional logic will be added.

Domain	Switch type $\#CH_{In}/\#CH_{Out}$	# SRAM transistors switch block	# WCHB transistors switch block	**Gate count per switch block [NAND]**	Number of switch blocks	**Total gate count [NAND]**
AGERN	8x2/8x2	288	240	117	14	1638
	8x2/8x1	144	240	101	8	808
ALERN	8x3/8x1	216	240	109	8	872
	12x1/3x1	324	90	68	4	272
	3x1/1x3	45	90	35	4	140
Total						**3730**

Table 9-2: Overview over resources used to implement the AGERN and ALERN, measured in NAND equivalents

Domain	Switch type $\#CH_{In}/\#CH_{Out}$	# SRAM transistors switch block	# WCHB transistors switch block	**Gate count per switch block [NAND]**	Number of switch blocks	**Total gate count [NAND]**
AGERN	8x2/8x2	288	240	117	14	1638
	8x2/8x1	144	240	101	8	808
ALFERN	8x3/8x1	216	240	109	8	872
	Copy element			22.67	14	317
	8x2/8x1	144	240	101	2	202
Total						**3837**

Table 9-3: Overview over resources used to implement the AGERN and ALFERN, measured in NAND equivalents

Domain	Computation Logic	**Gate Count [NAND] per TC LUT**	Number of TC LUTs	**Total gate count [NAND]**
TC LUT	Handshake	42	24	1008
	LUT + latch	38	24	912
Total		**80**		**1920**

Table 9-4: Estimated NAND gate equivalents for Timer Counter LUTs

Domain	Computation Logic	**Gate Count [NAND] per TCFB**	**Total gate count [NAND]**
TCFB	TC LUT	80	80
	Switches	50	50
Total		**130**	**130**

Table 9-5: Estimated NAND gate count for Timer Counter Functional Block

Domain	Computation Logic	**Gate Count [NAND] per TCCFB**	Number of TCCFBs	**Total gate count [NAND]**
TCCFB	TCFBs	780	4	3120
	CEs	227	4	908
Total		**1007**		**4028**

Table 9-6: Estimated NAND gate count for Timer Counter Combined Functional Block

The biggest overhead when it comes to gate equivalents is due to the WCHB pipeline buffers in the switches. SRAM cells are relatively small units when implemented from the standard cell library, and contributes only to about 25% of the total size. The measured NAND equivalent for a WCHB element like the one shown in figure 3.6 is approximately 10.05.
A simple calculation example for a switch block connecting 32 channels (8 channels on each side) creates a need for 8 WCHBs, costing 80.4 NAND gates. Implemented as a uni-directional switch 8 SRAM cells plus associated pass transistors are needed, equivalent to approximately 24.12 NAND gates or 25% of the total area contribution.
Routing area and area for additional logic is not considered in this section, but will be included in chapter 12 when comparing the Asynchronous Event System with the other proposed solutions.

Much of the gate overhead is due to the fine-grained pipelining, with WCHB pipeline elements in each switch point in every switch. This is a tradeoff in terms of system performance, since a coarser grained pipeline suffers from a lesser data transfer- and computation rate [1]. With the present structure event bits can be sent for every handshake transition, which ensures superior distribution capacity.

Supporting routing from an arbitrary peripheral to all other peripherals, the proposed AESRN requires a rather large amount of programming bits. The current AESRN constructed of AGERN and ALFERN needs 924 bits to control all the programmable switches due to the applied SRAM technology, and these bits will in most cases need to be supplied by FLASH memory. A full ALFERN implementation with TCCFBs requires approximately 1724 bits to be fully programmed with full range of LUT functionality. The estimated size overhead due to configuration bits will be fully explained in chapter 12.

Domain	**# FLASH Programming bits**
AGERN Port / I/O side	256
AGERN ADC / DAC side	320
ALERN	348
TC LUT	0
Total	924

Table 9-7: AESRN programming bits with ALERN without LUT functionality

Domain		# FLASH Programming bits
AGERN Port / I/O side		256
AGERN ADC / DAC side		320
ALFERN	Logic	208
	Copy Elements	40
TCCFB	TCFB (24 instances)	600
	Copy elements	160
Total		1724

Table 9-8: AESRN programming bits with ALFERN and TCCFBs

9.6 Conclusions for the Asynchronous Event System

Considering area the Asynchronous Event System will have a disadvantage compared to a synchronous solution because of the extensive handshake logic. WCHB elements used for token buffering occupy 75% of the switch block area, and only 25% are used by SRAM related logic like memory and pass transistors. The benefit of fine-grained pipelining is increased performance, which will be further analyzed in section 11. Relative cost increase is therefore an application specific factor, depending on the desired speed versus area ratio. Non-clock driven timing is also a bi-product of the cost increasing handshake elements, and must be included when important benefits are discussed. With no need for a complex clock-tree layout could be simplified, and both static and dynamic power consumption should be reduced because the circuit will only consume power in circuit areas taking part in a computation. Also supporting a more flexible routing structure than the current Event System, more events can be distributed concurrently with reduced latency for local events between Port/TC - peripherals. Computational LUT-elements are implemented inside existing Port/TC peripherals for added computational properties for events.

All mentioned benefits are good additions to an Event System, but does not come unconditionally. Increased area is one important factor, but complexity when it comes to development, testing and tool-chain addition are other costs which to some extent will be increased by introducing the Asynchronous Event System. Hazard issues when interfacing with the synchronous peripherals adds to the complexity, making it unrealistic to believe in an asynchronous Event System as a “near-future” Event System candidate.

With this conclusion in mind, the next sections will shed further light on benefits and drawbacks of an asynchronous Event System, and analyze the developed Verilog model as well as performance and market related cost factors. Only with these calculations in mind a proper conclusion on the feasibility of a fully asynchronous solution can be derived.

Section III –

Simulation and Implementation

10 Implementation, Simulation and Synthesis of the AESRN

This chapter considers the implementation aspects of key elements regarding an asynchronous Event System. Although a realistic implementation is beyond the scope of this thesis, a Verilog model illustrating important functional properties of the AESRN has been designed, simulated and synthesized for the Xilinx Spartan XC3S1000 FPGA. The following subsections will describe important design issues, simulation results and how the AESRN is programmed for a constructed event distribution scenario. Concluding this chapter are some remarks on synthesis results for the Spartan 3 FPGA.

10.1 Designing an AESRN Verilog Model

In a microcontroller environment the scenario of distributing events through programmable switches adopting an asynchronous transfer protocol with pipelining illustrates a new concept. To understand how the event distribution should be handled in such an environment, a Verilog model providing basic distribution and computational functionality was designed. The model features a fully programmable version of the Port/TC side AGERN and fully programmable ALFERN with simple TCCFB LUT functionality. The purpose of the designed model is to show basic AESRN functionality, and a full range of programmable functions is therefore not included. Figure 10.1 shows the scenario entitled to the design.

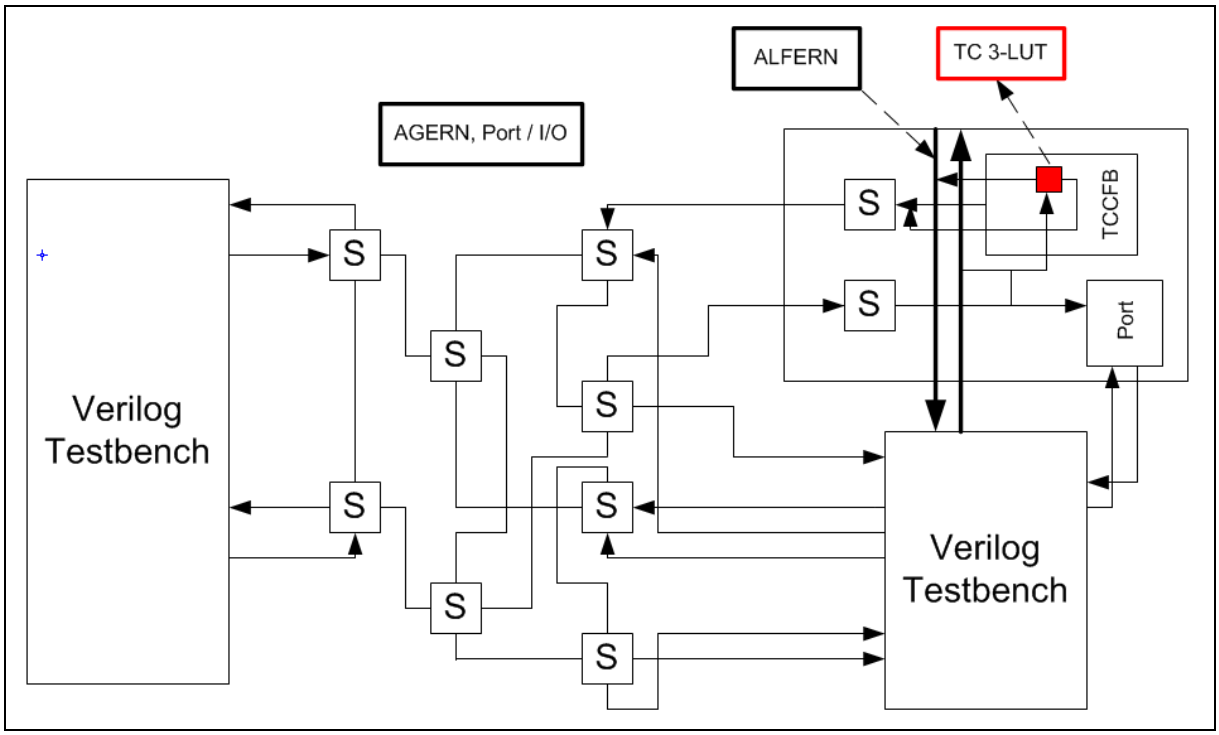


Figure 10.1: AESRN design scenario

A Verilog testbench interacts with the model as the ADC/DAC-peripheral side of the AGERN, along with providing I/O signals to the port and interact with the ALFERN. Since only **one** Port / TCCFB constellation is constructed for test purposes the testbench works as all excluded Port / TCCFB instances for the AGERN. One 3-input LUT is constructed inside the TCCFB block, and receives input events from both the ALFERN and AGERN through the TCCFB interface. LUT results can be distributed on the same connections. Copy elements for the ALFERN are not included in figure 10.1, but exist in the model as described in figure 9.3. Extending the Verilog model to include several Port / TCCFB constellations is trivial, but is excluded from the model to maintain simplicity.

10.1.1 Simulation Issues

As many papers have claimed, among them [13], designing systems in the asynchronous domain is quite a difficult challenge compared to synchronous system design. The author experienced this fully when trying to test the different modules, because of the simulation programs notion of time. When operating in an environment with no clock to synchronize the signals, the simulator does not propagate in time. All asynchronous transitions are simulated in the same real-time instant using deltacycles. Concurrent processes are simulated at the same deltacycle, and the next time the process is evaluated the deltacycle count is incremented by one. The consequence is that simulation results do not appear on the waveform, since there is no actual notion of time.

To overcome this problem a clock is used in the testbench to synchronize inputs to the test environment, while all handshakes are completed during the clock cycle. Since these handshakes are completed on a deltacycle basis, they will always finish before the clock checks the conditions for the next valid input. Conditional if/else-statements are used to control the handshake procedure from the testbench when interacting with pipeline elements. One stage of the handshake process is completed for each clock cycle, meaning that from one data token enters the pipeline to the next data token enters, a duration of two clock cycles have passed. Also using an additional delay in the WCHB elements of 5ns to verify the token flow in the pipeline, the testbench provides a good notion of the model's behavior.

Another issue it is very important to emphasize is that glitches are very hard to detect during simulation. Glitches due to dynamic circuit behavior have yet to be detected, although it is very probable that they exist. This fact exposes a weakness in the simulation model making it hard to verify a circuit module although the simulated behavior is seemingly correct.

10.2 Design of Asynchronous Circuit Elements in Verilog

10.2.1 Designing a Müller C-element

As explained in section 3.5.1 the Müller C-element is the most basic circuit element in most asynchronous designs. A Verilog model of this element has therefore been constructed for the AESRN, inspired by the solution presented in [30] and figure 3.9. Based on the theoretical assumptions from [30], this is a hazard free implementation as long as the 4-PDR handshake protocol is enforced correctly. Simulation results for the C-element are presented in appendix A 16.3-1 and the Verilog code can be viewed in appendix B.2.

10.2.2 WCHB Design

Figure 3.11 is the model inspiring the WCHB Verilog design. The differences include a reset signal to initiate the WCHB with its internal C-elements in each pipeline stage, and therefore a 2-input MUX is connected to the *ev_c*, *ev_d and t_e* outputs, and instead of an inverting C-element, two inverters are positioned before the *t_e* AND-gate on each input wire. To properly reset the WCHB, *t_e* from each pipeline stage must be "0" along with all outputs at reset. When reset is completed, *t_e* must be set to "1" from the first stage in each pipeline branch in order to propagate through all pipeline elements. This construction arise the need for a *"setup after reset"* timing condition, meaning that no transmission can begin in the pipeline before *t_e* has propagated through the longest interconnected path. All WCHBs in different switch blocks are constructed in this way, and this condition is of course enforced by all testbenches used in the design.

Each WCHB element is also bound by the 4-PDR handshake protocol described in section 3.4. Appendix A 16.3-2 shows a detailed block schematic with internal signals, and the Verilog code is presented in appendix B.2. Simulation results for a typical WCHB distribution is featured in appendix A 16.6-2.

10.2.3 Copy Element Design

A functional copy-element design is important in the ALFERN architecture. Inspired by the simple design presented in section 3.5.3, the element design is simple and intuitive with no reset or handshake protocol implemented. Each copy-element is configured by 4 bits, to indicate if the *t_e* signal is controlled by one connected receiver, or two connected receivers through a C-element. For the ALFERN design, each copy element can only communicate with 8 receivers with its 4 inputs. Important for correct circuit usage is the fact that two copy-elements cannot be interconnected because they lack 4-PDR handshake circuitry. Copy-elements employ a 4-PDR interface to act as an interface element between one sender and two receivers, making sure that the issued output is confirmed by each receiver. This role is in perfect accord with figure 3.14. A detailed block schematic with internal signal names as used in the Verilog model can be depicted in appendix A 16.3-4, and the Verilog code in appendix B.2. Figure 10.11 gives an example of simulated behavior.

10.2.4 Asynchronous LUT Design

Because of its custom design nature and complex internal circuitry, the asynchronous LUT was the most challenging circuit element to design. To illustrate asynchronous LUT computation and maintain simplicity, the LUT is designed as a 3-LUT featuring 8 bits to program the functionality.
Using the LUT architecture from figure 9.8, the design is almost identical with the exception of some extra signals for synchronization purposes. Implemented as a part of a TCCFB, the LUT interacts with the environment through the TCCFB. The dynamic behavior of asynchronous circuits is very dominant in the LUT, where all signal transitions are controlled by other asynchronous signals. A corresponding Verilog model can be studied in appendix B.2, and simulated behavior is shown in figures figure 10.7 and figure 10.10.

10.2.5 Switch Block Design

As the most complex of the low-level architectural elements in the AESRN, some features of the custom made bi-directional switch block should be explained. The essential element is the custom designed bi-directional switch point illustrated in appendix A 16.2-4. Since eight event channels are supported by each switch block, eight switch points providing routing flexibility constitutes the whole block. Each switch point consists of a WCHB element interconnecting with the input and output wires of the switch block according to the configuration bits. One-hot encoding is used to configure the connections between input and output channels to maintain an intuitive programming interface.
The code constructing a switch block is found in appendix B.2, and simulation results can be seen on all figures including AGERN simulation results among them figure 10.5. An architectural model with internal signals is included in appendix A 16.3-3.

10.3 Simulation Results

In order to simulate the whole Verilog model of the AESRN as a complete distribution system, two test scenarios involving all the designed modules were constructed for test purposes. As an extension to the textual description, figure 10.2 and figure 10.3 will give additional details. On these figures only connections used in the scenario is shown to maintain simplicity. Configuration of all switch blocks and computation elements are done by reading the configuration from file, and program each configuration register via the testbench. For further instruction on how to program the developed test system the reader is referred to chapter 11.4. All simulations are done with Active HDL v. 7.2 student edition from Actel.

Scenario 1: The TCCFB LUT in Port/TCCFB C is programmed to make a logical AND operation on three incoming events. Two events are sent on the AGERN, where the event on Event Channel 0 is issued from Port A/B via switch 7 and the event on Event Channel 1 is issued from Port E via switch 3. The third event is distributed locally on the ALFERN from CE_{Low}, and received by the TCCFB. The answer from the LUT is sent on the AGERN to its destination in Port A/B. Figure 10.2 illustrates the scenario in more detail.
The blue wire represents the event issued from Port A/B, red wire the event from Port E via switch 3, yellow wire for the ALFERN distributed event and the green wire is the result of the LUTs computation.

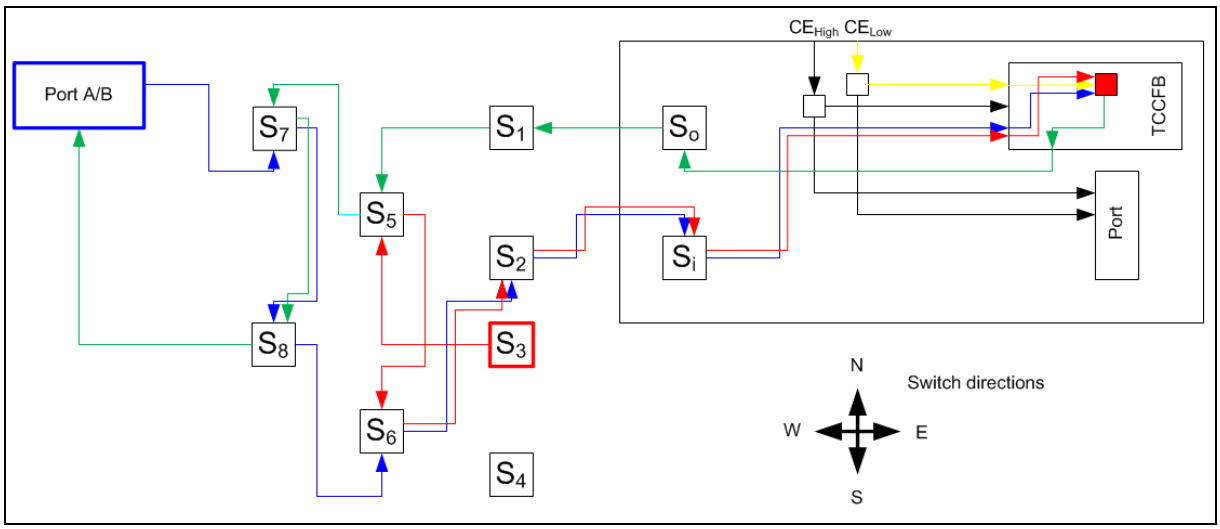


Figure 10.2: Event distribution scenario 1

Scenario 2: While scenario 1 focused on AGERN distribution, scenario 2 will only distribute events using the ALFERN. Two events are received on ALFERN Event channels 0 and 2 through the CE_{Low} copy element. The other event is distributed on ALFERN Event Channel 6 through CE_{High}. All events are received on the ALFERN inputs of the TCCFB. The LUT is programmed to do a logical AND of all inputs, and send the answer on Event Channel 2 to the CE_{Out} copy element. The forking from CE_{Out} will produce outputs on the ALFERN output event channels 2 and 6. Red, yellow and blue wires indicate the incoming events, while the green wire indicates LUT computation result.

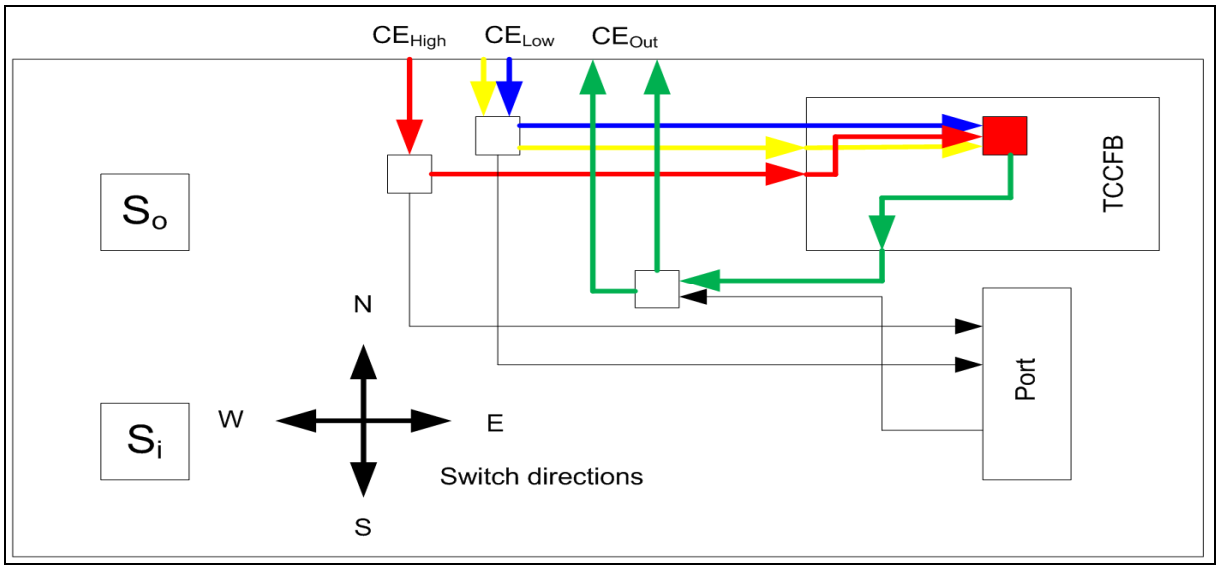


Figure 10.3: Event distribution scenario 2

10.3.1 AGERN Simulation Results

Scenario 1:

Part 1: Transmitting the events to the Port / TCCFB (Figure 10.4)

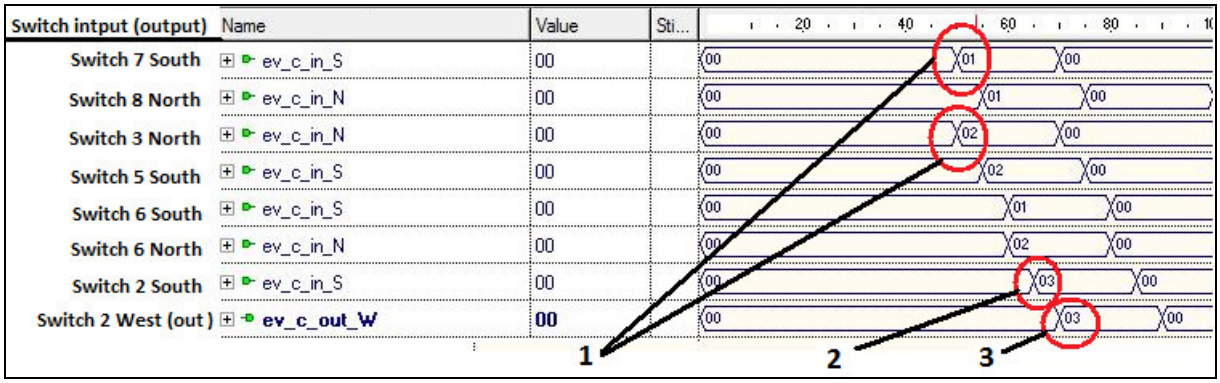


Figure 10.4: AGERN event flow for transmission of events to Port /TCCFB in scenario 1

- **Point 1:** The event on Event Channel 0 is issued from Port A/B, and received on the southern input of switch 7. At the same time the second AGERN event is issued from Port/ TCCFB E and received on the northern input of switch 3.
- **Point 2:** Both events have propagated to its desired output destination in switch 2 for distribution to Port/TCCFB C, and are received on the southern input. The value "3" on switch 2 input line in figure 10.4 show that both Event channel 0 and 1 are received.
- **Point 3:** After a simulated logic delay of 5ns in the switch, the events are sent to Port / TCCFB C. Also notice that after each event, a {00}-spacer is issued to reset the pipeline stage. This is according to the 4-PDR protocol.

Part 2: Handshake between switches (Figure 10.5)

This part of the event transaction is rather overwhelming in terms of signal amount. An example is therefore illustrated using AGERN switches 6, 2 and 5. The signaling procedure is according to the 4-PDR protocol, and is similar for all elements involved in a transaction. Appendix 16.6 gives additional examples of handshakes, and provides greater details.

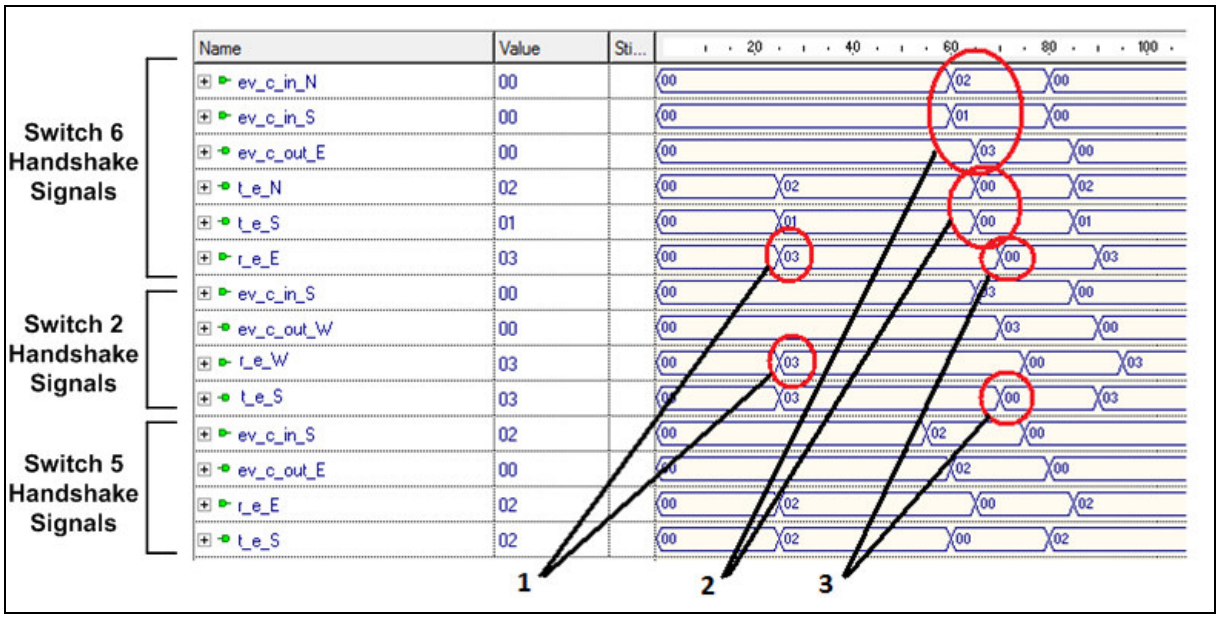


Figure 10.5: Selected AGERN switches performing handshak

- **Point 1:** Relationship between *r_e* and *t_e* signals. At reset switch 6 receives a high enable on Event Channel 0 and 1 from switch 2 on *r_e_E*. This means that events can be sent to switch 2 on these channels. *T_e_S* from switch 2 signals which southern inputs switch 6 can send to.
- **Point 2:** Events received on switch 6 is sent to switch 2 on Event Channel 0 and 1 on the eastern output after a simulated logic delay. The *t_e* signals for the corresponding input channels are set to "0" to signal that data is received. This is illustrated by *t_e_N* and *t_e_S*.
- **Point 3:** When switch 2 receives the events from switch 6 on its southern input *ev_c_in_S*, *t_e_S* is set low for those channels after a delay. This response from switch 2 is received on switch 6 on its corresponding *r_e_E* channels.

Distributions scenario 2 gives no additional information considering AGERN behavior, and is therefore omitted.

10.3.2 ALFERN Simulation Results

Scenario 1:
Part 1: Receiving events from AGERN and CE_{Low} (Figure 10.6)

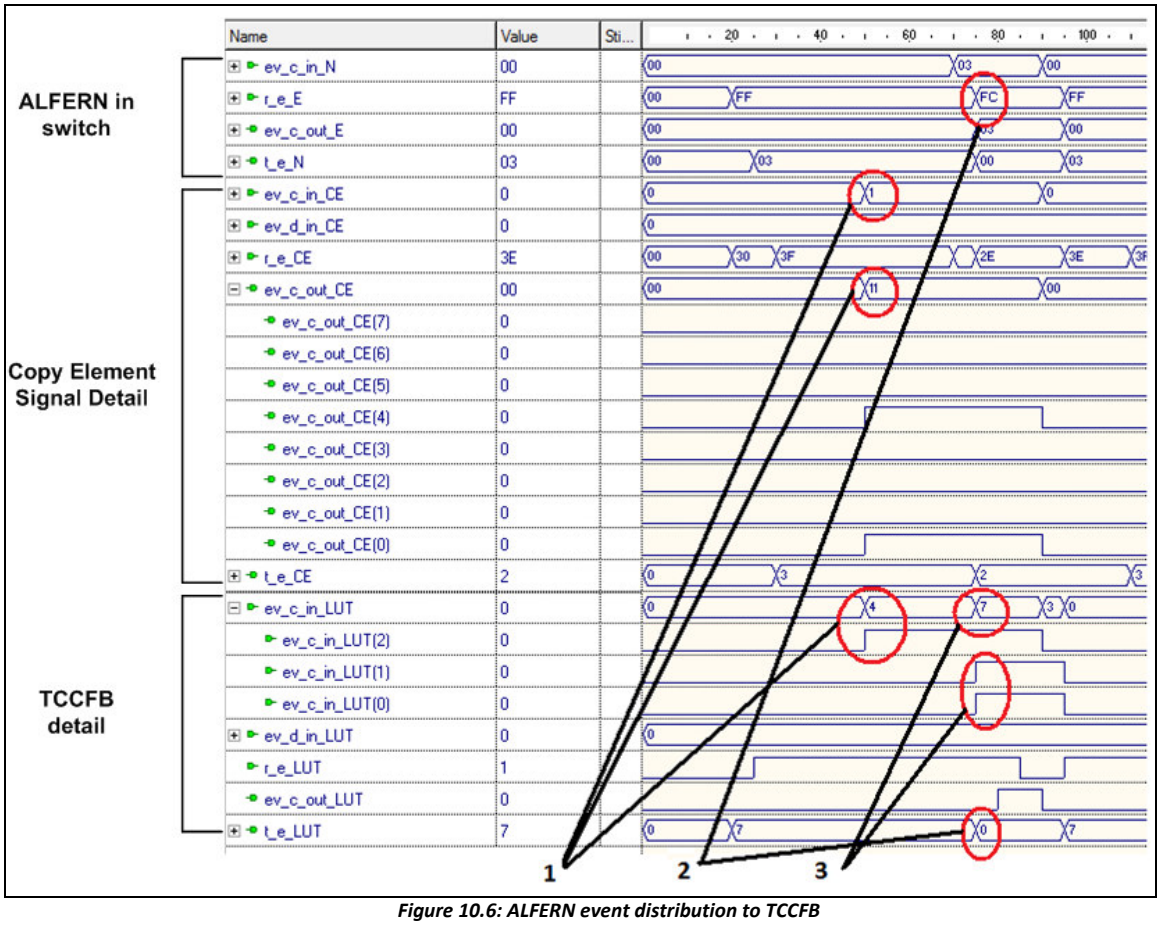


Figure 10.6: ALFERN event distribution to TCCFB

- **Point 1:** Since the ALFERN distributed event is issued concurrently with the AGERN event, and travels directly to the CE_{Low}, it is received first on Event Channel 2 by *ev_c_in_LUT*. Also notice the forking aspect on CE channel 0 and 4, where CE channel 0 is connected to the receiving TCCFB ALFERN inputs.
- **Point 2:** The two events distributed on the AGERN arrive, causing the TCCFBs internal LUT to start processing. *T_e_LUT* is set low to indicate reception and the AGERN in switch receives this indication on *r_e_E*.
- **Point 3:** The LUT inputs on *ev_c_in_LUT* indicate "111", meaning that the LUT computes the input combination to a logical "1" output. This is indicated on *ev_c_out_LUT*. Notice that even if the ALFERN input from CE_{Low} has lingered on the input a long time before the AGERN events arrived, it was not a valid condition for the LUT to start computing.

Part 2: LUT Operation (Figure 10.7)

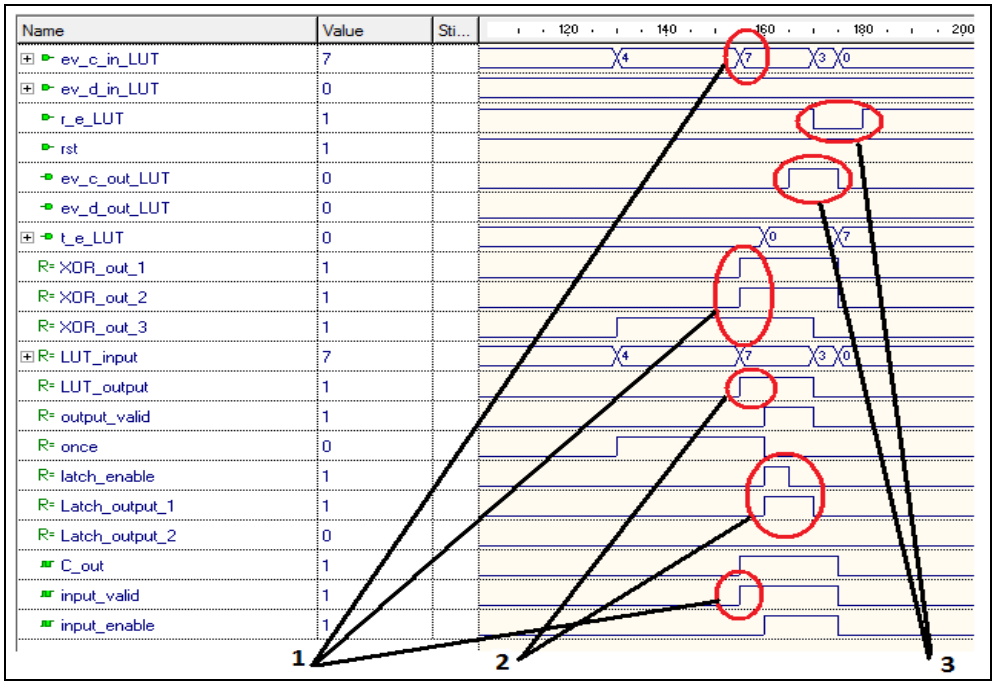


Figure 10.7: LUT operation on incoming events

- **Point 1:** When all inputs are received on *ev_c_in_LUT*, the XOR outputs will go high if the input combination is a valid "10" or "01" encoding for *ev_c* and *ev_d* inputs. *Input_valid* is set high to validate the current input combination. When only the ALFERN was received the other inputs where "0", indicating a non-valid input.
- **Point 2:** Since the input is valid, the *LUT_output* will be "1" according to the AND functionality implemented in this scenario. *Input_enable* goes high to show that the LUT is processing, trigging the *latch_enable* signal such that the *LUT_output* can propagate to the *latch_output*. Notice *that input_enable* is inverted when sent out of the LUT, to follow the signal protocol used in all other elements.
- **Point 3:** The latch output is routed to *ev_c_out_LUT*, and all outputs are reset when the receiving component confirms the reception by setting it *t_e* signal low, also setting *r_e_LUT* low. The transaction is complete when *r_e_LUT* goes high.

Part 3: Transmitting the LUT output (Figure 10.8)

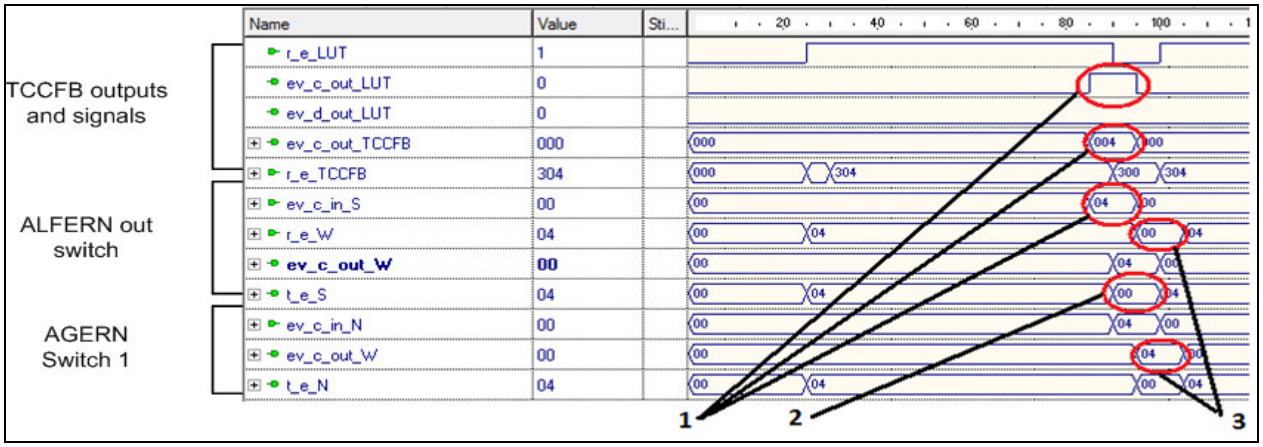


Figure 10.8: Handling the LUT output on the ALFERN

- **Point 1:** The LUT output from *ev_c_out_LUT* is first transmitted through the TCCFB containing the LUT without adding delay, and received on the southern input *ev_c_in_S* of the switch connecting to the AGERN.
- **Point 2:** To acknowledge the received LUT result, *t_e_S* is set low from the output switch. This causes *r_e_LUT* to go low, resetting LUT signals internally.
- **Point 3:** When the ALFERN output switch has sent the result to AGERN switch 1 for further transmission on the AGERN, *r_e_W* goes low for the output switch until AGERN switch 1 has transmitted the result to the next pipeline stage. This completes the transactions concerning ALFERN circuit elements. Since the AGERN distribution with the result has the same distribution pattern as in figure 10.4, no further details are given.

Scenario 2:
Part 1: Receiving inputs from ALFERN input Event channels (Figure 10.9)

- **Point 1:** Input is received on CE_{Low} on Event channels 0 and 2. Because both CEs interact with the testbench, the *t_e_CE* signal is set low 20ns after the received input. This delay is equivalent to one clock period. Input and handshake conditions are the same for both CE_{Low} and CE_{High}.
- **Point 2:** Since no logic delay is added for the CE outputs, they are received at the *ev_c_in_TCCFB* instantly after being issued from each CE. Note that the value transmitted from each CE is correctly received.
- **Point 3:** Detail of TCCFB handshake towards input CEs. Note that when the inputs are received, the *t_e_TCCFB* signals set low corresponds with the receiving input channels. TCCFB input channels [15:8] are reserved for ALFERN inputs.

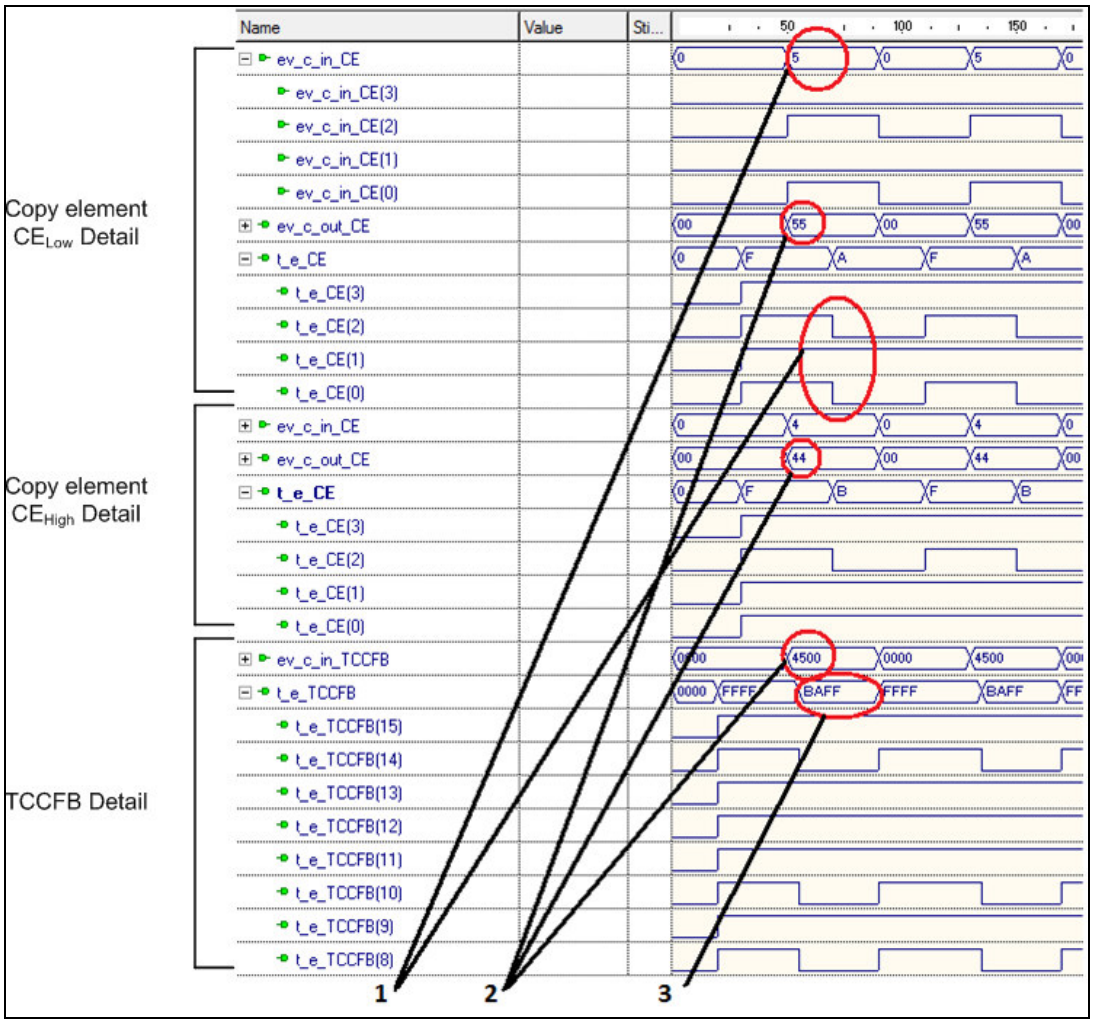


Figure 10.9: ALFERN event reception via copy elements

Part 2: LUT Computation (Figure 10.10)

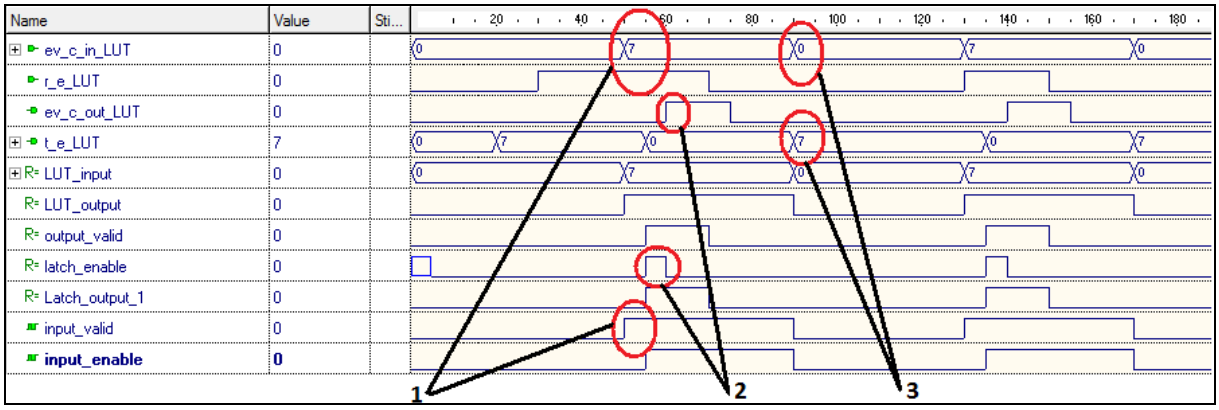


Figure 10.10: LUT computation

- **Point 1:** In contradiction to figure 10.7 where the LUT inputs are received at different time instances, all inputs are received simultaneously in scenario 2. *Input_valid* goes high immediately after the inputs are received. *T_e_LUT* is set low after a delay.
- **Point 2:** *Latch_enable* is set high to latch out the LUT outputs , which propagates to the output after a logic delay.
- **Point 3:** When the transmitting CEs send the "0"-spacer to reset the transmission, *t_e_LUT* is set high to signal that the LUT is ready to accept more inputs.

Part 3: Transmitting LUT result on the ALFERN (Figure 10.11)

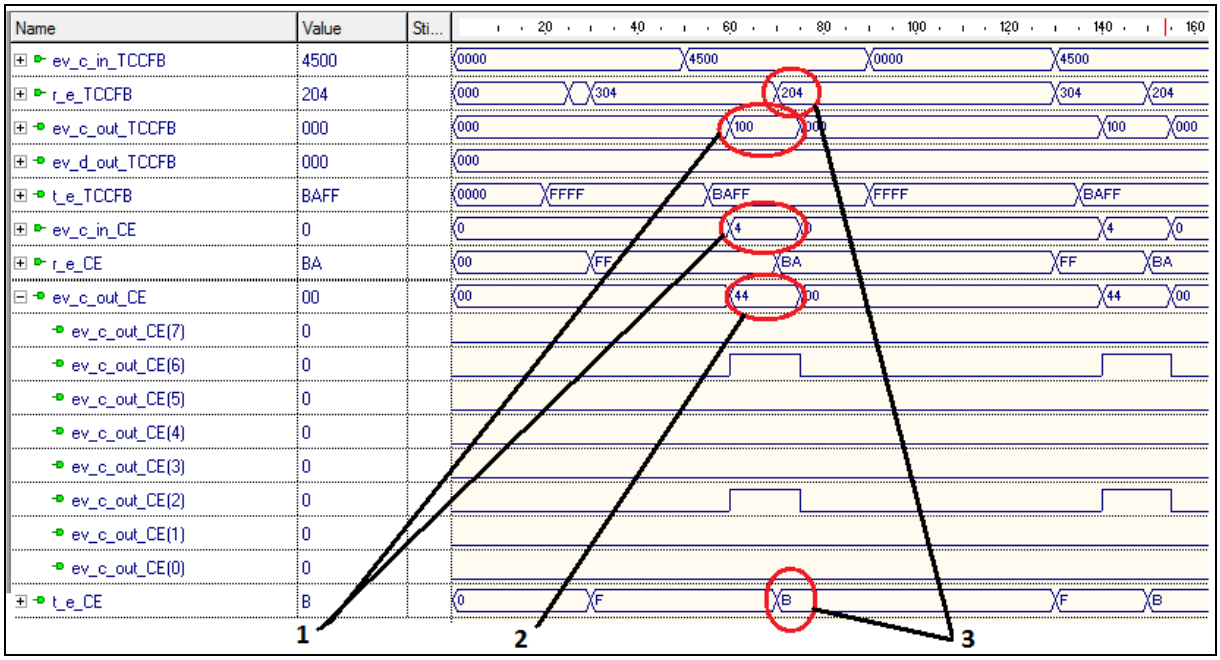


Figure 10.11: Interaction between TCCFB and Copy element

- **Point 1:** TCCFB output on *ev_c_out_TCCFB* is received on *ev_c_in_CE* on channel 2.
- **Point 2:** Forking of the input to ALFERN output Event channels 2 and 6.
- **Point 3:** When the CE sets its *t_e_CE* for the corresponding outputs low, this is received by *r_e_TCCFB* to finish the handshake.

10.4 Programming the AESRN

10.4.1 Description of a Plausible Event Distribution Scenario

This example is constructed in order to fully utilize the routing potential of the AGERN, and get some notion of the AGERNs capacity and flexibility during a realistic event distribution scenario. Table 10-1 describes the active event channels from each of the peripherals involved in the transactions. Since the numbering [7:0] is used for an Event Channel, the same system will be adapted to table 10-1. The Event Channel color is used to identify the path of each event according to figure 10.12 and figure 10.13.

Peripheral [Source]	Event Channel	Event Description	Channel Color	Peripheral [Destination]
Port / TC C	0	IO pin used as clock input		Port / TC E
Port / TC C	1	Generated by TC overflow		Port / TC E
Port / TC D	2	CC Channel A		Port / TC F
Port / TC D	3	CC Channel B		Port / TC F
Port / TC E	4	Trigger ADC conversion by counter		Global (ADC1)
Port / TC E	5	Trigger DAC conversion by counter		Global(DAC)
Port / TC F	6	TC overflow		Global (DMA)
Port / TC F	7	Trigger conversion by TC CC chan A.		Global (ADC2)
Global (DAC)	4	Conversion Complete		Port / TC D
Global (ADC)	5	Conversion Complete		Port / TC D
Port A	1	Pin Event		Port B
Port B	2	Pin Event		Port A

Table 10-1: Description of multiple events routed together on the AGERN

The switch block layout in appendix A 16.2-4 represents the cross switch used in the AGERN with 16 input channels and 16 output channels. In the current implementation a switch block is not programmable in the sense that Event Channel 1 as input can connect to Event Channel 4 and use this as output. The firm consistency of the switch block routes input channel 1 from a chosen input direction to output channel 1 in a chosen direction.
The situation described in table 10-1 represents the maximum capacity of the AGERN in the output routing direction, because each event passes through switch 5 using all its input and output channels. Figure 10.12 and figure 10.13 illustrate this further. Of the global input channels half of the capacity is used, but due to routing transactions including switch 6 much of the internal capacity is already used. This can also be depicted in figure 10.13 (right).

Although table 10-1 does not present the smartest scheme when it comes to using the full capacity in all global directions, twelve events are routed simultaneously from a variety of sources to different destinations. The original event system presented in [6] had a maximum capacity of eight events simultaneously, which gives the AGERN a capacity increase of four events compared to the original Event System for this scenario. With full utilization this increase numbers eight events to global peripherals, but would require a different routing scheme. Figure 10.12 illustrates possible connections between the switches.
Ordinary wire connections not in use to distribute events according to table 10-1 are black, while the colored wires represent wires taking part in an event transaction.

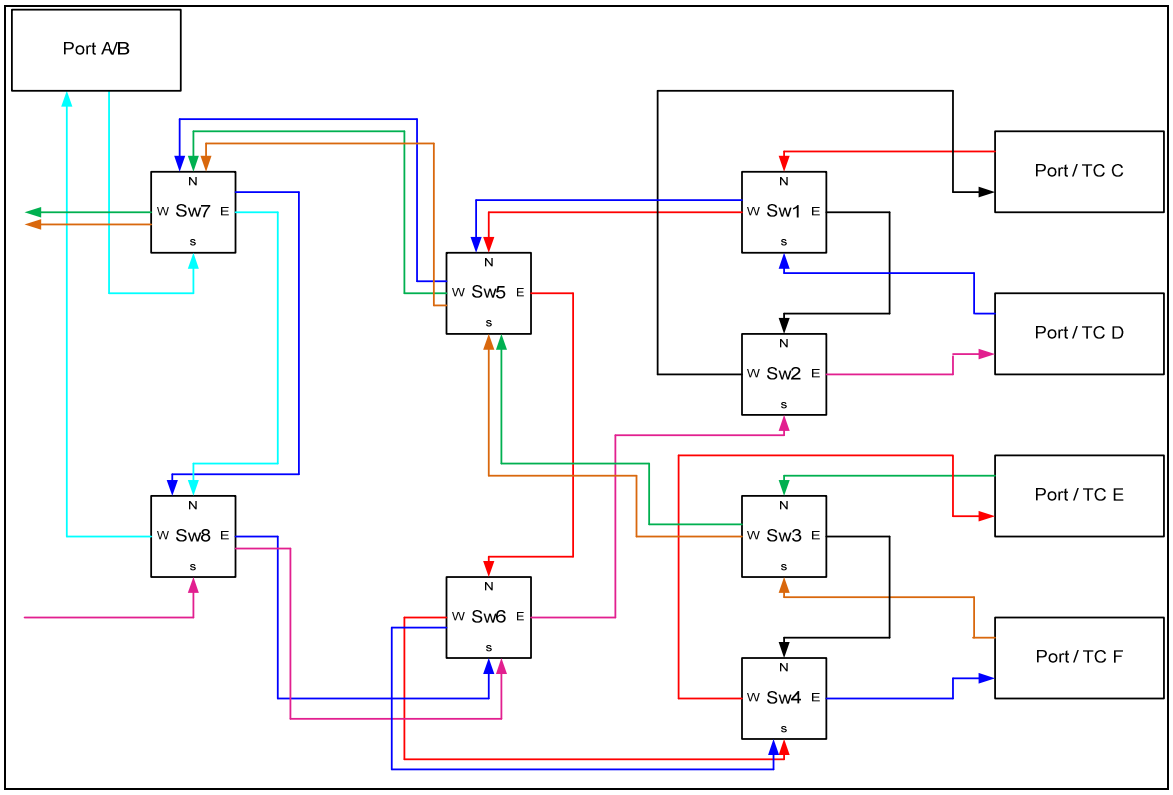


Figure 10.12: Graphical representation of event scenarios

Connected with figure 10.13 (left) it will be possible to follow the route of each event from source to destination. The peripheral on the leftmost side marks the source of the event, and the peripheral on the rightmost side marks the destination. Associated to table 10-1 and figure 10.12 each colored wire represent **two** event channels, since two neighboring channels always have the same source- and destination peripheral in this example. The reason is to make it easier to translate the figures, while still maintaining the principle of programming the AGERN for arbitrary event distribution.

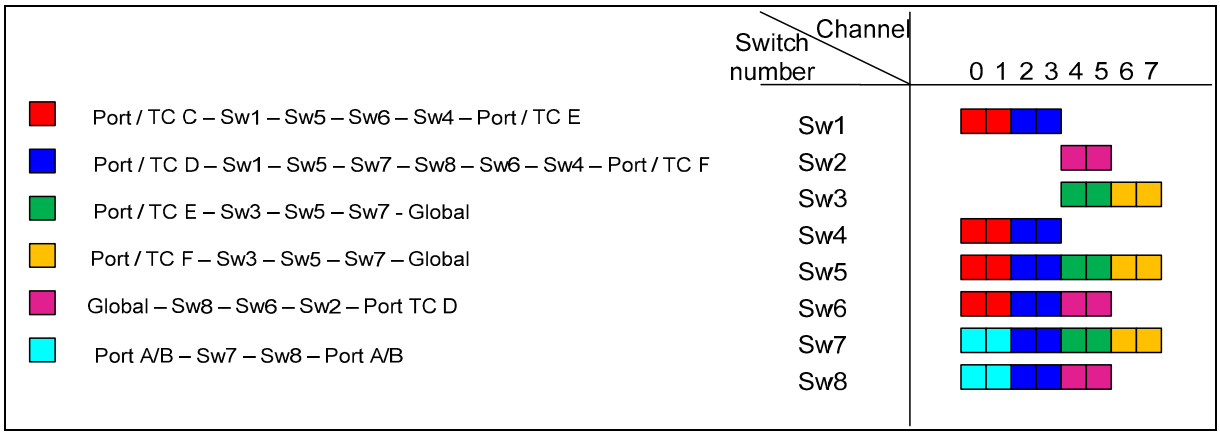


Figure 10.13: Event distribution pattern (left), Allocated switch channels (right)

To further illustrate the allocation of different event channels for each switch, figure 10.13 (right) shows which wires are allocated for which event for each of the involved switches.
Since each input connects to an output with the same number, the table is valid for both input and output allocation for each switch. As figure 10.13 (right) depicts, Sw5 and Sw7 are using all of their input and output channels and will each become the limiting factor of capacity if further events should be issued in their direction. Note that not all internal connectivity is used, for instance on switches 2 and 3.

10.4.2 Switch Block Configuration

Using the constructed event distribution scenario from table 10-1, the AGERN must now be programmed in order to connect the switch blocks correctly. Remembering that each switch block consists of eight switch points after the manner presented in appendix A 16.2-4 four bits are used to program each switch point. Because memory cells are used to control pass transistors for data- and handshake signals, one-hot encoding is used when connecting inputs to the desired output. With the applied encoding scheme 32 bits are needed to program one switch block. To illustrate the programming procedure, switch 5 from figure 10.12 can be used as an example. A more detailed image of switch 5 can be seen in Figure 10.14 while the encoding scheme for each switch block is presented in table 10-2.

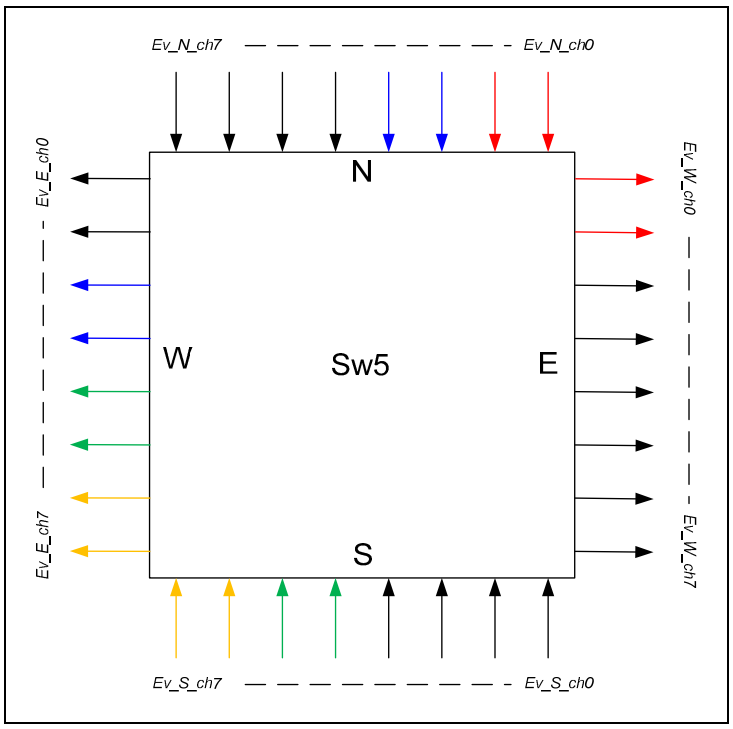


Figure 10.14: Detailed view of the events routed by switch block 5

Source	Destination	Encoding sequence
Ev_N_ch$_X$	Ev_W_ch$_X$	0001
Ev_N_ch$_X$	Ev_E_ch$_X$	1000
Ev_S_ch$_X$	Ev_W_ch$_X$	0010
Ev_S_ch$_X$	Ev_E_ch$_X$	0100

Table 10-2: Encoding sequence of 4 bits for each switch point within the switch block

The programming bit sequence of 32 bits is divided into eight 4 bit blocks, where the Event Channel notated 0 holds bit positions [3:0] and Event Channel 7 holds positions [31: 28]. Using figure 10.14 as the illustrating example, the two red wires representing event channels Ev_N_ch0 and Ev_N_ch1 should be routed towards Ev_E_ch0 and Ev_E_ch1. Using table 11.2, this means that both switch points must be programmed with the sequence “1000” for programming bit positions [3:0] and [7:4]. If the same method is used to program the other switch points as well, the programming bit sequence for switch 5 will be similar to table 10-3. Note that this encoding does not represent the bit pattern required by the architectural model in appendix

A 16.2-4, which would require a “1” bit to control both input and output event channels. The applied one-hot encoding is used for simplicity, and does not introduce any functionality restrictions towards the actual model.

Programming bits	[31:28]	[27:24]	[23:20]	[19:16]	[15:12]	[11:8]	[7:4]	[3:0]
Bit sequence	0 0 1 0	0 0 1 0	0 0 1 0	0 0 1 0	0 0 0 1	0 0 0 1	1 0 0 0	1 0 0 0
Source channel	Ev_S_ch7	Ev_S_ch6	Ev_S_ch5	Ev_S_ch4	Ev_W_ch3	Ev_N_ch2	Ev_N_ch1	Ev_N_ch0
Destination channel	Ev_W_ch7	Ev_W_ch6	Ev_W_ch5	Ev_W_ch4	Ev_W_ch3	Ev_W_ch2	Ev_E_ch1	Ev_E_ch0

Table 10-3: Complete programming sequence for the configuration of switch 5

It is important to emphasize that only routing channels pairs in the same direction is used to simplify the example, and provide a complete and manageable overview. Single events could be routed on any channel in any direction if it was necessary.

To fully utilize the AGERN capacity, an example distributing 24 events on the AGERN is included in appendix 16.7 with illustrating figures and the configuration bit table. In this example all 16 GLOBAL wires are used, along with 8 events distributed between neighboring Port-peripherals on the AGERN.

10.5 Synthesis of the AESRN

To verify the model designed for an AESRN implementation, a synthesized version aimed at an implementation for the Xilinx Spartan 3 XC3S1000 was constructed with Synplify Pro 8.9 from Synplicity. The Spartan FPGA-platform was chosen because it is used as FPGA on the Suzaku S-platform by Atmark Techno [54]. With an embedded µBlaze soft-processor running a µClinux Linux core and possibilities for direct register operation for test programs through the OPB (On Board Peripheral)-Bus, this is a powerful device for prototype testing.

Since the synthesis results for the target FPGA platform does not in any way mirror a realistic microcontroller implementation size, this chapter can be considered for orientation only. The synthesis results for all constructed modules can be viewed in appendix 16.5 and some comments on synthesis problems and issues will be featured in this section.

The most interesting thing about synthesizing asynchronous logic with stateholding capabilities is the error messages issued as synthesis warnings from Synplify Pro. With a loop connecting the output of a C-element to the inputs, a combinational loop is constructed. A combinational loop is not wanted in synchronous designs, and so a large amount of warning messages appear for all designs involving C-elements. These messages can altogether be ignored, even though they possibly represent a problem when it comes to physical FPGA implementation. Remembering the theory from sections 3.3 and 6.2.2, most synchronous synthesis tools do not support delay measurement functions for asynchronous models. Practically this means that each combinational loop that is constructed on the FPGA have an unknown delay, which makes it impossible to guarantee a correct behavior. Also remembering the QDI- delay model assumptions relying on balanced isochrnoic forks, there is no way to be sure that such delays have been balance by the synthesis tool.
These delay uncertainties mean that even though all modules are synthesizable, their physical behavior on the FPGA cannot be accounted for when delays in critical wire sections remain unknown.

Commenting on the actual synthesis results, it is notable that the seemingly large design is very densely implemented on the Spartan FPGA. Using 8% of the available LUT area when the whole AESRN example explained in section 10.1 it cannot be categorized as an area consuming design.

Section IV –

Performance Analysis, Results and Conclusions

11 Performance Analysis of Proposed Solutions

This chapter presents estimated performance of each Event System solution in terms of operating speed. Special focus is concentrated on analyzing the Asynchronous Event Systems performance. Being an asynchronous routing facility, the AESRN provides a challenge considering analysis of the system performance. By applying experimental analysis techniques to the AESRN pipeline it is possible to derive some measurement on the pipeline's performance. Exploiting the fact that the AESRN pipeline includes no computational elements, as is the case for many micropipeline constructions, measurements are simplified. HERN and CERN solutions are analyzed using datasheets and expected performance factors.

11.1 Analyzing HERN performance

The HERN performance is solely dependent on how the I/O-processor is implemented. With a simpler architecture than the AVR® CPU it should be possible to run at a higher clock rate than the CPU, and still maintain reasonable power consumption. Expected performance is below the 128MHz maximum frequency for the implemented Event System, but double the AVR® frequency, 64MHz, is expected with careful design.

11.2 Analyzing CERN performance

The CPLD Event Routing Network rely on architectures developed by Atmel, and so performance analysis have been made of the CPLD architecture providing basis for CERN's computational MCB elements. Estimated operation speed for the ATF1508RE CPLD is 333MHz running at 3.3V [7]. Using the peripheral clock of the XMEGA A1 [6] as reference, which can operate at 2-4 times the AVR® CPU speed of 32MHz, a peak performance of 128 MHz at operating voltage 1.6V is expected.
This means operating both the CIOBus and GEB at 128 MHz, making sure that an event is received within one peripheral clock cycle. Although Atmel's CPLD structures can tolerate higher velocities, power consumption is an issue reducing the operating voltages and hence operating speed.
Expected performance for the CERN solution is therefore the same as the peripheral clock speed.

11.3 Analyzing AESRN performance

11.3.1 Formal Equations for AESRN Pipeline

The methods describing the analysis flow is covered in background theory section 4.1, and described in [36] by Yahya et Al. figure 11.1 illustrates the difference between the computational pipeline used in [36], and the token buffer pipeline used in the AESRN.

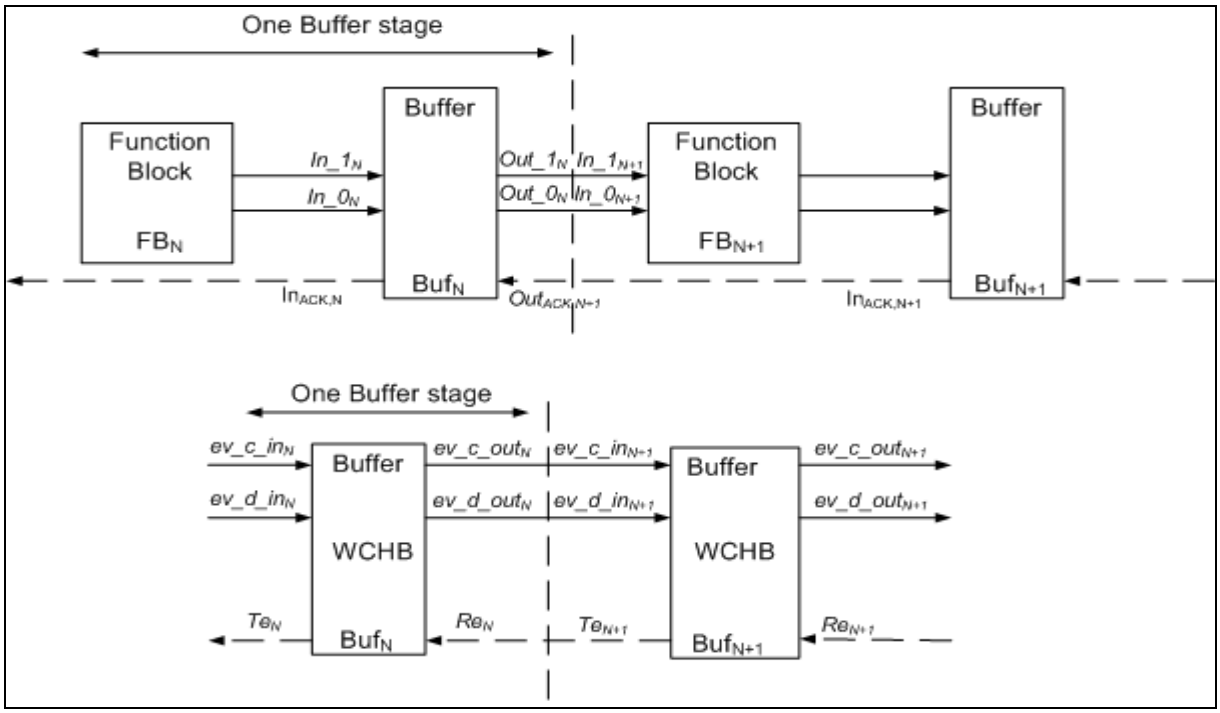


Figure 11.1: Conventional Computational pipeline (top) [35], AESRN token buffer pipeline (bottom)

The token buffer property omits the functional blocks used for computation, removing the variable functional block delay from the performance equation. Since it is the functional blocks in each pipeline stage which introduces the most crucial variable delay, the pipeline equations for analysis is simplified. Using dependency graphs to derive the pipeline equations, the AESRN token buffer pipeline can be modeled like in figure 11.2. The graph communicates the same behavior as the 4-PDR protocol, but makes behavioral analysis simpler.

Some transitions in an asynchronous pipeline can trigger concurrently, making it hard to analyze the correct behavior. As explained in section 4.1, there is always a main synchronization event to control behavior of proceeding events. For a WCHB element like the one on figure 3.11, this synchronization event is issued from each C-element as the transition C↑ and C↓ in figure 11.2. Using the proposed method from [36] and section 4.1 to analyze the unfolded dependency graph, and considering C↑ as the main synchronizer event, the following equations describe the Total Cycle Time (TCT) of a WCHB-element at stage N in the AESRN pipeline.

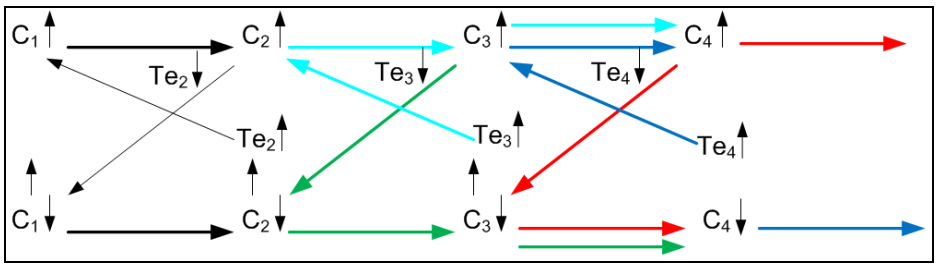


Figure 11.2: Unfolded dependecy graph for AESRN pipeline

Considering the main synchronizer event C↑, the transition delay is MAX [(C↑,T_e↓,C↓),(T_e↓,C↓,C↓)] for the transition C↑ to C↓. Transition delay back from C↓ to C↑ is given by MAX [(C↓,T_e↑,C↑),(T_e↑,C↑,C↑)]. Presented with a more specific delay notation considering both preceding and proceeding buffer stages, TCT for a random buffer stage N can be derived:

$$TCTn_{WCHB} = MAX[(T_{C(n+1)\uparrow}+T_{Te(n+1)\downarrow}+T_{C(n)\downarrow}), (T_{Te(n)\downarrow}+T_{C(n-1)\downarrow}+T_{C(n)\downarrow})] + MAX[(T_{C(n+1)\downarrow}+T_{Te(n+1)\uparrow}+T_{C(n)\uparrow}), (T_{Te(n)\uparrow}+T_{C(n-1)\uparrow}+T_{C(n)\uparrow})]. \quad \text{Eq. (10.1)}$$

The colors in eq. 10.1 represent the delay pattern described in figure 11.2, with stage 3 as $C_{(N)}/_{T(N)}$. Equation 10.1 is consistent with equation 4.1, but modified to only consider token buffer pipelines without computational elements. Notice that both the *eval* and *reset* token is included for TCT measurement in eq. 10.1. Since the AESRN, including the ALFERN or ALERN solutions, does not contain any split or merge of handshake channels, the linear equations are applicable for the whole pipeline structure.

In order to obtain appropriate delay measurements for each pipeline stage, values obtained through scientific research and presented in literature was investigated. As pointed out in [43] [44] the delay factors for each process technology are many, leading to large statistic variance.
Considering 90nm process technology the delay for a NAND gate is used as basis for estimated delay computations. Since an AND gate is used in the WCHB, NAND gate delay and inverter gate delay will together figure as an AND delay estimator. The largest value for separation distance and load capacitance derived from the measurements in [43] are used to obtain "worst-case" values for NAND gate delay. The presented conditions for the obtained measurements are close to the AESRN conditions, with respect to fan-out of NAND gates and hence the load capacitance for each gate. The selected NAND-delay is for VDD = 0.8V in the 90nm process. Delays for C-elements are approximately derived from these numbers.

11.3.2 AESRN Pipeline Performance

To calculate NAND and inverter delays for process technologies not specifically presented in papers, an approximation from Intel's research in [49] declaring an estimated reduction factor of 0.7, or 30%, in decrease of gate delay from one process generation to the next. The same factor is used for calculations in [47].Conducting a simple test to see how relevant this approximation is, the delay numbers for an inverter gate measured in different process technologies presented in [48] are used. Results of the comparisons can be viewed in table 11-1. The difference factor is the reduction in delay compared to the preceding technology.

Process techonology	Inverter delay [48]	Difference factor
600nm	165ps	-
350nm	95ps	0.58
250nm	67ps	0.71
180nm	47ps	0.70
130nm	33ps	0.71
90nm	22ps	0.71
65nm	15ps	0.71

Table 11-1: Scaling of inverter delays in different technologies

Since the delay factor follows the approximation from [49], it will be used to derive approximate performance parameters for the AESRN using different process technologies. Because the delay numbers derived for the NAND-gate in [43] represents conditions most similar to the AESRN, this delay number is used and scaled to fit preceding technologies. Table 11-2 gives an overview of the

measured results. Also bear in mind that a scaling factor of 0.7 represents the worst-case value from table 11-1.

Delay contributor	Delay 90nm	Delay 130nm	Delay 180nm	Delay 250nm	Delay 350nm	Delay 600nm
τ_{NAND}	50ps [43]	70ps	100ps	142ps	202ps	290ps
τ_{INV} [48]	20ps	33ps	47ps	67ps	95ps	165ps
$\tau_{AND} = \tau_{NAND} + \tau_{INV}$	70ps	103ps	147ps	209ps	297ps	455ps
$\tau_{C\text{-}el}$	100ps	140ps	200ps	284ps	404ps	580ps
$\tau_{Te} = \tau_{AND} + \tau_{INV}$	90ps	136ps	194ps	276ps	392ps	620ps
$\tau_{C} = \tau_{C\text{-}el}$	100ps	140ps	200ps	284ps	404ps	580ps
$TCTn_{WCHB}$	600ps	832ps	1.19ns	1.68ns	2,4ns	3.56ns
Data token frequency	1.66 GHz	1.2 GHz	840 MHz	595 MHz	417 MHz	281 MHz
Event delay $2xTCT_{nWCHB}$	1.2ns	1.66ns	2.38ns	3.24ns	4.8ns	7.12ns
Event frequency	833MHz	600MHz	420MHz	247MHz	208.5MHz	140.5MHz

Table 11-2: Measured TCT and frequency for AESRN pipeline

Using these approximated values inserted into Eq. 10.1, selected TCTs for a WCHB at stage *n* in the pipeline is:

$TCTn_{WCHB}$ = MAX[300ps,300ps]+MAX[300ps,300ps] = 600ps. [90nm]
$TCTn_{WCHB}$ = MAX[1780ps,1780ps]+MAX[1780ps,1780ps] = 3.56ns. [600nm]

Both parts of the equation will give the same value, since no logical computation is included to provide different delays for each stage, and all other delays are equal for each stage.
This gives an estimated frequency of 1.66 GHz for tokens travelling in the pipeline for a 90nm process, and 281 MHz for a 600nm process. For results concerning other process technologies, see table 11-2.

As may be pointed out there exist LUTs as a part of the total Asynchronous Event System pipeline if TCCFBs are included. As long as LUTs reside inside the peripherals, they are not considered a part of the distribution pipeline, and an event issued from a LUT is considered a peripheral originated event rather than a computational originated pipeline event. This assumption makes the estimations from table 11-2 valid for the whole pipeline structure even with the presence of computational elements.

11.4 Comments on analysis and results

Since the AESRN pipeline is a fine grained pipeline with no logical computation between each pipeline element, the only delays consist of gate- and wire delay enabling tokens to propagate fast through the pipeline.

Compared to the fine grained high-speed pipelines in [38] which can reach a TCT of 1.18ns using a 600nm CMOS process, and include functional blocks for computations as well, the assumption for the AESRN could prove valid. Also keep in mind that the calculations in section 11.3.2 used worst-case delay assumptions from [43]. The asynchronous pipelined FPGA designed by Teifel et Al. [1] include computational logic blocks with LUTs, and reaches a peak speed of 400MHz using a 250nm CMOS process. A refined architecture using an 180nm CMOS process is presented in [32], reaching a peak speed of 700MHz.
Considering these circuit speeds, and removed computational logic, the AESRN pipeline should reach much higher distribution speed. All the other pipelines considered from literature are also using a dual-rail computation and transfer interface.

Acknowledging the fact that all the variable delay parameters connected to aggressive processes presented in [43] [44] [45] must be taken into account, the computed values for the AESRN should be considered for orientation only. With no knowledge of how an implementation would look like on the silicon level, accurate measurements are hard to derive.

12 Event System Cost Factors and Tools overview

When designing a new system it is always important to evaluate the costs associated with the new system design. This chapter will evaluate the costs for the proposed Event System solutions, and compare them to the costs generated by the current Event System.
All solutions feature some degree of programmability not before encountered by the existing tool chain. Trying to visualize how programmability features can be integrated in the tool flow and show how the user easily can use the new features and interact with the GUI, the last section is dedicated to this purpose.

12.1 Introduction to AVR® XMEGA Size Factors

Important notice

Important library and process related numbers and data used for calculations in section 12.2 contains sensitive process information related to the Atmel 35k9 process [10], and is therefore included in a restricted appendix B.1. The numbers presented in appendix B.1 will form the basis for the calculations in section 12.2.1 - 12.2.3.
Some important size factors for area estimations are included in appendix table B-1. The tables in section 12.2.1- 12.2.3 only include NAND equivalent sizes for all modules, partitioned into logic area, additional logic and total area. Logic area is the area directly estimated from architectural drawings, without including interface logic, extra memory or routing area occupied by wires. Additional logic consists of FLASH memory, logic used for interface, synchronization or signaling purposes and extra routing logic because of the FPGA inspired topology. It will be specified for each of the Event System solutions how much extra logic is included for different purposes.

For detailed size information related to die-occupation in mm^2 and additional area information, the reader is referred to appendix B.1. Tables from table 12-2 - table 12-5 can be viewed with full area information in appendix tables B-2 – B-5.

12.2 Cost Overview and Performance Evaluation

Each of the proposed Event System solutions HERN, CERN and AESRN have their strengths and weaknesses compared to each other. This section evaluates the cost of each implementation with respect to performance in important areas. To compare each solution, table 12-1 presents a grading system were red means poor performance, yellow means average performance and green means good performance. This table will provide the basis for a more detailed discussion.

Event System solution / **Performance factor**	**HERN**	**CERN**	**AESRN**
Implementation Size			
Circuit Speed			
Power Consumption			
Programmability			
Computational power			
Ease of Use			

Table 12-1: Cost gradation of performance factors

12.2.1 HERN Performance Evaluation

Area Estimation

Referring to table 7-1 for a measured NAND gate count for different I/O-processor solutions and applying these gate counts to the approximated numbers presented in appendix B.1, a detailed area estimate can be derived. Table 12-2 estimates the total NAND equivalent area for different HERN solutions including additional logic.

Domain	NAND eq. logic	NAND eq. additional Logic	**Total NAND equivalents**
Simple I/O	2552	598	**3150**
Medium I/O	3852	798	**4650**
Advanced I/O	6352	1173	**7525**

Table 12-2: Estimated HERN area

For the HERN solution additional logic consist of 10% extra for routing area, 5% extra for interface related logic and 500 bit of FLASH memory to hold configuration bits for included switch blocks. All extra logic percentage is related to the NAND equivalent logic from table 7-1. More detailed estimation can be found in appendix table B-2. Estimated gate count for each I/O-processor alternative includes shared memory with the AVR®, and no extra area is added for this purpose.

Compared to the current Event System implementation medium a HERN solution is about the same size in terms of NAND equivalents. The increase for an advanced I/O-processor is 67%, while the simplest I/O-processor gives an area decrease of 43% with the proposed architecture.

Computation Speed

The limiting factor for performance is the amount of consumed energy associated with the I/O-processor. This is considered an application specific choice, depending on the functionality range wanted for the specific XMEGA. With good power-down design, a clock speed between 64 MHz and 128 MHz should be achievable depending on the simplicity of the processor, offering about the same computation speed as the current Event System with a maximum peripheral distribution speed at 128 MHz, or four times the AVR® CPU clock frequency [6].

Energy Consumption and Programmability

Including a separate I/O-processor will increase the power consumption compared to the Event System. Depending on the selected implementation of either a simple, medium or advanced processor alternative, the associated energy consumption will increase accordingly. With no accurate numbers to achieve a good estimation the probable increase factor remains an unknown variable. With the possibility of adding instruction sets for increased functionality at need, the I/O-processor offers good programmability features.

12.2.2 CERN Performance Evaluation

Area Estimation

Based on the estimations for logic area from appendix table B-1, a more complete CERN area model including logic area occupied by routing wires, additional interface logic and FLASH memory can be derived than the one presented in table 8-1. Table 12-3 summarizes the total area estimations for 1 and 2 CPLD MCBs included in each Port / TC peripheral.

# CPLD MCBs	NAND equivalents logic	NAND eq. additional logic	**Total NAND equivalents**
4	3564	1744	**5308**
8	6428	4087	**10515**

Table 12-3: CERN area estimations

With 2 CPLD MCBs in each Port / TC peripheral the amount of additional FLASH memory to store configuration bits have reached 1kB, which makes the CERN solution expensive considering memory usage. Area consumed by switches, memory cells and wires dominate the logic area, especially if compared to table 8-1. The area overhead related to routing and configuration bit resources are approximately 57% with 2 CPLD MCBs implemented in each Port/TC peripheral. This fact reveals a potential weakness with using PLA blocks as Event System computational blocks: Without careful planning of routing the memory usage grows to rapidly as more computational blocks are added. Additional logic includes 10% for routing because of increased bus-capacity, 5% for related interface logic and up to 1kB of extra FLASH memory. A full overview is given in appendix table B-3.

Computation Speed

Using PLA blocks with predictable delay, deterministic event distribution should be possible. The original ATF1508RE architecture operates with 5ns pin-to-pin propagation delay, a delay that is more likely to decrease than increase with the simpler CERN architecture. 5ns delay gives an estimated computation speed of 200MHz, which is about 1.5 times faster than the maximum speed of 128MHz offered by the XMEGA peripheral clock. Expecting a maximal event propagation delay of 2 AVR® CPU clock cycles of 32MHZ [6], this demand is easily met by the CERN.

Energy Consumption and Programmability

As a direct result of the large memory requirement depicted in table 8-2, CERN programmability is excellent and makes it possible to compute almost any product term combination on incoming events. Energy consumption would increase compared to the current Event System because of larger area occupation and more memory cells.

12.2.3 AESRN Performance Evaluation

Area Estimation

The implementation cost associated with the AESRN in terms of equivalent NAND gates can be viewed in table 9-2, table 9-3 and table 9-6.
Approximated numbers from appendix B.1 are used for detailed area estimations. If only the routing facilities of the AESRN is under consideration, the equivalent gate count is 3730 NAND equivalents for an ALERN dependent solution, and 3837 NAND equivalents for an ALFERN solution according to table 9-2 and table 9-3. Combining results from table 9-3 - table 9-6 gives the full implementation size, including a full range of 4 TCCFB blocks containing 24 4-LUTs contained within 24 TCFBs. Table 12-4 summarizes Asynchronous Event System area, and presents total area estimation in terms of NAND equivalents. A more detailed overview is presented in appendix table B-4.

Domain	NAND equivalents logic	NAND eq. additional logic	**Total NAND equivalents**
AGERN + ALERN	3730	1695	**5425**
AGERN + ALFERN	3837	1718	**5555**
AGERN + ALFERN + 4 TCCFBs	7865	3548	**11413**

Table 12-4: AESRN size overview

Even though the number derived for estimated logic area includes Atmel's standard routing and SRAM estimations presented in appendix B.1, an additional 10% have been added for extra routing because of the FPGA inspired topology. Other additional logic includes glue-logic, synchronizers and Event Channel selectors in peripherals. 25% has been added for this purpose, along with 0.125 – 0.250 kB of FLASH memory to store configuration bits for switch blocks and LUTs.

To illustrate the impact of TCFBs on the Asynchronous Event System size, numbers from table 9-5 can be used to conduct an estimation. Table 12-5 shows how a varying number of TCFBs influence the total area in terms of NAND equivalents. For a full overview the reader is referred to appendix table B-5. Note that the number of TCFBs also corresponds to the number of 4-LUTs with associated routing. Extra area occupied by Copy-elements inside each TCCFB will be added to the NAND equivalent logic as a factor of the total number of TCFBs after the formula:

$$\frac{\mathrm{NAND}_{\mathrm{TOT,CE}}}{\#\mathrm{TCFBs}_{\mathrm{TOT}}} \cdot \mathrm{N}_{\mathrm{TCFBs}} = \frac{908}{24} \cdot \mathrm{N}_{\mathrm{TCFBs}} \ .$$

# TCFBs	NAND equivalents logic	NAND eq. additional logic	**Total NAND equivalents**
0	3837	1718	**5555**
2	4173	2245	**6418**
4	4509	2351	**6860**
8	5181	2446	**7627**
12	5853	2827	**8680**
16	6525	3065	**9590**
24	7865	3548	**11413**

Table 12-5: TCFB impact on the Asynchronous Event System circuit size

Compared to the current Event System the functionally equivalent Asynchronous Event System is about 23% larger and a fully functional LUT version 154% larger in terms of NAND equivalents. The fully functional LUT version is large mainly because an increased amount of additional logic and routing resources. Two TCFBs can be included for an overhead of 43%, 4 TCFBs for 52% size increase, while 8 TCFBs require about 69% total area overhead compared to the Event System.

Pipeline distribution speed

According table 12-1 the AESRN is graded with good performance and power consumption. Performance results derived from table 11-2 estimates the event frequencies for different process technologies. The XMEGA peripheral clock can operate at a clock speed of 2-4 times the CPU clock speed of 32MHz [6], which makes the AESRN pipeline offer 6.5- 1.6 times faster event rate for the process range from 90 – 350nm compared to the XMEGA peripheral clock at 128MHz. Implemented in 600nm technology the Asynchronous Event System offers about the same performance as the peripheral clock speed used in the current Event System.

Although the rate tokens and events can enter the pipeline with are higher than offered by the original Event System, delays connected to the number of pipeline stages must also be taken into consideration. According to figure 9.1 the maximum amount of switches an event can encounter during global AGERN transportation is 7 switches for event transportation from an arbitrary Port / TC peripheral to a DAC / ADC peripheral. The maximum latency for an event traveling through the pipeline is given in table 12-6.

# Switch Blocks / Process	**3**	**4**	**5**	**6**	**7**
Delay 600nm	31.36ns	28.48ns	35.6ns	42.72ns	49.84ns
Delay 350nm	14.4ns	19.2ns	24ns	28.8ns	33.6ns
Delay 250nm	10.08ns	13.44ns	16.8ns	20.16ns	23.52ns
Delay 180nm	6.48ns	8.64ns	10.8ns	12.96ns	16.56ns
Delay 130nm	4.98ns	6.64ns	8.3ns	9.96ns	11.62ns
Delay 90nm	3.6ns	4.8ns	6ns	7.2ns	8.4ns

Table 12-6: Latency for events traversing switches

Table 12-6 is useful when analyzing if the assumption that an event is received in a peripheral with a maximum latency of two CPU clock cycles after it is issued is valid in the AESRN [6]. Two CPU clock cycles corresponds to $\frac{1}{32 \cdot 10^6 Hz} \cdot 2 = 62.5ns$, which is a demand being met by the Asynchronous Event System solution for all process technologies mentioned in table 12-6.

Energy Consumption and Programmability

Power consumption is only an issue in parts of the circuit participating in an event computation. This enables natural power down abilities for the AESRN at all times, making it less energy consuming than the original Event System. To find a scaling factor is difficult, because no average power consumption for the XMEGA is released in [6].

Programmability features are excellent, requiring 1764 bits for a fully programmable version with good computational power due to 24 4LUTs implemented in TCCFBs. Practically this means logical operations on up to 24 events concurrently, which is more than the global capacity of the AGERN. If both AGERN and ALERN / ALFERN resources are used at full capacity, 24 AGERN + 12/4 ALERN /ALFERN events can travel the pipeline concurrently. This is over 3 times the original Event System capacity of 8 events concurrently.
Ease of use is achieved through intuitive routing and easy programmability.

12.3 Associated Cost Challenges

When considering the costs of an implementation not only costs related to size and power consumption in terms of gates and energy usage are relevant. Costs tied to time-to-market, new settings for production equipment, development of test strategies, design cost, costs related to new tools and development software and so on are at least as important when looking at the big picture. An overview of costs regarding synchronous versus asynchronous implementations can be viewed in chapter 6, and this chapter will add more economical factors to the cost equation. Because it is hard to give good estimates to these cost factors, this section can be considered as a superficial overview putting the proposed Event System solutions in a marketing perspective.
Table 12-7 summarizes the most vital cost factors for a new design considered for manufacturing, and rates the proposed Event System Solutions from low- to high cost increase in different cost areas.

Event System Solution / Cost factor	HERN	CERN	AESRN
Process development			
Design Cost			
Time-to-market			
Test development			
Tool chain additions			
User software			

Table 12-7: Cost gradation of associated cost factors

The *process development* factor indicates if the implementation size of the solution could benefit from a more aggressive process technology. Both the CERN and AESRN demand a rather large area according to table 12-3 and table 12-4, and a more aggressive process technology would make the implementations more beneficial in terms of consumed chip area according to scaling theory [49].

Design Costs are potentially large for an AESRN design due to the factors mentioned in sections 3.3 and 6, and also considerable for the HERN I/O-processor. An AESRN implementation will also suffer from *test development* overhead, due to the asynchronous environment testing. New methods for testing would have to be developed, along with test tools adding to the total cost. HERN and CERN solutions can to a larger extent rely on conventional testing methods using well-known architectural elements.

Time-to-market is very important for a profitable release. CERN reuses much of existing Atmel CPLD technology, and is therefore considered the solution with best time-to-market potential. The AESRN will take time to develop, especially since new libraries for standard cells must be developed, and has the worst time-to-market potential of the solutions. HERN's custom I/O-processor represents the medium time-to-market alternative, and is also slightly more advanced than the CERN.
Tool-Chain additions are an important cost factor, and sorts under the *time-to-market* parole. For an AESRN solution this cost is considered high. Large programmability features means that a new mapping and allocation software should be implemented, customized for the Asynchronous Event System. Even though the routing structure itself is rather intuitive, the programmability feature is a new introduction to the whole tool development process. For the HERN a lighter programmability means less advanced tools, and interfacing with an I/O-processor should be familiar for the existing tool chain.

User software is needed to make the systems intuitive and easy to use. All solutions represent a relatively intuitive structure, and should impose few problems for the experienced designer.

12.4 Proposed Solution for Graphical Tool Design

To release the full potential of the new Event System solutions, tools supporting an increased level of programmability should be designed. It is not a part of this thesis to construct the applicable tools, but an overview of some ideas and concepts are presented to visualize intended use and application areas. Since the Asynchronous Event System has been the main area of focus, and partly realized as a Verilog model, this section will concentrate on tools for the constructed system.

First of all an interface managing to visualize the two-way routing system applied in the AESRN in an intuitive way to the user should be considered. To disconnect the user as much as possible from detailed routing decisions, a framework supporting an intuitive GUI could be a possible solution. Inspired from the AGERN representation depicted in figure 10.12, the whole AESRN with LUTs could be represented graphically, leaving to the user to direct which peripherals should communicate with each other over different Event Channels. Determining the best channel allocation is left to the routing tool, which must configure the routing network accordingly. Up to 1800 bits are required for the full functionality specter provided by AESRN resources, programming a single AGERN Event Channel route will need 32x7 = 224 bits to be written. This route represents the longest traversal path possible through routers in the presented architecture, and programs a route from a Port / TCCFB peripheral to an ADC / DAC peripheral.

One issue always present when considering programmability and routing is increased complexity. For a Xilinx Virtex II device the amount of programming bits range from 339kB in the simplest device to 26Mb in the most advanced device, according to table 26 in [50].
Considering an Asynchronous Event System requiring 1.8kb, or 0.25kB, of configuration bits for programming full routing and computational LUT functionality, a relatively simple tool should be an appropriate solution. With such a tool implemented, the user should have the option if the path allocation for each event should be a manual procedure for full custom layout, or if the tool abstracts the user from this level by introducing GUI elements generating the correct programming bits when representing an allocation.

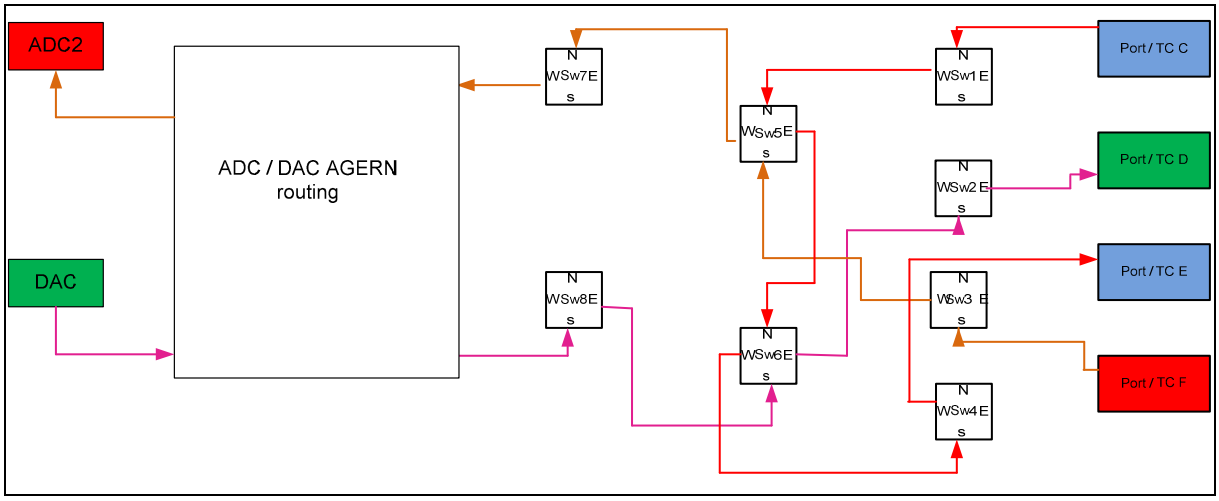


Figure 12.1: GUI example for event routing on the AGERN

One intuitive GUI solution is presented in figure 12.1. The example is based on the event routing scenario from table 10-1, and the reader is referred to section 10.4 for further details. The GUI represents all Event System peripherals and AGERN / ALFERN elements at a block schematic level. In figure 12.1 the block representing the Port / TCCFB side of the AGERN is abstracted away to show all switch details. The maximum level of detail could be presenting all output / inputs for each switch,

as in figure 10.14. Showing the selected event routes in highlighted color gives the user the overview to determine the event flow.

If all internal connectivity in the AGERN boxes is hidden, the user is left with specifying which peripheral should be used as event source, and which peripheral should be used for destination. By clicking on the source and destination peripherals the tool determines the best available route, and sets up the proper configuration bits for event channel allocation.

In addition to deciding the source and destination peripherals, clicking on the switch blocks enables manual routing possibilities between the peripherals. This function makes it possible to pin-point an event route, and the tool will generate the proper configuration bits as the route is determined.

The tool is also responsible for setting the appropriate connections at the peripheral level, such that an allocated Event Channel is guaranteed to trigger the determined peripheral actions. Of course there must exist a function for full user-programmability, disabling much of the allocation help provided by the tool. Being a rather small system, manual routing is absolutely achievable and has been practiced by the author in all test cases presented throughout this thesis.

13 Discussion

A choice was made during this thesis to focus on research related to an asynchronous Event System. It is therefore natural to discuss the model that was developed to represent the asynchronous system, both as an architectural model as a whole, and as a modular system. First of all, the suggested routing topology is a custom designed pseudo-hierarchical solution consisting of custom designed switch blocks. Each switch block is constructed after a commonly accepted model for switch design, and modified to fit into the given routing context. Some uncertainties are brought into the picture regarding this design methodology, because none of the proposed switch blocks have been designed or tested in previous literature according to the researched source material. In essence FPGA technology has been modified and re-designed to fit into a microcontroller environment, and it is therefore not certain that the accepted models for FPGA elements proposed in literature remain valid. For all the designed elements presented in this thesis, the basic assumption is that a functional FPGA model is also valid in terms of IC-design.

To justify such an assumption, it was important to design a Verilog model containing as many asynchronous design principles as possible. Through simulations built on exact models of the custom designed circuit elements, the system behavior was verified and the fundamental models behaved like expected in the test environment. However, simulations alone cannot be considered proof that the model is verified, especially considering the simulation issues emphasized in section 10.1.
A synthesis was therefore conducted to investigate if the circuit model was synthesizable. This was indeed the case, but still sufficient proof of complete model verification cannot be considered obtained. The reason is the immense complexity of asynchronous circuitry at the layout level. Delay matching is a critical issue not easy to replicate in simulations, and as mentioned in section 10.5 and in background theory conventional synthesis tools do not support delay matching to the extent needed for the proposed asynchronous design. In worst case this could mean that even though the designed model behaved correctly according to simulations, it could be hard to construct a working physical layout. A direct consequence is that a realistic hazard analysis of the system is not obtained. This fact leaves some critical areas open to further research, and makes it difficult to be certain about if an Asynchronous Event System is practical for an XMEGA implementation. In depth hazard research is therefore considered an important part of future work.

Asynchronous testing is a vast area barely touched in the scope of this thesis.This choice was made in order to keep focus on the model and architectural development, and present a result which to a greater extent can narrow the search for test methodologies to the ones specifically applicable for the model. Verification by simulation served as proof of the conceptual models, leaving test development as a good suggestion for future work.

Uncertainties regarding the developed model are also to some extent reflected in analysis considering the asynchronous implementation. Estimated gate count is derived from the architectural models under the assumption that these are valid. The most uncertain estimates are considering additional logic and routing area consumed by wires. Included in the presented gate count are standard numbers used by Atmel for routing area influence, and an additional 10% has been added to compensate for the FPGA inspired topology demanding more routing resources. 25 % has been added for additional logic like synchronizers, interface-logic and control logic. Since developing an asynchronous Event System would require much custom designed elements and new standard cell layouts, good area utilization is deemed possible. This includes clustering of SRAM cells and effective routing layout, possibly making the area contribution from extra routing and interface logic smaller than expected. The implementation area is also dependent on the optimization from the synthesis tool, making an exact estimation hard to derive without a physical layout.

Estimated results put an Asynchronous Event System with the same feature range as the current Event System about 23% larger, using 5555 NAND equivalents including 0.125 kB of additional FLASH memory. An Asynchronous Event System with full range computational functionality featuring 24 interconnected 4-LUTs for logical computation demands an estimated area increase of 154% compared to the current Event System. In terms of NAND gate equivalents, the full range version with 0.25 kB extra FLASH memory uses 11413 NAND gates.

Being only a sub-system on the XMEGA microcontroller an increase of 23% in area is considerable, and 154% out of the question. To put these numbers in a technology perspective, the scaling theory applied for AESRN performance analysis in section 11.3 also estimates a decrease in area of 50% for each new process generation [49]. It must be kept in mind that these are only estimated numbers, but they give a good indication as to what potential future technologies hold for integrating more complex architectures. Scaling theory was also used to estimate the performance of the pipeline structure used in the Asynchronous Event System. Numerous references were used to obtain data for estimation, and many of these emphasized variable process parameters influencing the final result with respect to circuit delay. A reference “worst-case” NAND gate delay was derived from a paper applying test conditions similar to the AESRN pipeline. Scaling this number from a 90nm process to preceding processes was done by a scaling factor commonly accepted in literature [49]. The potential for miscalculations because of process dependent variables is present, and the derived numbers should therefore be considered as guidelines for performance rather than absolute reliable values. However, compared to other attempts in literature using the same pipeline structures, the numbers were consistent enough to be plausible performance results.

Synchronizers interfacing the synchronous peripheral domain are not included in the pure pipeline event rate calculations, but will add to the delay when applied for a realistic implementation. In worst case with a two-flop synchronizer the added delay could be as much as two CPU clock cycles. Continued research in the field of synchronizers applicable for the Asynchronous Event System is therefore essential to come up with reasonable event transportation latency.

Possible savings in power consumption by applying an asynchronous transfer protocol is perhaps the most interesting feature offered by an Asynchronous Event System. Unfortunately it is also the hardest feature to estimate and predict. With no accurate data considering power consumption or energy usage for the Event System [6], measurements are hard to derive even at estimation level and would leave these numbers to pure guesswork.

Although being the main focus of this thesis, the Asynchronous Event System is a result of two preceding architectural suggestions. First to introduce the concept of programmable routing was the HERN solution with embedded I/O-processor. Analysis with respect to size measurement can be deemed accurate for the I/O-processor based on earlier Atmel research [9], and for the switch blocks the model first discussed in this section was used. According to table 7-1 the proposed HERN routing architecture with the simplest I/O-processor developed by Atmel [9] measures 3150 NAND equivalent gates, or about 43% smaller than the current Event System. The most realistic alternative is a more advanced I/O-processor, which increases the gate count to 4650 NAND. As for the asynchronous solution with no computational elements this HERN architecture could be interesting for implementation in a more aggressive process. The biggest issue by introducing an I/O-processor is perhaps associated energy consumption, which requires careful design optimizations and power down capabilities to compete with the current Event System. Analyses on these issues are also hard to obtain due to the custom layout required for each processor alternative, and must be associated with the applied instruction sets.

The second solution relies on CPLD architectural elements, and implements these as an additional peripheral unit available to selected Port and TC modules. Even though Atmel's own CPLD design is used as basis, it is very difficult to derive exact estimates for size, performance and power consumption. Datasheets used for these estimations does not provide detailed information on these areas when scaled to fit an XMEGA implementation, and the estimations should therefore be considered strictly as guidelines. The programmability in the CERN solution lies in configuring each product term computed by each macrocell.
By controlling the number of configurable MCBs associated with each Port / TC peripheral, a tailored implementation is achievable. Communicating with traditional bus solutions, this solution is the most easy to interface into the existing Event System, and is considered the most promising Event System solution in a "near-future" development perspective. Suffering from area overhead due to the large amount of configuration bits needed to configure the CERN, some work tailoring the proposed architecture to an XMEGA implementation is essential before launching the CERN as a new Event System.

Common for all proposed Event System solutions is the concept of programmability. Supporting the required amount of configuration bits by FLASH memory a limited amount of configuration bits can be included at a relatively cheap cost increase. Ranging from approximately 416 configuration bits for the HERN solution, 1800 bits for a full range functional AESRN solution and 3120 bits for the smallest CERN solution, the amount of programming bits are considerable. Another issue that has not been investigated is to determine how programmability should be solved run-time, and how configuration bits can be efficiently supported by FLASH. Programmability is a nice feature, and enables the user to customize the system to a larger extent than the original Event System. Programmability issues include configuration setup-time, extra routing from FLASH memory to the switch block memory cells and extra area to hold the configuration bits. Larger area consumed by memory also increases energy consumption, which means that it is an application choice to determine the level of programmability which is most cost effective.

In terms of meeting the requirements defined in the problem description of this thesis, all proposed solutions each represent a unique way to approach the defined problems. Cost effective routing of signals is achieved for all solutions, because the extent of available programmability can account for the area increase compared to more traditional MUX or bus based solutions. Included is also the hierarchical aspect, which in the AESRN simulations indicates less latency than global signals. Concerning implementation costs, all solutions are more expensive than the current Event System. With increased functionality the cost versus functionality ratio is an application and device dependent factor, and all solutions are scalable with respect to event channels and computational elements. For instance the AESRN solution can offer increased capacity, performance, flexibility and a possible decrease in power consumption. The penalty is larger circuit area, increased design time and more complex tools. Peripherals are reused or modified for increased functionality in all solutions. Although many of the presented problems have been solved, this section also points out that much more work must be done before a new Event System prototype which fundamentally changes the existing architecture can be realized.

14 Conclusion and Future Work

14.1 Conclusion

The research conducted throughout this thesis has resulted in three distinct proposals for new Event System solutions; The HERN, CERN and Asynchronous Event System. All solutions have been analyzed with respect to estimated performance and area, including an overall cost analysis for non-measurable cost metrics related to market and development costs. Special focus was given on the research needed to develop an asynchronous solution for Event System implementation, and the result features a synthesized model of such an Event System designed in Verilog. Simulations used to verify the functionality of the Asynchronous Event System have successfully distributed 24 events over the global routing network AGERN. Cooperating with the local ALFERN network, a maximum capacity of 32 unique events concurrently is achievable, which is four times the capacity of the current Event System. According to simulation asynchronous LUT functionality has been included according to architectural specifications, promising logic computational elements for events. Fully programmable routing has also been simulated, making arbitrary routing possible between peripherals. The simulations display relevant test scenarios, and are regarded as proof of the asynchronous event distribution concept introduced by the Asynchronous Event System.

Area estimations for Atmel's 35k9 process indicate a size 23% larger than the current Event System for an AGERN/ ALFERN routing system without computational elements. In terms of NAND equivalents this equals 5555 NAND gates. An addition of 2 TCFBs with one 4-LUT each increases the area overhead to 43% with 6418 NAND gates, while 8 TCFBs demand 69% increase costing 7627 NAND gates. Estimated performance gain estimates the Asynchronous Event System between 6.5 and 1.6 times faster than the Event System in pure event transportation rate depending on the process technology applied for implementation. Concluding on the estimated numbers, the prospect of implementing an asynchronous Event System becomes more achievable as process geometry decrease, which according to scaling theory will reduce the gate count by 50% for the next process generation. Having experienced the difficulties of asynchronous design, another conclusion indicates that more research on specialized synthesis tools and test methods must be initiated before a commercial Asynchronous Event System is realistic. An asynchronous Event System is therefore not the best solution for a "near-future" Event System.

The HERN and CERN solutions have also been analyzed with respect to area and performance. A simple HERN with full range programmable routing is estimated at 3150 NAND gates, while and advanced I/O-processor would demand up to 7525 NAND gates. Estimated area compared to the Event System is 43% smaller and 67% larger for Atmel's 35k9 process. A CERN solution with 1 CPLD MCB in each Port / TC peripheral occupies 5308 NAND equivalents due to large memory requirements. Due to the simple interface towards existing peripherals for the CERN solution, this solution shows the best prospects for "near-future" Event System implementation.

14.2 Future Work

To ensure correctness of the asynchronous model, work on developing a library of asynchronous elements by handmapping should be an interesting area of research. Mainly targeted for FPGA implementation, a full scale realistic test with the Asynchronous Event System as a functional circuit could provide valuable data in areas like delay measurements and power consumption. Based on the Verilog designs conducted for this thesis, optimizing the given code and fabricate a layout for testing purposes should be an achievable task. Only with obtained test data is it possible to verify or discard the proposed architecture as a model for future implementation. Reducing the configuration bit count could also be achieved by substituting the applied one-hot encoding with a more intricate configuration scheme.

Another aspect of continued asynchronous research is applying a new handshake protocol to further increase circuit speed. Some of the referenced literature includes research on ultra-fast pipeline structures, maintaining higher computation rate than the current 4-PDR handshake protocol. If such protocols could be implemented with minimal area overhead compared to the 4-PDR protocol, it could prove a good addition. Not presenting detailed research in the field of hazards associated with asynchronous circuits in the scope of this thesis, continued work in this area will be important to conclude further on the feasibility of an asynchronous Event System. Especially considering that the synchronizer work presented in [24] [26] [25] does not consider a 4-PDR protocol, but relies on the 4-PBD protocol. Integrating synchronizers as part of the suggested implementation work would be a valuable addition to the Asynchronous Event System research with respect to total system latency.

As an additional task to more accurate power estimations, the effect of using more aggressive process technologies is an alternative area of research. As transistor size decrease, the static power consumption increase while dynamic power consumption decrease. Determination of this relationship is important to characterize the power benefits offered by an asynchronous solution.

I/O-processor research is also an area of focus, along with developing a more tailored CPLD based solution with limited programmability compared to the existing CERN solution. A more in depth analysis on which signals that should be included for each CPLD MCB and how many MCBs that are needed to implement efficient event computations in each Port / TC peripheral, are good suggestions for future work.

By concentrating on the lower abstraction levels of an implementation, this thesis leaves much work in the area of tool development to support programmability features, and analyze if programmable solutions can be integrated in the tool flow at a reasonable cost at all. How to effectively program all configurations and map the connections towards each programmable element is also a future challenge. This will include researching effective instruction sets for fast run-time FLASH configuration.

15 Sources

[1] John Teifel and Rajit Manohar: **"An asynchronous dataflow FPGA architecture"**. *IEEE Transactions on Computers. Volume 53, issue 11. November 2004, pages 1376 -1392.*

[2] Lars S. Nielsen and Jens Sparsø: **"Designing Asynchronous Circuits for Low Power: An IFIR Filter Bank for a Digital Hearing Aid"**. *Proceedings of the IEEE, Volume 87, Issue 2. February 1999, pages 268-281.*

[3] Herman Schmit and Vikas Chandra: **"Layout Techniques for FPGA Switch Blocks"**. *IEEE Transactions on Very Large Scale Integration (VLSI) System,. Volume 13, issue 1. January 2005, pages 95-105.*

[4] Scott Hauck, Gaetano Borriello, Steven Burns and Carl Ebeling: **"MONTAGE: An FPGA for Synchronous and Asynchronous Circuits**". *Field-Programmable Gate Arrays: Architectures and Tools for Rapid Prototyping. Springer Link 1993, pages 44-51.*

[5] Alireza Kaviani and Stephen Brown: **"Hybrid FPGA Architecture"**. *Proceedings of the Fourth International ACM Symposium on Field Programmable Gate Arrays. 1996, pages 3-9.*

[6] **Preliminary AVR XMEGA A Manual**
Available at: http://www.atmel.com/dyn/products/datasheets.asp?family_id=607.
Last updated: 04-2009, Last Cited: 28.05-2009. Doc Rev: G.

[7] **Atmel Datasheet ATF1508RE High-Performance CPLD**
Available at: http://atmel.com/dyn/products/datasheets.asp?family_id=653
Last updated: 10-2008, Last Cited: 02.06-2009. Doc Rev: A.

[8] **Atmel Documentation: Multifunctional Timer I/O unit (Internal)**
Available at: Atmel Internal Document

[9] **Atmel Documentation: Previous Research on Implementation of I/O-processors. (Internal).** *Available at: Atmel Internal Document.*

[10] **Atmel Documentation: 35k9 in-house Process Standard Cell Library (Internal)**.
Available at: Atmel Internal Document.

[11] Paul Chow, Soon Ong Seo, Jonathan Rose et Al. : **"The Design of an SRAM-Based Field Programmable Gate Array- Part I: Architecture"**. *IEEE Transactions on VLSI systems. Volume 7, Issue 2. June 1999, pages 191-197.*

[12] Paul Chow, Soon Ong Seo, Jonathan Rose et Al. : **"The Design of an SRAM-Based Field Programmable Gate Array- Part II: Circuit Design and Layout"**. *IEEE Transactions on VLSI systems. Volume 7, Issue 3.September 1999, pages 321-330.*

[13] Scott Hauck: **"Asynchronous Design Methodologies: An Overview"**. *Proceedings of the IEEE. Vol. 83, issue 1. January 1995, pages 69-93.*

[14] Jonathan Rose, Abbas el Gamal and Albert Sangiovanni-Vincentelli: **"Architecture of Field-Programmable Gate Arrays".** *Proceedings of the IEEE Volume 81, issue 7. July 1993, pages 1013-1029.*

[15] Aditya A. Aggarwal and David M. Lewis: "**Routing Architectures for Hierarchical Field-Programmable Gate Arrays**". *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors. October 1994, pages 475-478.*

[16] Marcos Ferretti and Peter A. Beerel: **"Single-track Asynchronous Pipeline Templates Using 1-of-N Encoding"***. Proceedings of Automation and Test in Europe Conference and Exhibition, 2002. 2002, pages 1008-1015.*

[17] Herman Schmit and Vikas Chandra: **"FPGA Switch Block Layout and Evalutation"**. *Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays. February 2002, pages 11-18.*

[18] Vikas Chandra and Herman Schmit: **"Simultaneous Optimization of Driving Buffer and Routing Switch Sizes in an FPGA using an Iso-Area Approach"**. *IEEE Computer Society Annual Symposium on VLSI, 2002. August 2002, pages 28-33.*

[19] Yao-When Chang , D. F. Wong and C. K. Wong : **"Universal Switch-Module Design for Symmetric-Array-Based FPGAs"**. *Field-Programmable Gate Arrays, 1996. FPGA '96. Proceedings of the 1996 ACM Fourth International Symposium. 1996, pages 80-86.*

[20] Yao-When Chang and D. F. Wong: **"Universal Switch Modules for FPGA Design"**. *ACM Transactions on Design and Automation of Electronic Systems. Volume 1, issue 1. January 1996, pages 80-101.*

[21] R. Payne : **"Asynchronous FPGA architectures"**. *IEEE Proceedings on Computers and Digital Techniques. Volume 143, Issue 5. September 1996, pages 282-286.*

[22] Jens Sparsø and Steve Furber: **"Principles of Asynchronous Circuit Design – A Systems Perspective"**. *Book/Web, Springer Link, ISBN: 0792376137, 9780792376132. 2001. Available as pdf.*

[23] Arne Aas: **"Metastabilitet ved synkronisering"** . *Elektronikk. Issue: 4, 1995.*

[24] Ran Ginosar: **"Fourteen Ways to Fool Your Synchronizer"**. *Proceedings of the Ninth International Symposium on Asynchronous Circuits and Systems, 2003. May 2003, pages 89-96*.

[25] Jens U. Horstmann, Hans W. Eichel and Robert L. Coates: **"Metastability Behavior of CMOS ASIC Flip-Flops in Theory and Test"**. *IEEE Journal of Solid-State Circuits, Volume 24, Issue 1. February 1989, pages 146-157.*

[26] Mika Nyström and Alain J. Martin: **"Crossing the Synchronous – Asynchronous Divide"**. *Workshop on Complexity Effective design. May 2002.*

[27] Atabak Mahram, Mehrdad Najibi and Hossein Pedram: **"An Asynchronous FPGA Logic Cell Implementation"**. *Proceedings of the 17th ACM Great Lakes symposium on VLSI. March 2007, pages 176-179.*

[28] Atabak Mahram, Behnam Ghavami and Hossein Pedram: **"The Impact of copy-elements on QDI Asynchronous FPGA Interconnect Structure"**. *International Conference on Design & Technology of Integrated Systems in Nanoscale Era. September 2007, pages 87-91.*

[29] Alexander Borisovitch Smirnov: "**Asynchronous Micropipeline Synthesis System"**. *Doctor of Philosophy 2009, Boston University.*

[30] Quoc Thai Ho, Jean-Baptiste Rigaud, Laurent Fesquet et Al.: **"Implementing Asynchronous Circuits on LUT Based FPGAs**". *Springer Link, Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream. ISBN: 978-3-540-44108-3. January 2002, pages 36-46.*

[31] Tzyh-Yung Wuu and Sarma B. K. Vrudhula: **"A Design of a Fast and Area Efficient Multi-Input Muller C-element"**. *IEEE Transactions on VLSI Systems, Volume 1, Issue 2. June 1993, pages 215-219.*

[32] John Taifel and Rajit Manohar: **"Highly Pipelined Asynchronous FPGAs**". *Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays. 2004, pages 133-142.*

[33] Danil Sokolov ,Julian Murphy, Alexander Bystrov and Alex Jakovlev: **"Design and analysis of Dual-Rail Circuits for Security Applications"**. *IEEE Transactions on Computers. Volume 54, Issue 4. April 2005, pages 449-460.*

[34] Katherine Compton and Scott Hauck: **"Reconfigurable Computing: A Survey of Systems and Software"**. *ACM Computer Survey 2002, Volume 34. June 2002, pages 171-210.*

[35] Eslam Yahya and Marc Renaudin: **"Performance Modeling and Analysis of Asynchronous Linear-Pipeline with Time Variable Delays"**, *IEEE Electronics, Circuits and Systems, 14th IEEE International Conference. December 2007, pages 1288-1291*

[36] Eslam Yahya and Marc Renaudin**: "QDI Latches Characteristics and Asynchronous Linear Pipeline Performance Analysis"**, *Springer Link, Lecture Notes in Computer Science. September 2006,pages 583-592*

[37] Feng Shi, Yiorgos Makris, Steven M. Nowick and Montek Singh: **"Test Generation for Ultra -High-Speed Asynchronous Pipelines"**. *IEEE International Test Conference, Proceedings. ITC 2005. November 2005, Pages 1008-1018*.

[38] Montek Singh and Steven M. Nowick: **"High-Throughput Asynchronous Pipelines for Fine-Grain Dynamic Datapaths"**. *Sixth International Symposium on Advanced Research in Asynchronous Circuits and Systems, 2000. (ASYNC 2000) Proceedings. April 2000, pages 198-209.*

[39] Andrew Matthew Lines: **"Pipelined Asynchronous Circuits"**. *Master's Thesis, California Inst. Of Technology. June 1995.*

[40] I. E. Sutherland: **"Micropipelines"**. *Communications of the ACM. June 1989, pages 720-738.*

[41] Matthew L. King and Kewal K. Saluja: **"Testing Micropipelined Asynchronous Circuits"**. *International Test Conference 2004 (ITC 2004) proceedings. October 2004, pages 329-338.*

[42] Stephen Brown and Jonathan Rose: **"FPGA and CPLD Architectures: A Tutorial"**. *IEEE Design & Test of Computers. Volume 13, issue 2. July 1996, pages 42-57.*

[43] Paul Friedberg, Yu Cao, Ruth Wang, Jan Rabaey and Costas Spanos: **"Modeling Within-Die Spatial Correlation Effects for Process-Design Co-Optimization"**. *Proceedings of the Sixt International Symposium on Quality Electronic Design (ISQED'05). March 2005, pages 516-521.*

[44] Yu Cao and Lawrence T. Clark: **"Mapping Statistical Process Variations Toward Circuit Performance Variability: An Analytical Modeling Approach"**. *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems. Volume 26, issue 10. October 2007, pages 1866-1873.*

[45] Kenichi Okada, Kento Yamaoka and Hidetoshi Onodera: **"A Statistical Gate-Delay Model Considering Intra-Gate Variability"**. *Proceedings of the International Conference on Computer Aided Design (ICCAD'03). November 2003, pages 908-913k.*

[46] Peggy B McGee, Steven M. Nowick and E. G. Coffman Jr.: **"Efficient Performance Analysis of Asynchronous Systems Based on Periodicity"**. *Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/Software Co-design and System Synthesis. September 2005, pages 225-230.*

[47] Jayanth Srinivasan, Sarita V. Adve, Pradip Bose and Jude A. Rivers: **"The Impact on Technology Scaling on Lifetime Reliability"**. *The International Conference on Dependable Systems and Networks (DSN'04). June 2004, pages 177-186.*

[48] Rostislav Dobkin, Ran Ginosar and Avinoam Kolodny: **"Fast Asynchronous Shift Register for Bit-Serial Communication"**. *Proceedings of the 12th IEEE International Symposium on Asynchronous Circuits and Systems (Asynch'06). March 2006, pages 118-127.*

[49] Shekar Borkar: **"Design Challenges of Technology Scaling"**. *IEEE Micro, Volume 19, issue 5. August 1999, pages 23-29.*

[50] **Xilinx Virtex II Platform FPGAs: Complete Datasheet.** *Available at: http://www.xilinx.com/support/documentation/virtex-ii.htm. Last updated: 05.11-2007, Cited: 28.05-2009. Doc Rev: 3.5*

[51] Mark Holland and Scott Hauck: **"Improving Performance and Robustness of Domain-Specific CPLDs"**. *Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays. 2006, pages 50-59.*

[52] Mark Holland: **"Automatic Creation of Product-Term-Based Reconfigurable Architectures for System-on-a-Chip"**. *Doctor of Philosophy Dissertation. University of Washington 2005.*

[53] Jakov N. Seizovic: **"Pipeline Synchronization"**. *Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems, 1994. November 1994, pages 87-96.*

[54] **Suzaku-S Hardware Manual, Atmark Techno**. *Available at http://www.atmark-techno.com/en/products/suzaku/suzaku-s Last updated: 14/12-2004, Cited: 25.04-2009. Doc. Rev: 1.0.4.*

16 Appendices

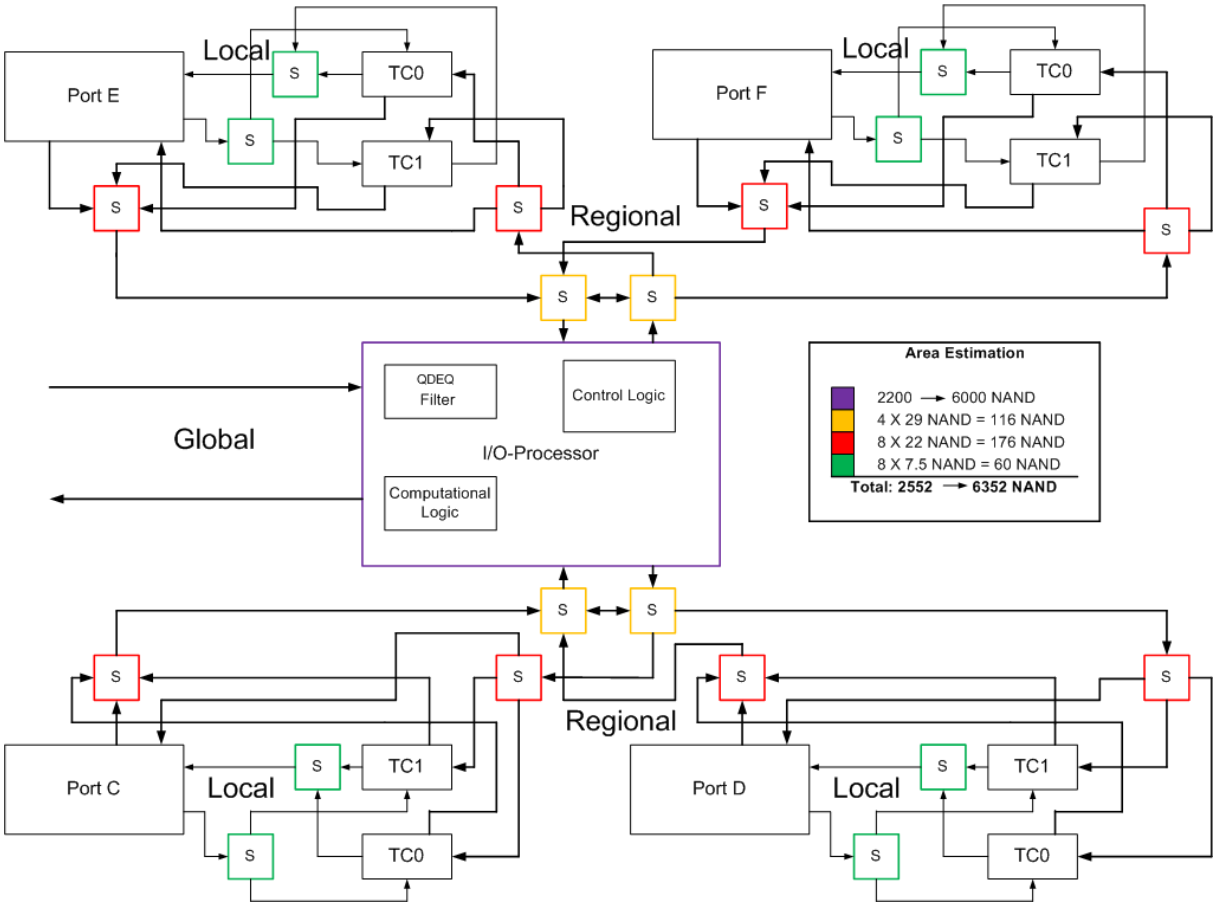
Appendix A Contents

Appendix A Figures

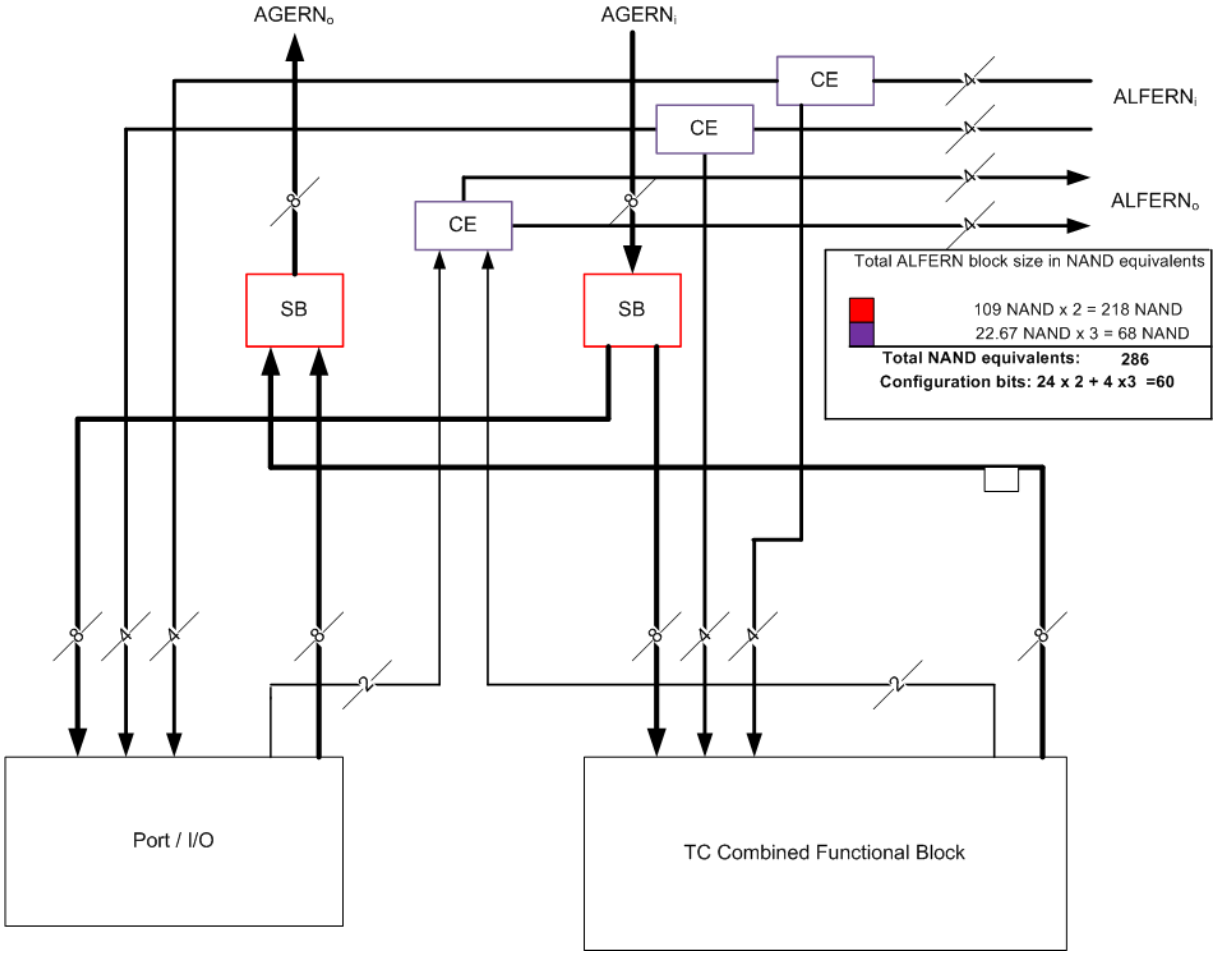
Appendic C Contents

16.1 Architectural Drawings

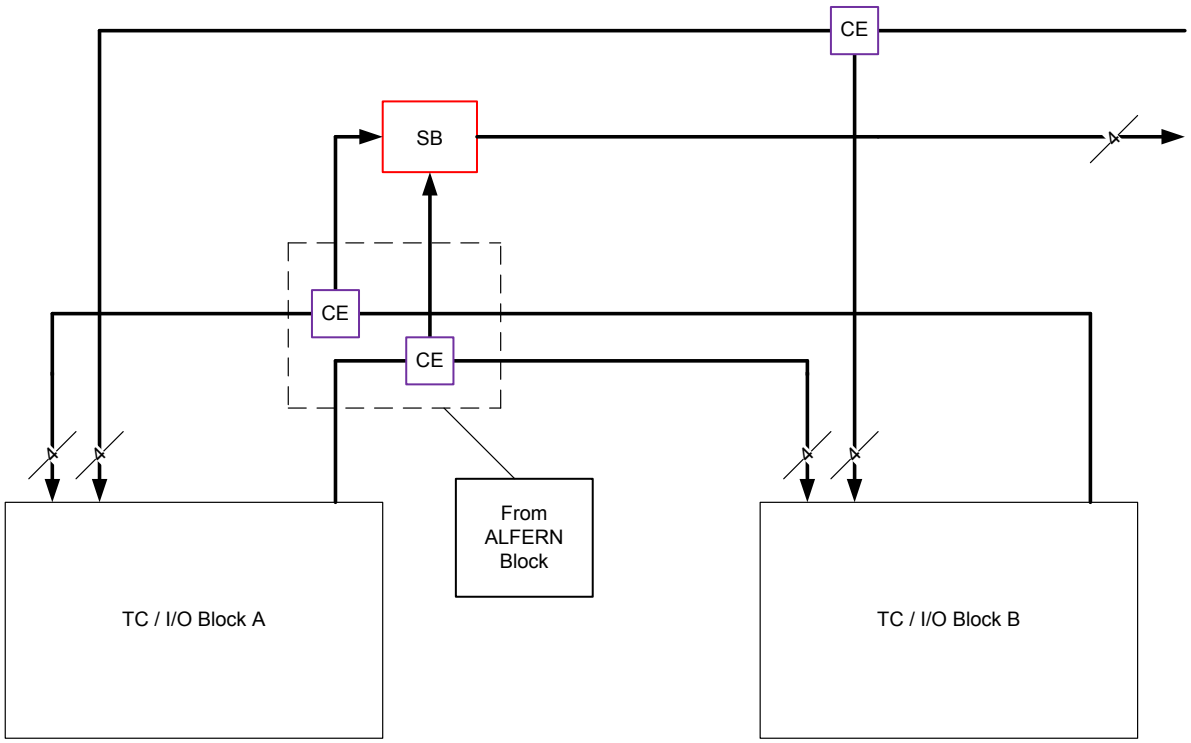
A 16.1-1: HERN Hierarchical Overview



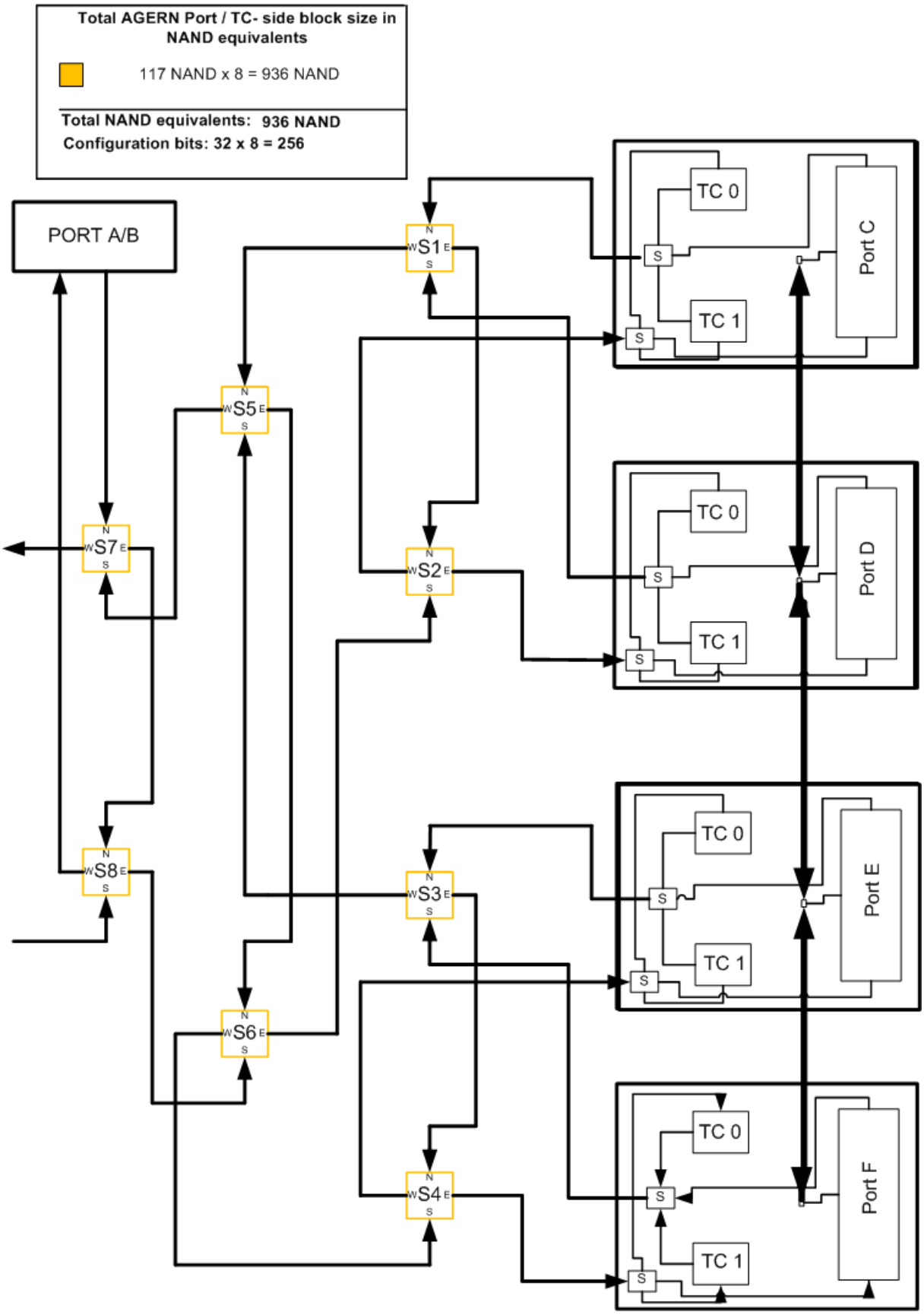
A 16.1-2: ALFERN Local View



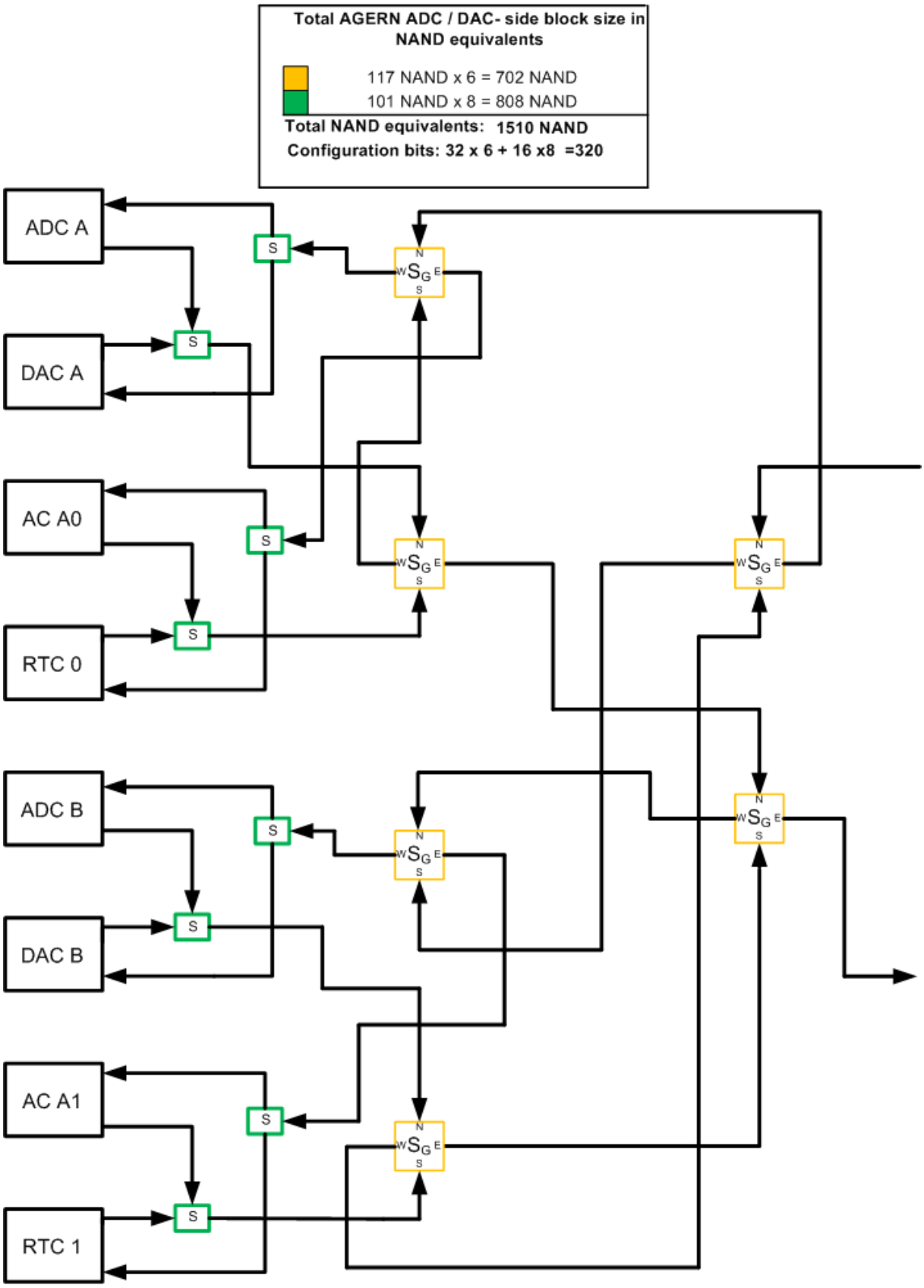
A 16.1-3: ALFERN Hierarchical View



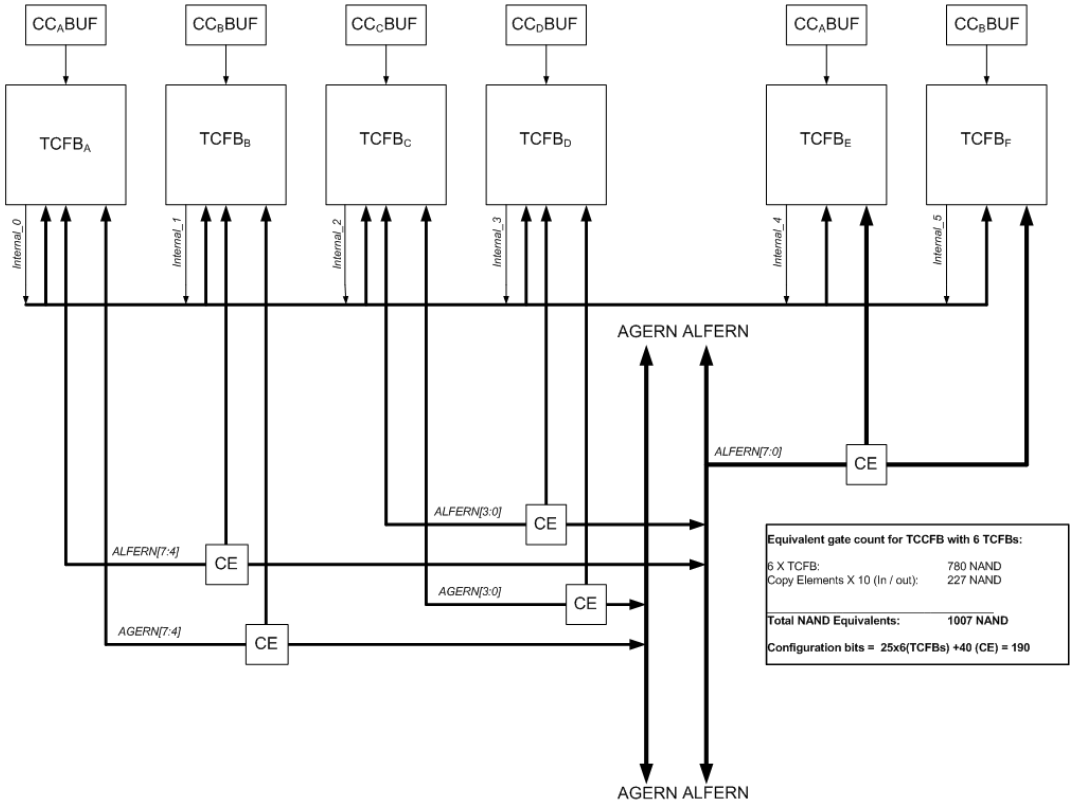
A 16.1-4: AGERN viewed from Port / TC Peripheral side



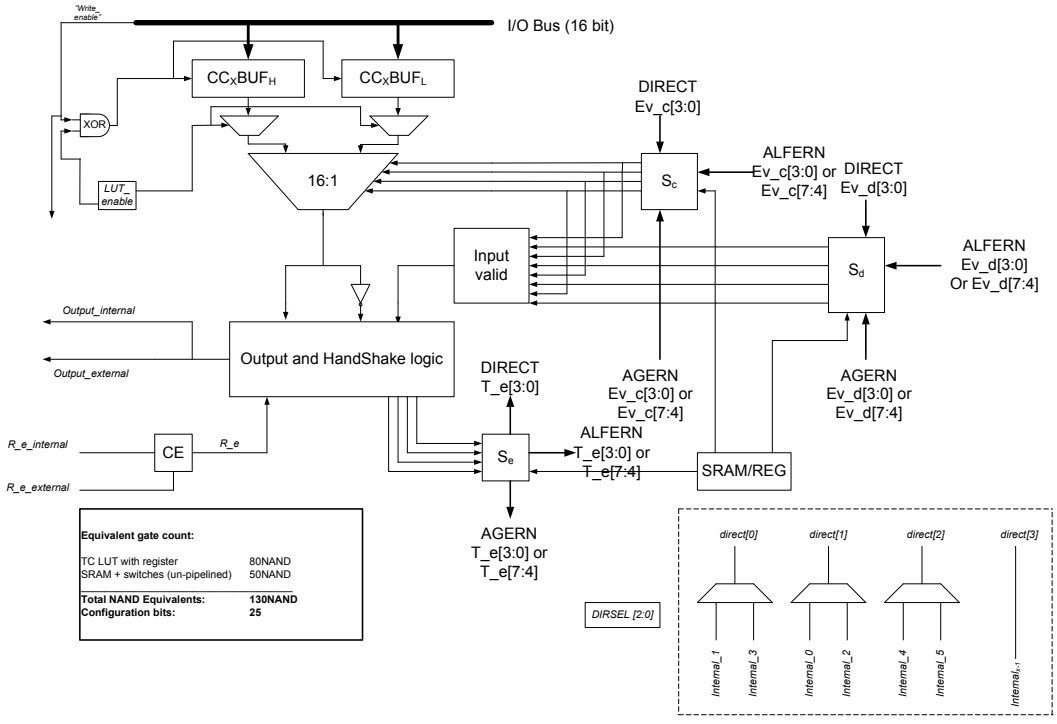
A 16.1-5: AGERN Viewed from ADC / DAC side



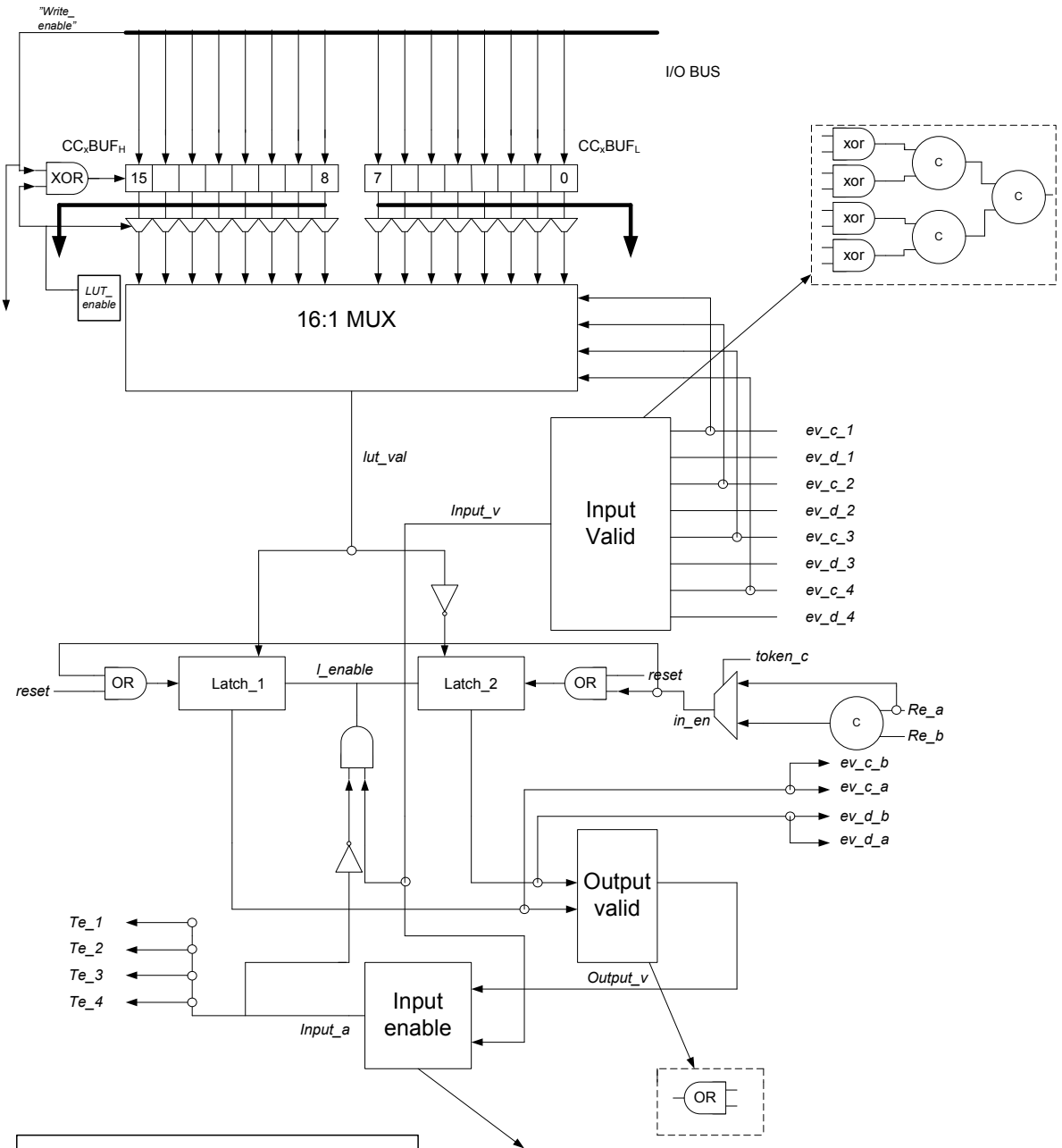
A 16.1-6: Timer Counter Combined Functional Block Overview



A 16.1-7: Timer Counter Functional Block Overview



A 16.1-8: Timer Counter Computational Block overview

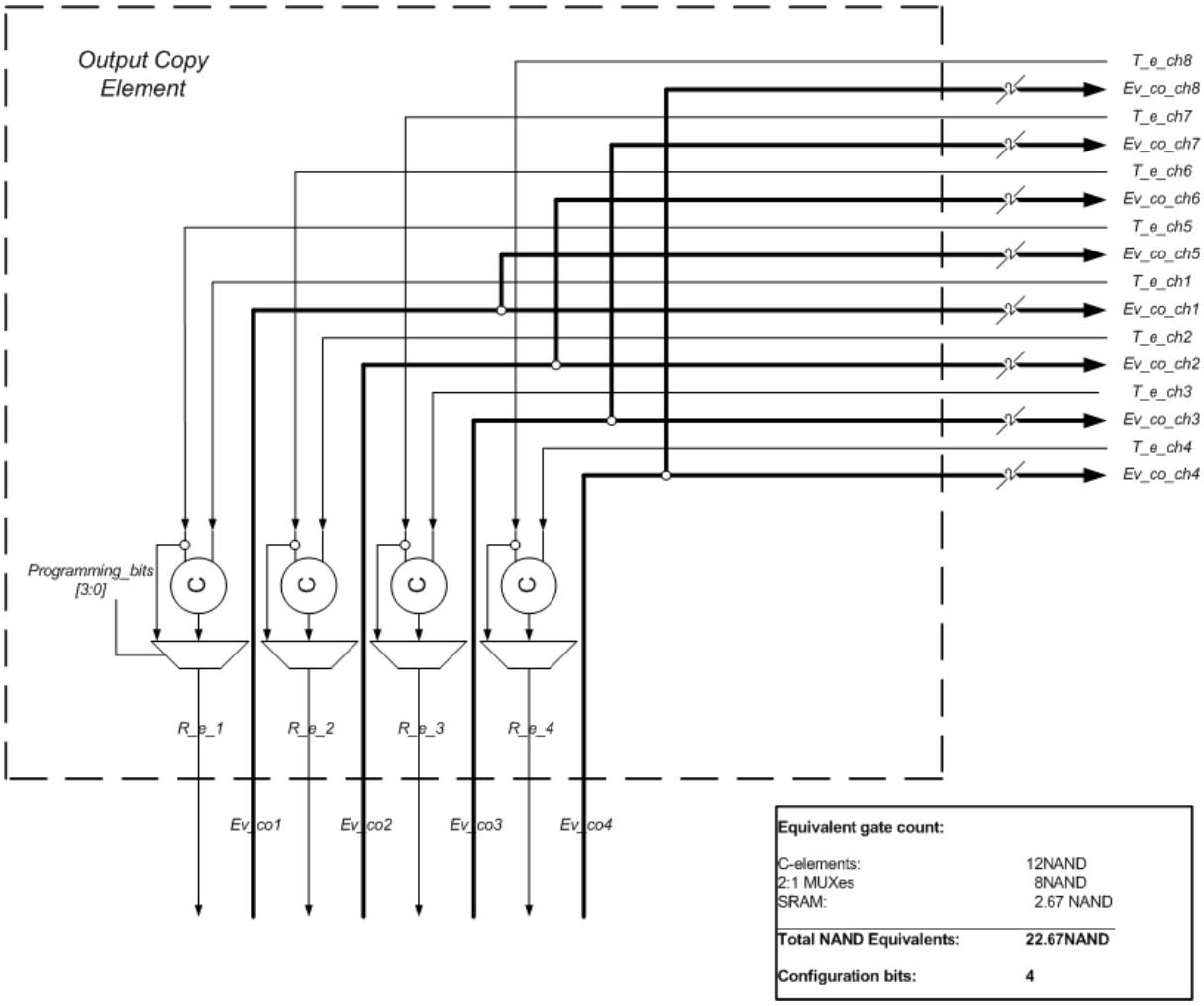


Equivalent gate count:

Total NAND: 80 NAND

Refer to table B-1 in appendix B.1 for standard cell values used for architectural calculations.

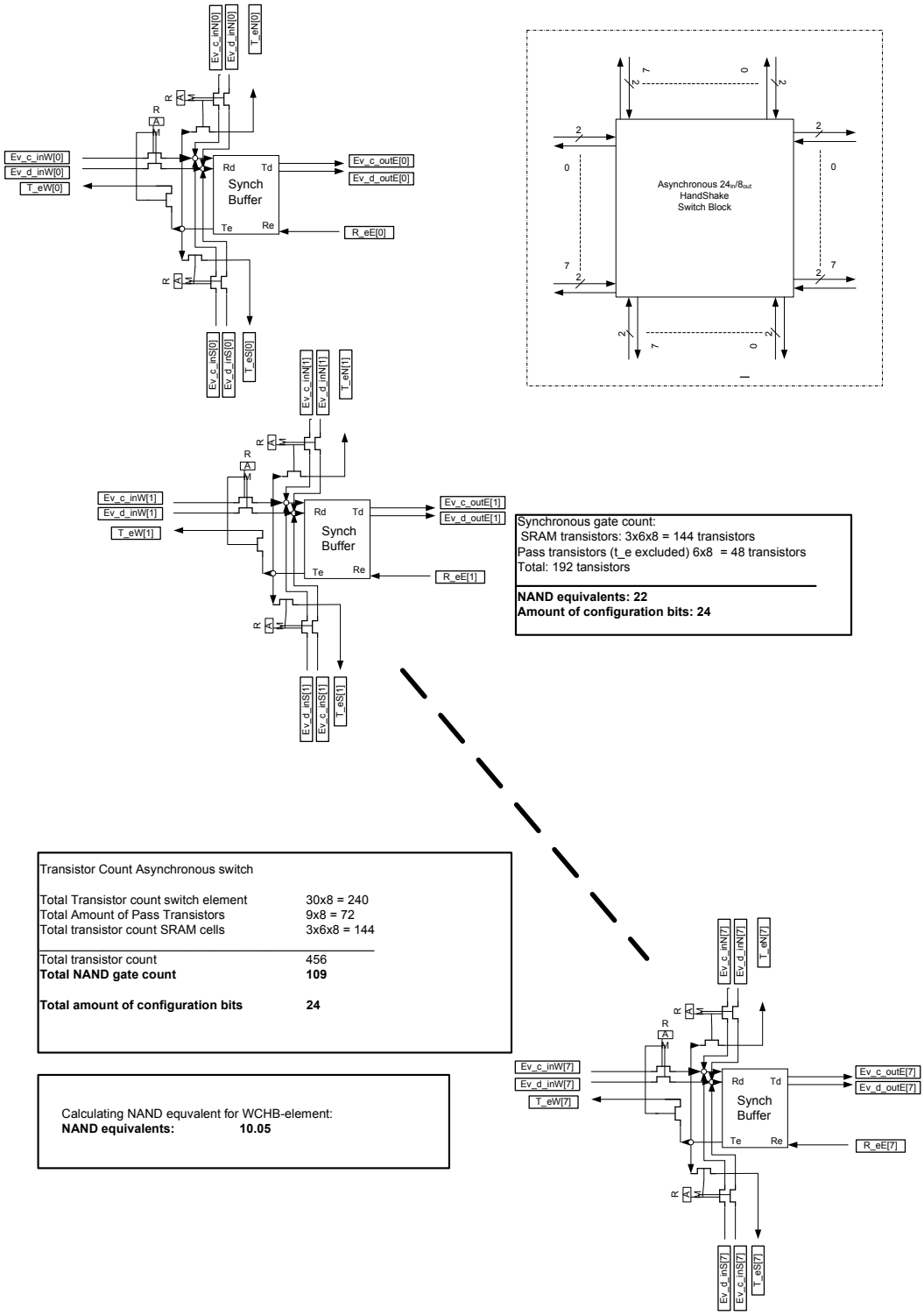
A 16.1-9: Ouput Copy Element



Equivalent gate count:	
C-elements:	12NAND
2:1 MUXes	8NAND
SRAM:	2.67 NAND
Total NAND Equivalents:	**22.67NAND**
Configuration bits:	**4**

16.2 Custom switch design

A 16.2-1: Uni-directional 24 channel input/8channel output



Synchronous gate count:
SRAM transistors: 3x6x8 = 144 transistors
Pass transistors (t_e excluded) 6x8 = 48 transistors
Total: 192 tansistors

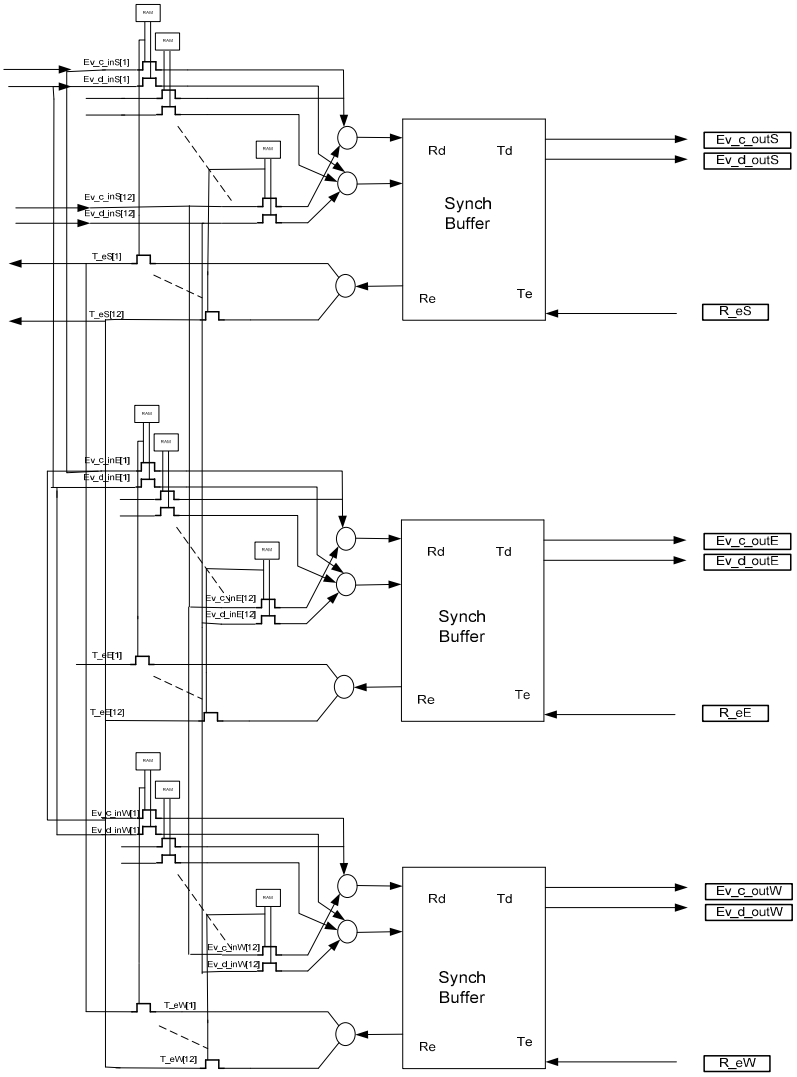
NAND equivalents: 22
Amount of configuration bits: 24

Transistor Count Asynchronous switch

Total Transistor count switch element	30x8 = 240
Total Amount of Pass Transistors	9x8 = 72
Total transistor count SRAM cells	3x6x8 = 144
Total transistor count	456
Total NAND gate count	**109**
Total amount of configuration bits	**24**

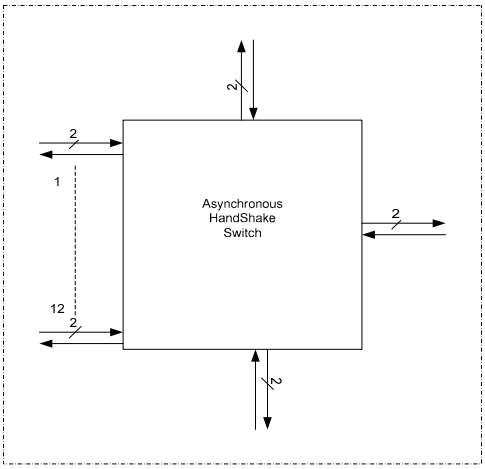
Calculating NAND equvalent for WCHB-element:
NAND equivalents: 10.05

A 16.2-2: Uni-directional 12 channel input/ 3x1 channel output

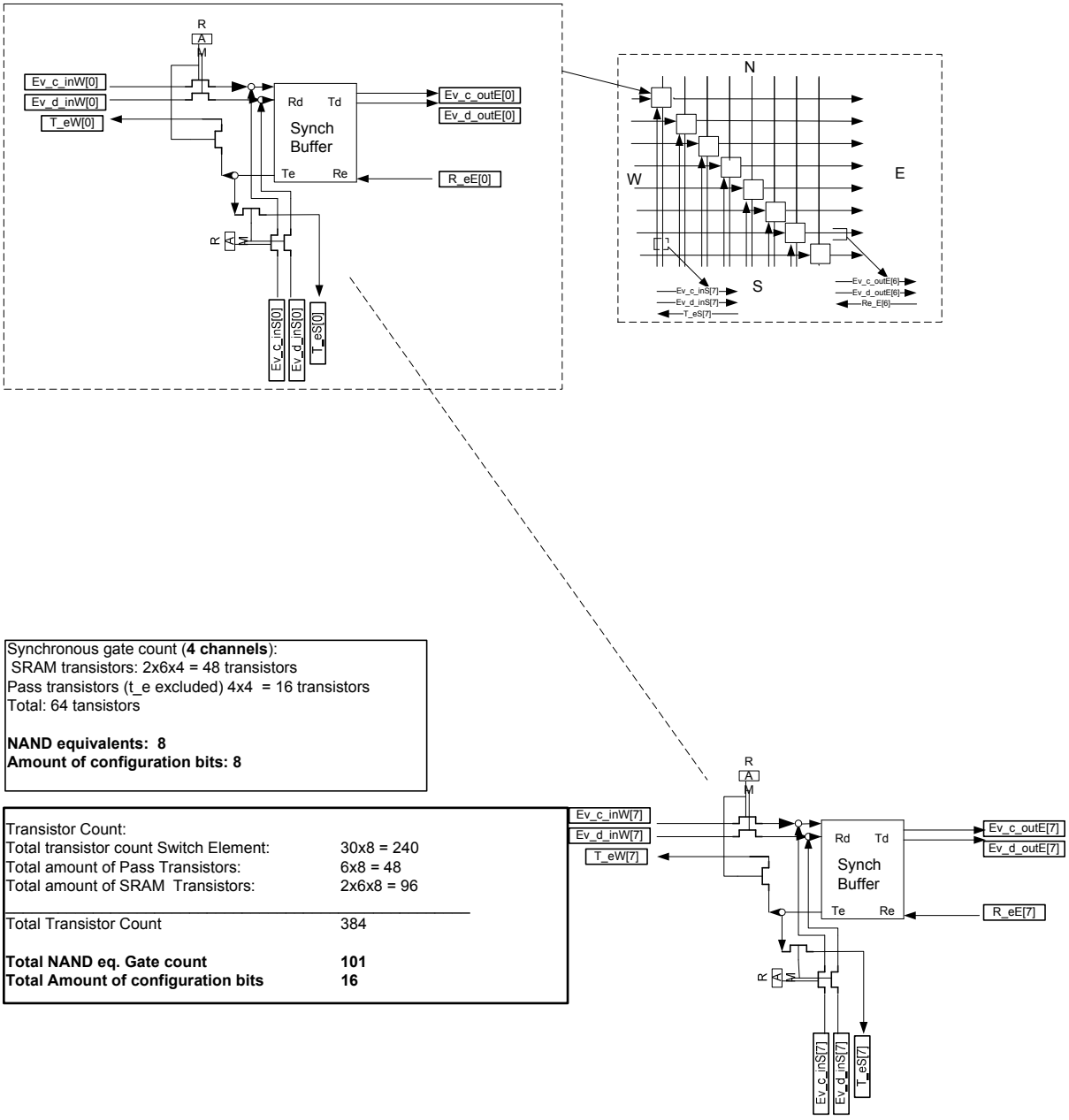


Transistor Count:		
Total transistor count Switch Element:	30x3 = 90	
Total amount of Pass Transistors:	36x3 = 108	
Total amount of SRAM Transistors:	12x6x3 =216	
Total Transistor Count	414	
Total NAND gate count		**68**
Total Amount of configuration bits		**36**

Synchronous gate count: 306 transistors
35 NAND equivalents



A 16.2-3: Uni-directional 16 channel input/8 channel output

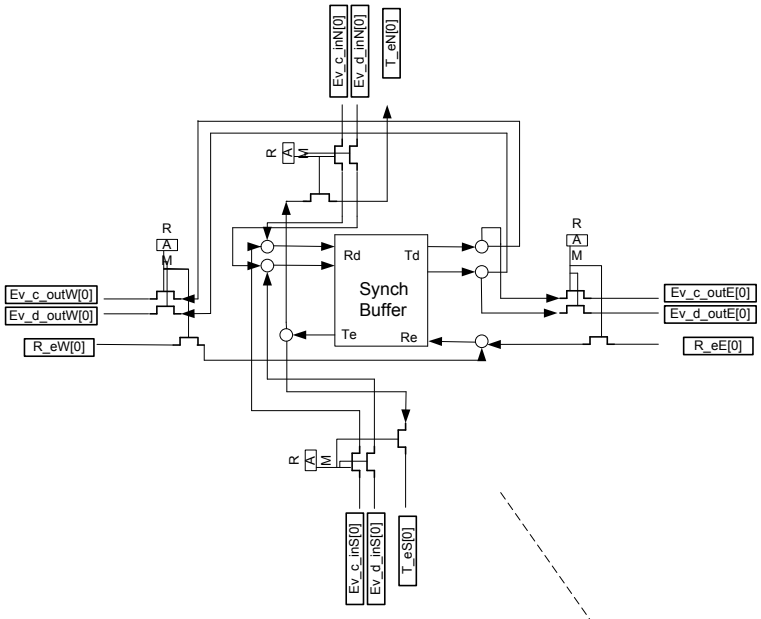


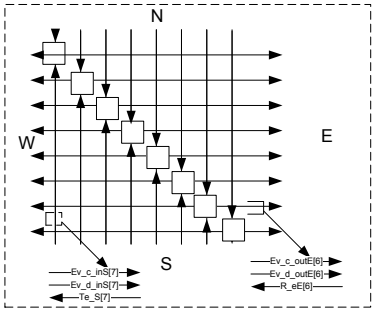
Synchronous gate count (**4 channels**):
SRAM transistors: 2x6x4 = 48 transistors
Pass transistors (t_e excluded) 4x4 = 16 transistors
Total: 64 tansistors

NAND equivalents: 8
Amount of configuration bits: 8

Transistor Count:	
Total transistor count Switch Element:	30x8 = 240
Total amount of Pass Transistors:	6x8 = 48
Total amount of SRAM Transistors:	2x6x8 = 96
Total Transistor Count	384
Total NAND eq. Gate count	**101**
Total Amount of configuration bits	**16**

A 16.2-4: Bi-directional 16 channel input/ 16 channel output





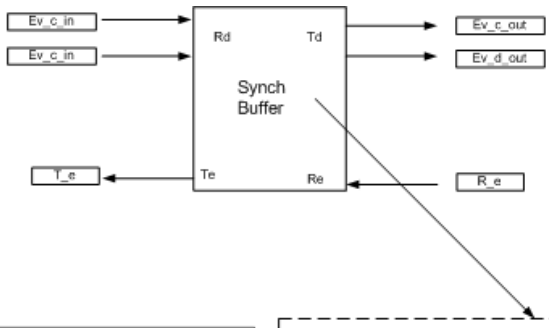
Synchronous gate count:
SRAM transistors: 4x6x8 = 192 transistors
Pass transistors (t_e excluded) 8x8 = 64 transistors
Total: 256 tansistors

NAND equivalents: 29
Amount of configuration bits: 32

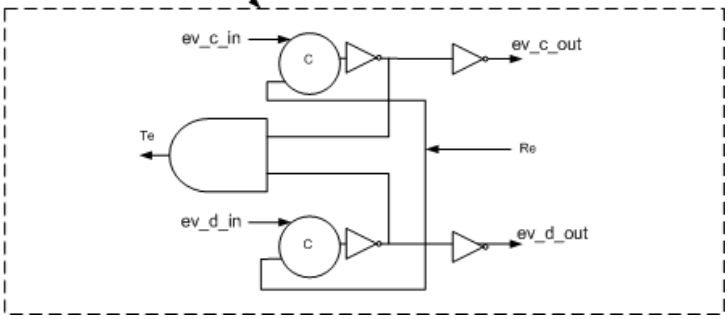
Transistor Count:	
Total transistor count Switch Element:	30x8 = 240
Total amount of Pass Transistors:	12x8 = 96
Total amount of SRAM Transistors:	4x6x8 = 192
Total Transistor Count	528 (Rev 1.1)
Total NAND eq. Gate count	**117**
Total Amount of configuration bits	**32**

7

A 16.2-5: WCHB Design Detail

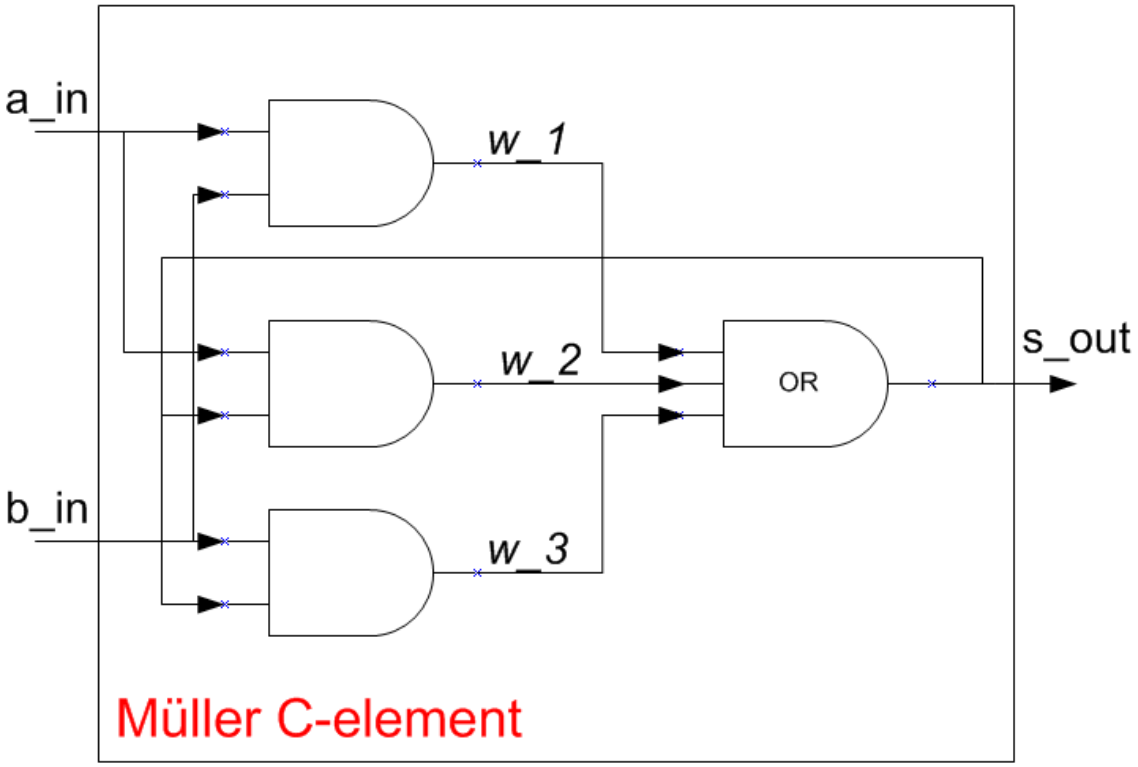


Transistor count WCHB switch element	
Müller-C element transistors:	2x8/10 = 16/20
AND-gate transistors:	6
Inverter transistors	2x2 = 4
Total amount of transistors	28/30

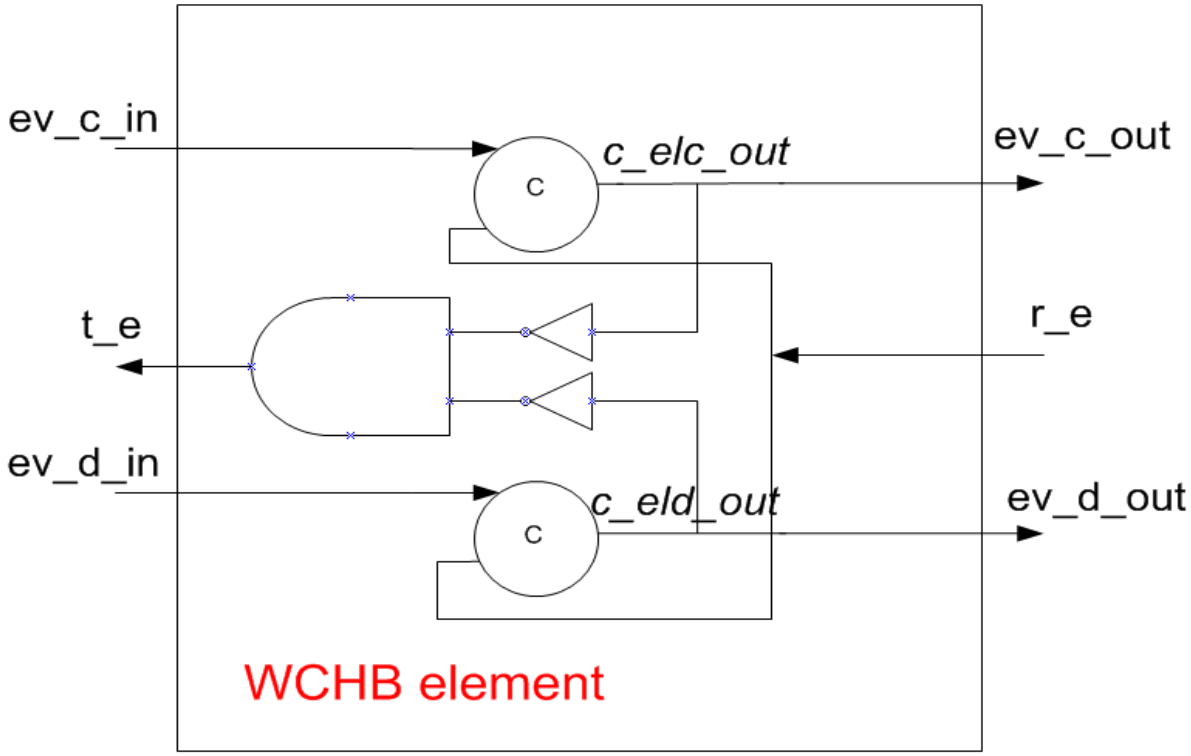


16.3 Verilog Circuit Modules

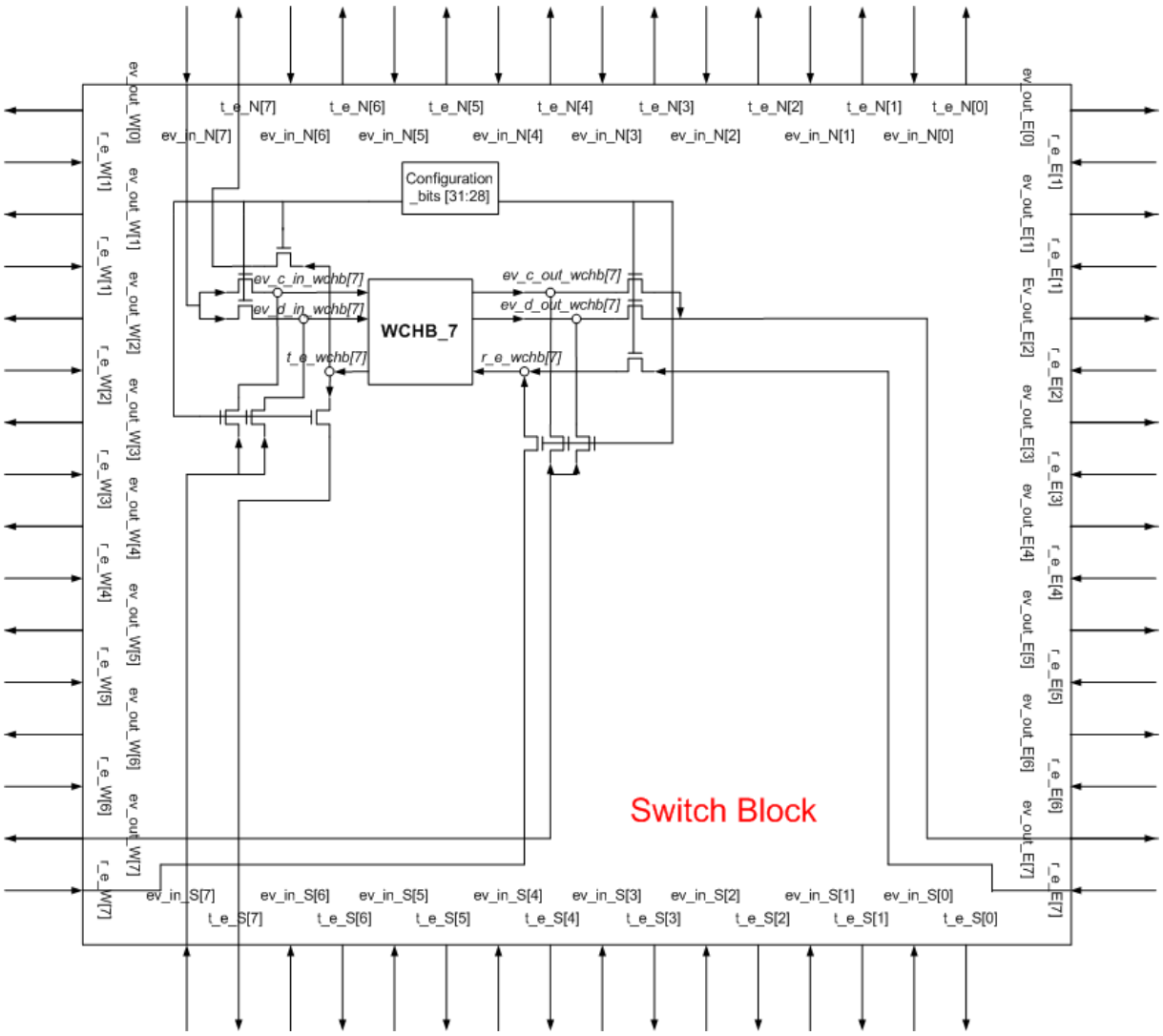
A 16.3-1: Müller C-element



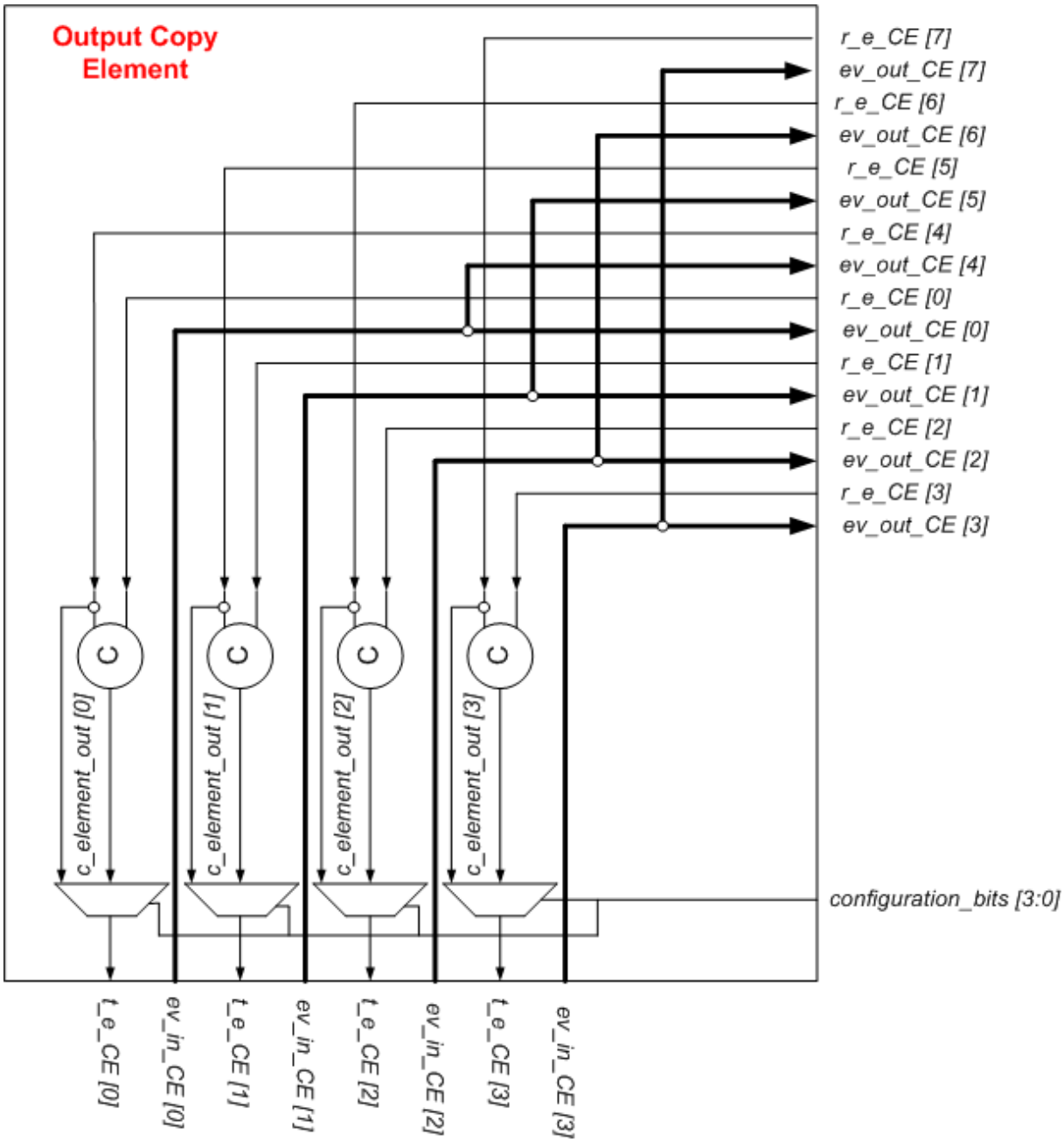
A 16.3-2: WCHB element



A 16.3-3: Switch Block architectural overview



A 16.3-4: Output Copy Element



16.4 CPLD Area Estimation

As mentioned in section 5.4 and chapter 0 the CPLD architecture targeted for implementation is the Atmel ATF1508RE architecture. To estimate the equivalent area this CPLD architecture there are some important aspects that must be clarified, where the important questions are: How is the switch matrix connected to input / output terminals, and how many connections are there in the programmable AND-plane? Area occupied by these programmable connections is significant, and therefore relevant literature was investigated to answer these questions. Some of the assumptions taken in these estimations may not be in accord with the architecture in question, but they are taken after the author's best knowledge.

A 16.4-1: CPLD Area Contributors

Designing a crossbar for allocating inputs to a specific number of outputs are important to do in an intelligent way for PLA structures, so that unneeded programmable connections can be removed from the array to save significant area [51]. For the area estimations in this thesis only to versions of crossbar structures are considered: Full crossbars and sparse crossbars. A simple illustration from [51] shows the difference between the crossbar architectures.

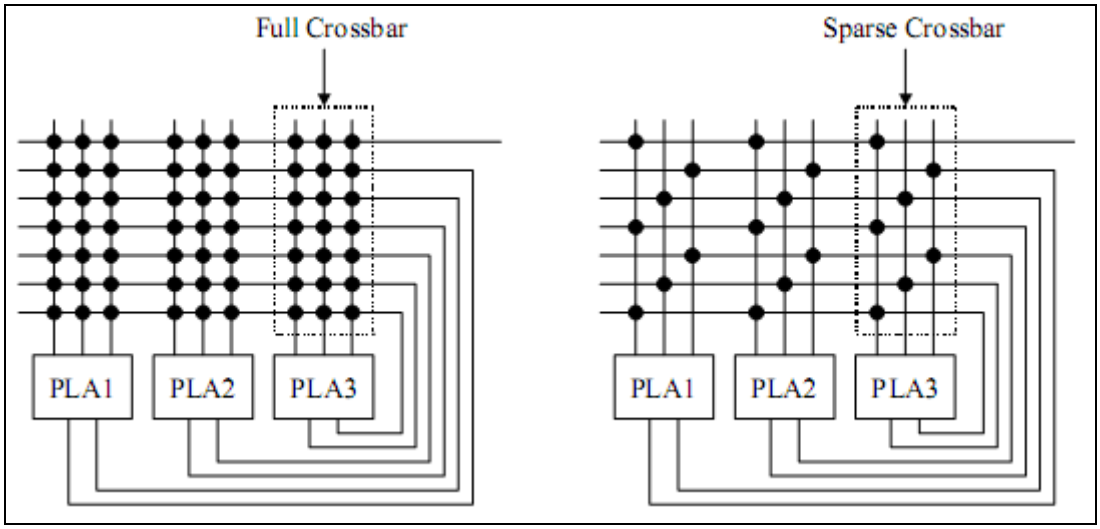


Figure A 16.4-1: CPLD crossbar architectures [51]

As can be depicted in the figure, a large saving of switch area can be obtained by using a sparse crossbar. According to the scientific work provided by Mark Holland in [52] sparse crossbars can be reduced to about 40% of the size of the corresponding full crossbar. Applying this factor to the switch matrix featured by the ATF1508RE architecture, an estimated number of switches and SRAM cells needed for a specific number of inputs and outputs can be determined. According to [7] all outputs from the switch matrix must be available for the programmable AND-plane, and therefore a full crossbar is used to estimate programmable switches and SRAM cells for this region. Each macrocell has a fixed amount of internal logic, which is estimated by counting the number of different elements and apply area according to the Atmel 35k9 standard cell library [10].

A 16.4-2: CERN Area Estimation

The architecture used for estimation is shown in figure 8.3. For detail on the size of SRAM standard cells used for estimation, the reader is referred to appendix B.2.
The switch matrix is included in each CPLD MCB to provide the correct input signals to each connected macrocell. Assuming a sparse crossbar structure, a 56 input 10 output crossbar will consume $56 \cdot 10 = 560$ pass transistors and SRAM cells to control these transistors. Using the sparse crossbar scaling factor of 0.6 as obtained from [52], the total amount of pass transistors and SRAM cells are $560 - (560 \cdot 0.6) = 224$. Counting pass transistors as SRAM equivalents the crossbar's equivalent gate count is 175 NAND. One sparse input switch matrix will be needed for each CPLD MCB, and support an arbitrary number of macrocells.

Each macrocell involves five 5-input AND gates connected to the PTMUX, each connected to a programmable AND-plane. The programmable AND-plane is considered a full crossbar for maximum amount of programmable connections. As shown in figure 8.3 28 wires are connected to the programmable AND-plane, needing $28 \cdot 5 \cdot 5 = 700$ pass transistors and 700 SRAM cells. Measured in NAND-equivalents the total count is 586 NAND equivalents.

To calculate logic area the architectural drawings from [7] are used directly, mapping the logic elements to NAND equivalents for the 35k9 standard cell library. In appendix B.3 the detailed estimation is shown in table B-6. As can be depicted in table 8-1 the final result is 130 NAND gates for internal macrocell logic.

Since the exact CPLD macrocell construction is not known, these estimates are meant as guidelines only and should not be counted as exact under any circumstances.

16.5 Synthesis Results

C-element:

```
Mapping Summary:
Total  LUTs: 1 (0%)
```

WCHB-element:

```
Mapping Summary:
Total  LUTs: 5 (0%)
```

Switch Block:

```
Mapping Summary:
Total  LUTs: 136 (0%)
```

AGERN network:

```
Mapping Summary:
Total  LUTs: 1048 (6%)
```

Copy-element:

```
Mapping Summary:
Total  LUTs: 8 (0%)
```

ALFERN network:

```
Mapping Summary:
Total  LUTs: 234 (1%)
```

AESRN (Test example) :

```
Mapping Summary:
Total  LUTs: 1263 (8%)
```

16.6 Simulation Sequences

A 16.6-1: C-element simulation (Figure A 16.6-1)

To verify the behavior of the Verilog c-element model a test scenario was constructed. Using a testbench model similar to the one described in section 10.1.1, with a testbench clock to synchronize test signals, a series of different input combinations was enforced on the C-element.
For this simulation the negative clk edge is used to issue test signals, but since the C-element is asynchronous the choice of clock edge does not matter for test purposes. Signals highlighted in bold face are marked in Figure A 16.6-1.

clk	1	**2**	3	**4**	5	6	7	8	**9**	**10**	11	12
a_in	0	**0**	1	**1**	0	0	1	1	**0**	**0**	1	1
b_in	0	**1**	0	**1**	0	1	0	1	**1**	**0**	1	0
s_out	0	**0**	0	**1**	0	0	0	1	**1**	**0**	1	1

Table A 1: C-element test sequence

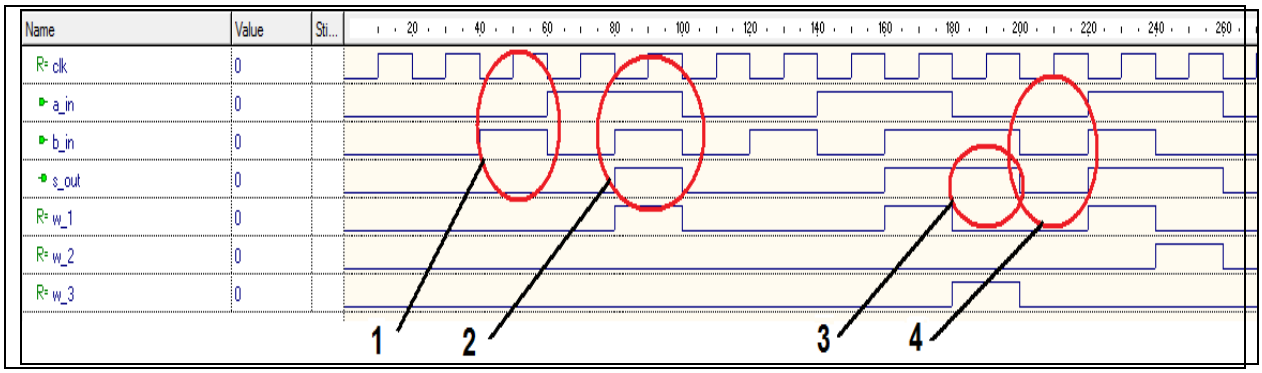


Figure A 16.6-1: C-element test sequence

- **Point 1:** First test sequence after reset condition. Since only one of the inputs are high, the output remains “0”.
- **Point 2:** Both inputs are high, setting the internal signal *w_2* to “1”. Since the output *s_out* is an OR operation of all internal signals, the “1” propagates to the output.
- **Point 3:** Only input *b_in* is “1”, but since the last output of *s_out* was “1” internal signals *w_3* is “1”, setting the new output of *s_out* to “1”. This sequence shows the stateholding capabilities of the C-element implementation.
- **Point 4:** Both inputs are “0”, representing a reset condition for the C-element. The output is also set to “0”.

A 16.6-2: WCHB simulation (Figure A 16.6-2 and Figure A 16.6-3)

Since the WCHB element is a crucial element in all the constructed switch blocks, it is very important to understand the behavior of a WCHBs handshake protocol. The 4-PDR protocol used for all asynchronous signal transfer in this thesis must be executed with a specific token sequence to ensure correct handshaking. In this simulation the rising edge of the clock is used to synchronize test signals. The first test scenario enforces the 4-PDR handshake protocol, to maintain correct pipeline functionality. To see erroneous behavior of the WCHB element, a test sequence applying an incorrect handshake protocol is conducted for test scenario 2.

Scenario 1:
The applied test sequence can be depicted in Table A 2, where columns in bold face are marked in Figure A 16.6-2.

clk	**1**	3	**5**	7	9
ev_c_in	**1**	0	**1**	0	1
ev_d_in	**0**	1	**1**	0	0

Table A 2: WCHB test sequence for scenario 1

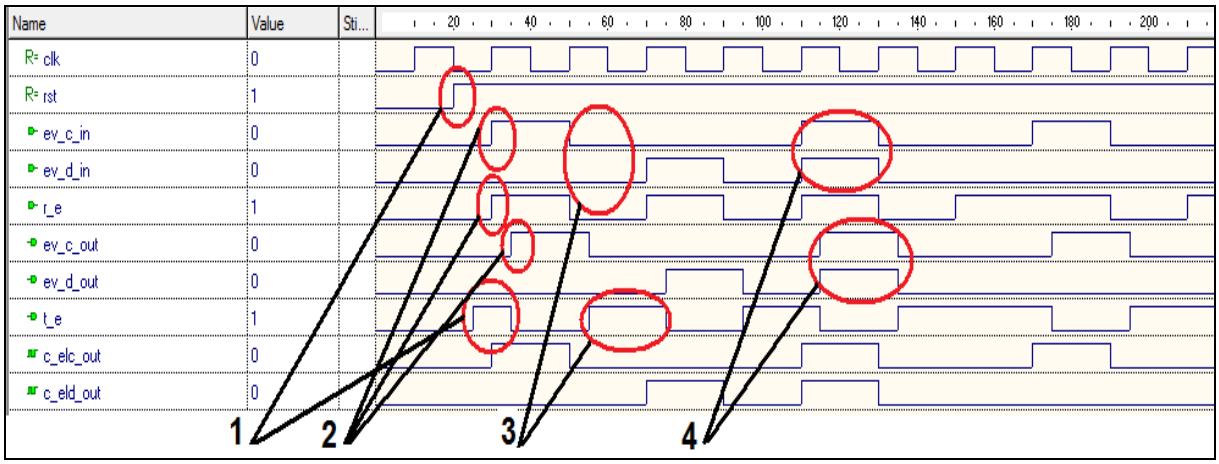


Figure A 16.6-2: Correct WCHB simulation sequence

- **Point 1:** After reset is set high, the WCHB sends its *t_e* signal to the connected testbench to signal that it is ready for receiving data. *T_e* is always set with an internal logic delay used in simulations of 5ns.
- **Point 2:** On the next rising *clk* edge the testbench senses that *t_e* from the WCHB is "1", and issues an event on *ev_c_in*. *T_e* is set low from the WCHB to indicate data reception, and an event is sent on *ev_c_out* after 5ns delay.
- **Point 3:** The event is detected by the testbench on the next rising *clk* edge, and the proper zero spacer, or reset token, is issued by the testbench. When the WCHB receives the spacer, the handshake sequence is complete, and *t_e* is set to "1" to indicate that a new data token can be received.
- **Point 4:** Both *ev_c_in* and *ev_d_in* set to "1" is an invalid input according to 4-PDR signaling protocol. However it is only invalid in the sense that decoding of the inputs are not valid, and will not cause a circuit failure. Both outputs *ev_c_out* and *ev_d_out* are set to "1", and is handled by the testbench as a valid event to ensure proper reset conditions.

Scenario 2:

This scenario follows the same test sequence as scenario 1, but will try to omit the reset token after *clk* = 5 in Table A 2. This violation of the 4-PDR handshake convention will cause the pipeline to jam, until a reset token is produced. An example is illustrated in Figure A 16.6-3.

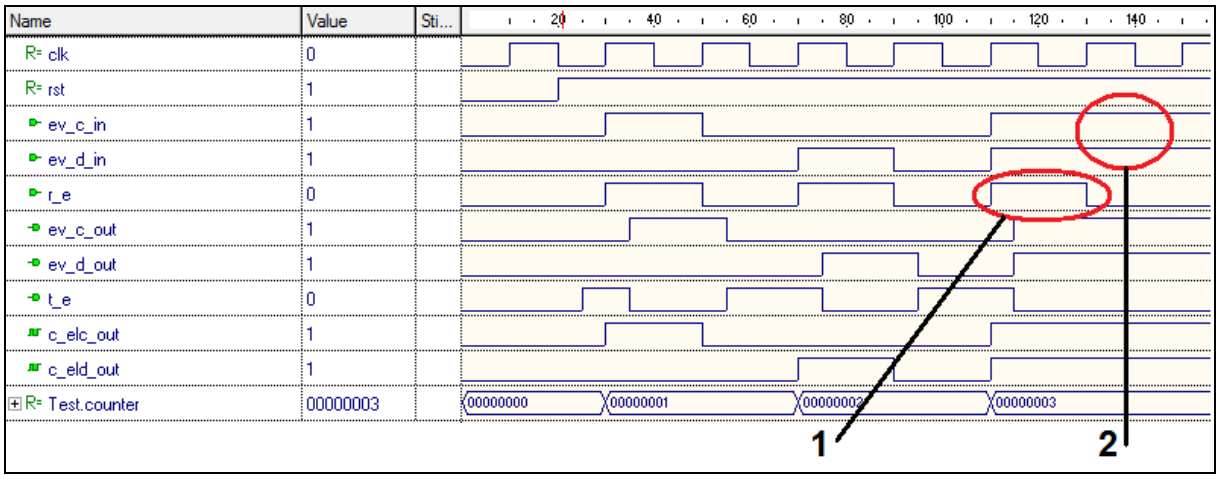


Figure A 16.6-3: WCHB handshake sequence with errors

- **Point 1:** The output is received by the testbench, but no confirming reset token is produced in return.
- **Point 2:** The absence of a reset token leaves the WCHB inputs high, and therefore the *t_e* signal will always be "0" to indicate a busy state. WCHB outputs are also "1", making *r_e* received from the testbench "0". This situation is completely locked since both instances think that the other instance is computing, and by definition busy.

If any signal is not set correctly during a handshake, no matter if it is *t_e*, *r_e* or some of the input / output signals, the same locked situation will occur.

A 16.6-3: AGERN handshake simulation (Figure A 16.6-4 and Figure A 16.6-5)

To fully understand how the switch blocks distribute signals, an expanded handshake description is given in the coming example. Based on the constructed scenario described in section 10.4, the reader is referred to this section for scenario details and descriptions. A delay of 5ns is used for WCHBs inside switch blocks, too visualize the signal propagation.

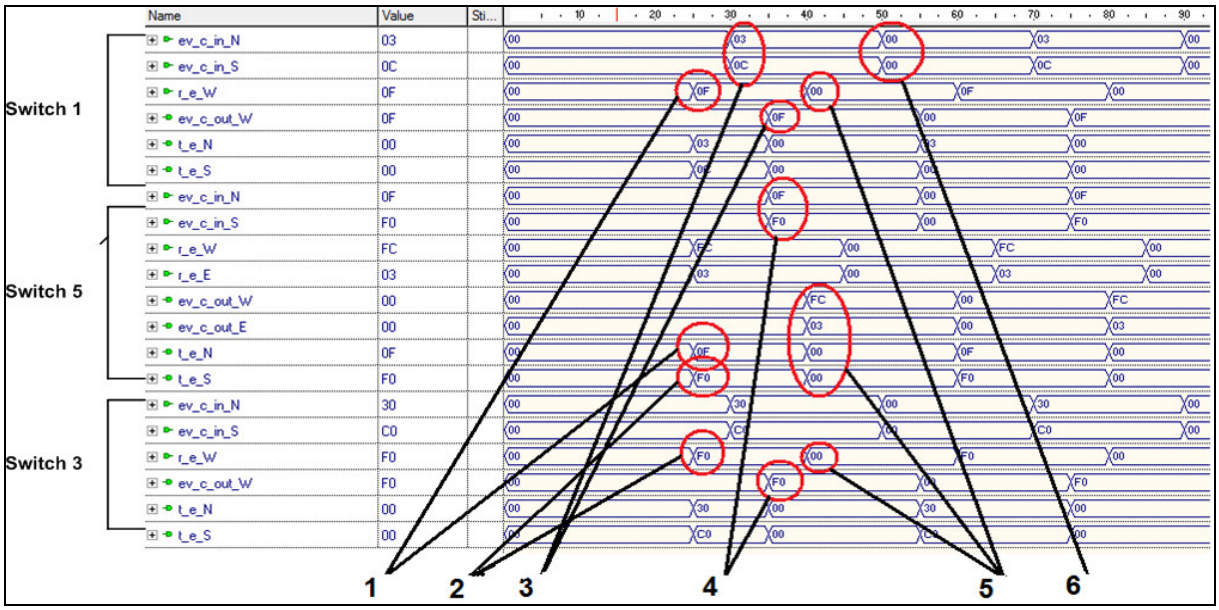


Figure A 16.6-4: AGERN handshake sequence between switches 1,3 and 5

- **Point 1:** The northern input channels of switch 5 are connected to the western output channels of switch 1. Therefore *t_e_N* from switch 5 is equivalent to *r_e_W* on switch 1.
- **Point 2:** Switch 3 connects its western output channels to switch 5's southern input channels. The connection is reflected through *t_e_S* on switch 5, and *r_e_W* on switch 3.
- **Point 3:** Switch 1 receives inputs from Port C on northern input channels 0 and 1, and from Port D on southern input channels 2 and 3. These propagate to the western output channels 0-3, and are sent to switch 5.
- **Point 4:** Switch 5 receives the output from switch 1 on it northern inputs, and the western outputs from switch 3 on it southern input. Note that for both switch 1 and switch 3 the corresponding *t_e* channels are set to "0" a 5 ns delay after the input is received.
- **Point 5:** Outputs are issued from switch 5 on both northern and southern outputs, on output channels 0-7 utilizing the full capacity of the switch block. T_e_N and t_e_S are set to "0" accordingly, and received on *r_e_W* of both switch 1 and switch 3.
- **Point 6:** The first reset tokens arrive one *clk* cycle after the data tokens, in the test set to 20ns. The reset token propagates the pipeline in the same manner as the data token, resetting all switch blocks for the next transmission.

While Figure A 16.6-4 shows a detailed handshake correspondence, Figure A 16.6-5 shows how AGERN outputs are delayed in different channels according to how many switch points they encounter on their path through the AGERN.
Figure 10.13 (left) shows the destined path for each event, and the delay is reflected in Figure A 16.6-5.

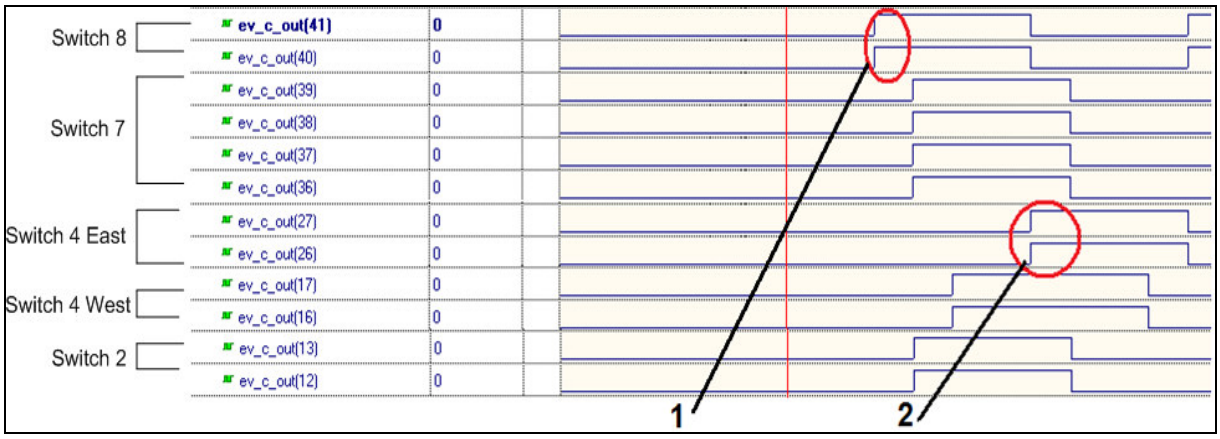
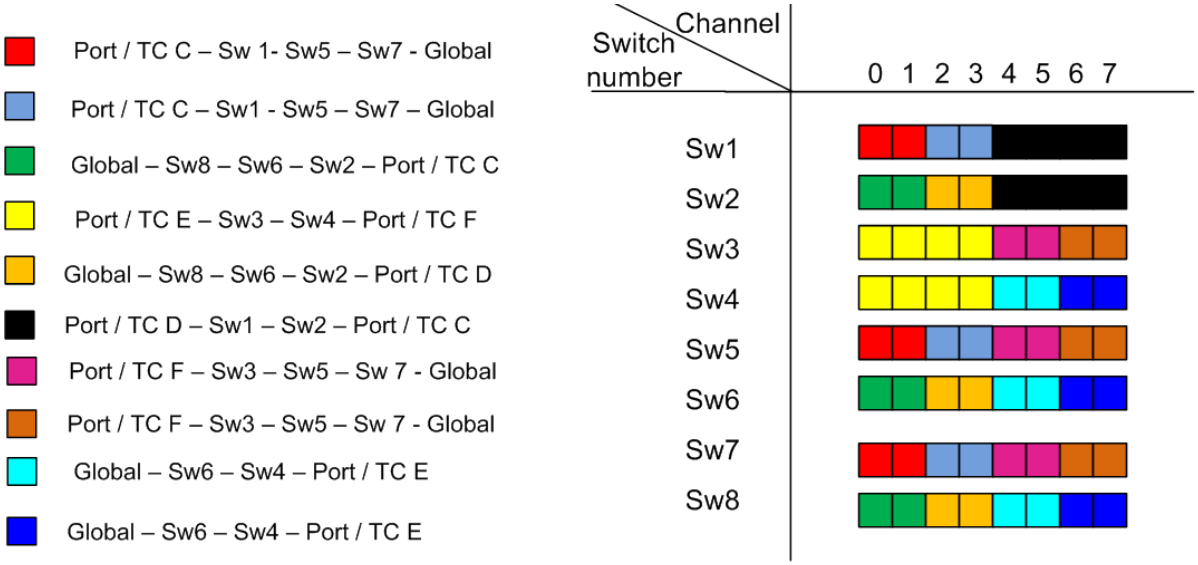


Figure A 16.6-5: AGERN output delay

- **Point 1:** Output channels from switch 8 have only encountered the equivalent delay of two switch blocks, and are produced 10ns after the input events reached the AGERN.
- **Point 2:** Switch 4 delivers outputs that have traveled through 6 switches, and encountered an equivalent delay of 30ns, on its eastern output. This is 20ns later than the output from switch 8.

16.7 AGERN 24 Event Distribution Scenario

A 16.7-1: AGERN 24 Event Distribution Scenario



The top figure depicts a possible event distribution scenario for 24 concurrent events on the AGERN. Below is the configuration used to program each switch block, in accord with the example from section 10.4. The same configuration is also available for demonstration as one of the test setups on the included CD.

[31:28] – [3:0]	Conf. bit
01000100010001000001000100010001	Sw1
00010001000100010100010001000100	Sw2
00100010001000100100010001000100	Sw3
00100010001000101000100010001000	Sw4
00100010001000100001000100010001	Sw5
00100010001000100100010001000100	Sw6
00010001000100010001000100010001	Sw7
01000100010001000100010001000100	Sw8

16.8 Appendix C – Scientific Paper

C 16.8-1: Scientific Paper - "Event Control and Programming for Microprocessor Peripheral Systems"

Event Control and Programming for Microprocessor Peripheral Systems

Rune André Bjørnerud, Morten W. Lund , Kjetil Svarstad

***Abstract*—To handle communication between on-chip peripherals without interference from CPU, DMA or interrupt resources, the AVR XMEGA microcontroller introduces a peripheral resouce known as the Event System. The Event System currently implemented on the AVR XMEGA offers limited resources for logical event computation, and can basically be considered an advanced routing facility for I/O and peripipheral signals. This work proposes a new Event System solution, trying to enhance routing flexibility by offering a programmable asynchronous interconnect topology with pipelined switches, and increased computational power by using asynchronous LUTs to handle logical event computations.**

I. Introduction

The AVR XMEGA Event System [4] is designed to operate in a paripheral rich environment, distributing events between peripherals with limited involvement of CPU resources using the Event Routing Network (ERN) as shown in figure 1. Althoug a power saving resource for the AVR XMEGA, the Event System is limited to 8 global event channels on the ERN and offers no functionality for logical operations like AND or OR on distributed events. This limits the computational power of the current Event System.

A master's thesis [11] adresses these limitations by proposing an Asynchronous Event System with a pipelined interconnect network known as the Asynchronous Event System Routing Network (AESRN). To ensure full routing flexibility the AESRN interconnect structure is based on a pseudo-hierarchical FPGA topology using custom designed SRAM programmable switches to route events propagating the pipeline. In [16],[17] Teifel et. Al presentes an asynchronous pipelined FPGA interconnect structure, where fine-grained pipelineing is employed in each switch block to ensure high througput. Hierarchical FPGA topologies as discussed by Aggarwal et. Al in [1] promise reduced cost in terms of switching area compared to segmented routing topologies. Inspired by elements from these papers, the Asynchronous Event System proposes a custom designed FPGA-inspired topology designed to fit a microcontroller implementation. Applying hierarchical design elements the new routing topology also makes it possible to utilize local communication between selcted modules for minimal event latency.

Asynchronous computation is ensured by LUTs implemented in a dual-rail encoded handshake environment, where two implementation methods is discussed in [16] and [2]. A modified version of the logic cell discussed in [2] is designed for the AESRN to provide full asynchronous computation for events.

Morten W. Lund with Atmel Norway

This paper is structured as follows: Section II will provide background information on the event notation developed by Atmel, while section III considers asynchronous logic in general especially emphasizing asynchronous pipeline. Section IV presents the proposed Asynchronous Event System solution, which will be further analyzed with respect to performance and area in section V. Section VI concludes this paper and outlines further work.

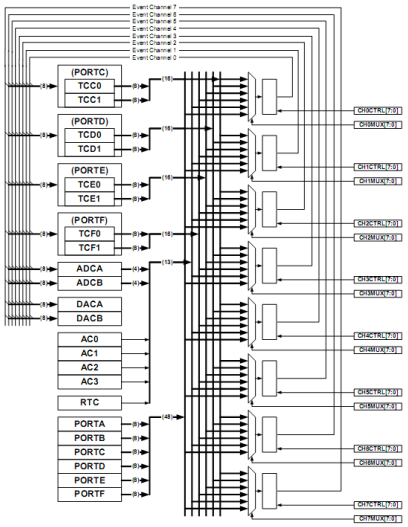


Figure 1. Original Event Routing Network [4]

II. The AVR XMEGA Event System

Atmel has developed a unique event notation and definition for the Event System, which will be used to describe event channels and event flow in the Asynchronous Event System. The presented notation i strictly applicable for the Event System used in the AVR XMEGA, and should not be confused with the general event notation presented in literature.

An *event* indicates that a change of state has occured in a peripheral [4]. The two main event types used in the Event System is the *data* event and the *signaling* event. A signaling event indicates that a peripheral state change has occured, while a data event can hold additional information. The event channel consist of two event wires *ev_c* and *ev_d*. The combined bit pattern represented by these two wires defines the event type received by a peripheral. Each peripheral can have a different interpretation of a specific event, and depending on the interpretation of the event different actions can be triggerd in different peripherals. More details on events are given in [4],[11].

III. Asynchronous Circuit Design

The asynchronous design paradigm differs fundamentally from synchronous design, and a good overview of asynchronous design is given by Scott Hauck in [6] and Jens Sparsøet. Al in [15]. Some mentioned benefits over synchronous systems include: Better adaptability to new and more aggressive process technologies due to delay insensitivety properties [16],[14],[15], average-case timing assumptions compared to synchronous systems which must consider worst-case timing performance to meet clock skew demands and global timing issues [6], lower power consumption due to event driven consummation [16] and zero power consumption when in standby [15].

Especially the low-power feature is interesting for a proposed Event System solution, but exploiting the fact that events are asynchronous by nature[6] could also yield performance benefits over a synchronous Event System. Some well accepted drawbacks compared to synchronous circuits are poor EDA- and CAD tools support [14] and larger circuits due to redundant handshake elements [15]. One of the biggest challenges connected to asynchronous circuits are concerning different types of hazards. Static and dynamic hazards are covered in [6] and [15], while metastability issues are covered in [7]. Metastability issues are particularly important to be aware of considering an Asynchronous Event System, because it will interface a totally synchronous environment.

A. Asynchronous QDI-pipelines

Since asynchronous timing is non-clock dependent several timing models exist, but the proposed AESRN only considers the Quasi-Delay Insensitive (QDI) timing model[6],[14],[15]. With the absence of clock-driven timing, some other means of synchronization must be implemented. The Müller C-element [10],[18] is commonly used for this purpose because of it's stateholding property. In asynchronous pipelines handshake protocls are used to obtain synchronization for data elements, called *tokens*, propagating the pipeline. According to [8] the four-phased bundled data (4-BD) protocol or four-phased dual-rail (4-PDR) protocol is usually considered for asynchronous circuits. The 4-PDR protocol is used in several pipeline structures [16],[17],[19], with the benefits of meeting the QDI-assumptions and thereby offering a natural robustness to delay variations [15]. Dual-rail encoding operates with two data rails where the request for data is encoded in the combination of the two rails, and the *acknowledge* is sent on a separate wire. Because the request for data is encoded in the received token, no timing violations between *request and acknowledge* can occur as is the case for the 4-BD protocol [8]. More information on the choice of handshake protocol is featured in [11].

Fine-grained pipeline structures rely on token buffer elements in the switch blocks to maintain high throughput [16],and two common implementations for this purpose is the Weak-Condition Half-Buffer (WCHB) and PreCharge Half-Buffer (PCHB) [16],[14],[19]. One possible WCHB implementation using an *enable* signal is provided in figure 2. WCHBs are the smallest and fastest QDI buffer with the shortest cycle time [5], but a mixture of both PCHBs and WCHBs in the pipeline can lead to more optimal throughput [19],[20]. For a more in depth discussion on this topic the reader is referred to [11],[20].

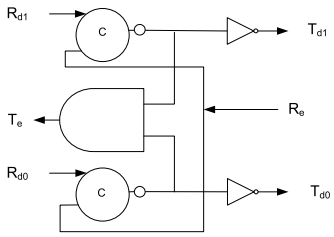


Figure 2. A possible WCHB element implementation [16]

IV. The Asynchronous Event System

To utilize the fact that selected Port and Timer/Counter (TC) peripherals are closely interconnected in the AVR XMEGA [4], the Asynchronous Event System proposes a pseudo-hierarchical topology to utilize the frequent communication between these modules. Partitioned into an Asynchronous Global Event Routing Network (AGERN) and Asynchronous Local Functional Event Routing Network (ALFERN) both local and global communication can occur concurrently with minimal latency for local event distribution between Port / TC peripherals. To make the event flow more intuitve to predict a two-way pipeline structure is applied, meaning that the event channels are partitioned in two main event flow directions. This can be depicted in figure 3, where the Port / I/O side of the developed AGERN is illustrated.

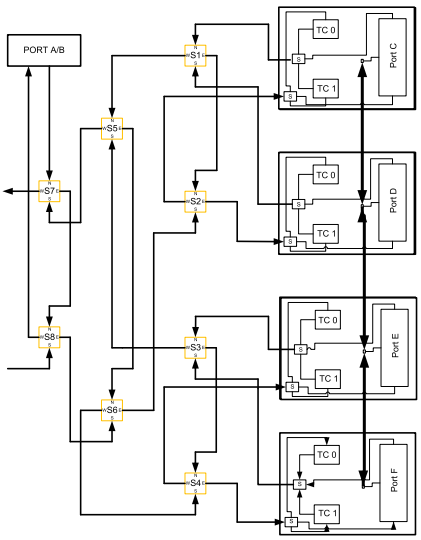


Figure 3. Asynchronous Global Event Routing Network Port / I/O side [11]

Each switch block is custom designed to fit the two-way routing scheme, with a capacity of 8 event channels in each direction [11]. Each switch block is constructed by multiple switch points, each switch point providing token buffer capacity to maintain a fine-grained pipeline model [16]. Utilizing the fact that an Event Channel in the original AVR

XMEGA Event System [4] is constructed by the two wires *ev_c* and *ev_d* the 4-PDR protocol is applied for the pipeline structure to minimize wire overhead. One WCHB is used in each switch point to provide token buffering, emphasizing fast token echange and small implementation area. Another reason to use WCHBs is that the AESRN does not include computation elements in the pipeline, limiting the need for more advanced buffer structures like PCHBs.

Figure 4 illustrates a switch point inside the switch block. SRAM cells hold the configuration for each switch point, where one SRAM cell control both the event data rails and the corresponding *enable* wire for each Event Channel. As in [16] an *enable* wire which represents the logical inverse of the *acknowledge* is used to yield smaller WCHB circuits. The switch point illustrated in figure 4 needs 4 bits for configuration and the whole switch block i configured with 32 bits.

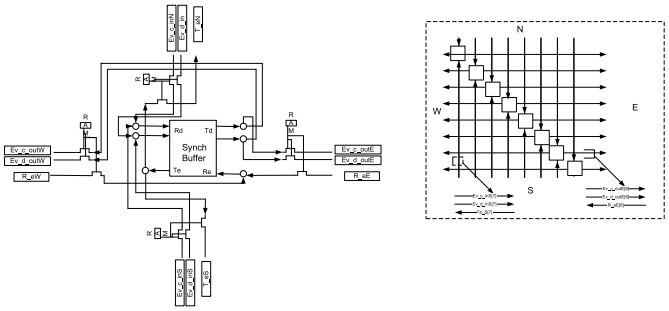


Figure 4. Custom asynchronous switch point and switch block layout

To implement LUT functionality the proposed solution targets at using buffer registers implemented in each TC module [4], and use these to hold LUT configurations when not used for buffering. By combining a 16:1 MUX controlled by asynchronous inputs and surrounded by handshake logic, interfacing to the dual-rail protocol applied in the AESRN pipeline is trivial. One register supporting LUT functionality with handshake logic and routing is denoted a Timer Counter Functional Block (TCFB). One Port / TC constallation can include 6 TCFBs if all buffer registers are used for LUT functionality, giving a total capacity of 24 TCFBs with one 4-LUTs each. More details on implementation and issues are given in [11].

V. Simulation and Performance Results

A. Verification of the AESRN

Some papers emphasize that the lacking EDA- and CAD tool support for asynchronous designs yields unoptimized disigns [6],[14], and how handmapped asynchronous standard cell libraries [10] or altering of the mapping software [13] are possible solutions. These well known issues put physical implementation beyond the scope of the master's thesis in [11], making verification by simulating selected asynchronous distribution principles employed in the AESRN the chosen method. Simulation of the AESRN pipeline distributing events using both AGERN and ALFERN resources, also including logical computation in the asynchronous LUT environment, was the setting for an applicable test scenario. Sucsessfull simulations showed a capacity of up to 28 concurrent events distributed on the AGERN and ALFERN, and that the applied 4-PDR protocol worked as expected in the AESRN pipeline. The simulations confirm the proposed Asynchronous Event System model, but does not verify if a physical implementation is possible. This is partly due to the mapping problems mentioned above, and partly because run-time configuration of the routing network by FLASH memory is a physical implementation issue not verifyable by simulation. More details on these issues are presented in [11].

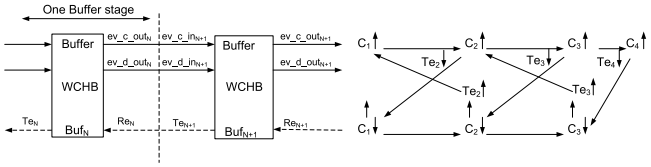


Figure 5. AESRN pipeline-model and corresponding dependency graph

B. Performance Evaluation of the AESRN Pipeline

To analyze the AESRN pipeline performance, a method described by Yahya et. Al in [19] and [20] applicable for QDI-pipelines are modified to fit analysis of the AESRN pipeline. Relying on analysis of the pipeline dependency graph, it is possible to derive equations for the Total Cycle Time (TCT) for each pipeline stage, which is defined as the time each pipeline stage needs to process a set of one *data* and one *reset* token [19]. Figure 5 shows the AESRN pipeline model and associated dependency graph. Since the AESRN pipeline includes no computational elements directly in the token distribution path, the pipeline model depcited in figure 5 represents a simplified model of the one presented in [19].

Delay contributor	Delay 90nm	Delay 130nm	Delay 180nm	Delay 250nm	Delay 350nm	Delay 600nm
τ_{NAND}	50ps [41]	70ps	100ps	142ps	202ps	290ps
τ_{INV} [46]	20ps	33ps	47ps	67ps	95ps	165ps
$\tau_{AND} = \tau_{NAND} + \tau_{INV}$	70ps	103ps	147ps	209ps	297ps	455ps
$\tau_{C\text{-}el}$	100ps	140ps	200ps	284ps	404ps	580ps
$\tau_{Te} = \tau_{AND} + \tau_{INV}$	90ps	136ps	194ps	276ps	392ps	620ps
$\tau_{C} = \tau_{C\text{-}el}$	100ps	140ps	200ps	284ps	404ps	580ps
$TCTn_{WCHB}$	600ps	832ps	1.19ns	1.68ns	2,4ns	3.56ns
Data token frequency	1.66 GHz	1.2 GHz	840 MHz	595 MHz	417 MHz	281 MHz
Event delay $2xTCT_{nWCHB}$	1.2ns	1.66ns	2.38ns	3.24ns	4.8ns	7.12ns
Event frequency	833MHz	600MHz	420MHz	247MHz	208.5MHz	140.5MHz

Figure 6. Estimated WCHB TCT for differnt process technologies [11]

The equations used to calculate the TCT for a WCHB element at stage N in the pipeline is featured in [11].

With no accurate data for element delay, a NAND gate delay obtained by reaserach in [9] using pipeline conditions close to the AERSRN with respect to element load and fan-out was used. Scaling theory predicts a decrease of 30% in element delay for each process generation [3], a number

independently indicated by inverter delay analyses for different process technologies in [12]. The worst case numbers for both NAND-gate delay and scaling factor was used to obtain the AESRN pipeline performance results in figure 6. With applied Dual-Rail encoding in the AESRN, a new event notation presented in [11] now calculates an event from two proceeding data tokens instead of one single *ev_c* and *ev_d* combination. For this reason the event frequency in figure 6 is half of the token frequency.

C. Asynchronous Event System Area Estimation

By Employing asynchronous handshake elements in every pipeline stage of the AESRN, the Asynchronous Event System will have a size disadvantage over the currently implemented Event System. Calculations based on architectural drawings included in [11] priovides an area estimate when the equivalent NAND-gate count is put in the context of approximated area numbers provided by Atmel. All area related numbers released in this section is released with the courtesy of the Atmel Corporation.

Logic area estimations in table I are equivalent NAND gate estimations from architectural drawings based on the Atmel 35k9 in-house standard cell library. Additional area includes extra area for routing resources, synchronizers, glue-logic and extra FLASH memory to support configuration bits. Total NAND equivalents sums up both architecural logic area and additional logic area in terms of NAND equivalents. For more details the reader is referred to [11].

#TCFBs	*Logic NAND eq.*	*Add. logic NAND eq.*	*Total NAND eq.*
0	3837	1718	5555
2	4173	2245	6418
4	4509	2351	6860
8	5181	2446	7627
12	5853	2827	8680
16	6525	3065	9590
24	7865	3548	11413

Table I
ESTIMATED ASYNCHRONOUS EVENT SYSTEM AREA [11]

VI. CONCLUSION

This paper has provided a concept study of a new Asynchronous Event System, and givs a breif introduction to the asynchronous design domain. Important concepts of the AESRN pipeline has been verified through simulations, emphasizing switch programmability and event distribution on both the AGERN and ALFERN networks. Area estimations put a basic AGERN and ALFERN solution without computational elements about 23% larger than the original Event System with its 5555 NAND equivalents, and with 2 included TCFBs about 43% larger with 6418 NAND equivalents. Considering future process technologies this is not an unrealistic alternative in terms of size. Included is also a capacity increase from 8 events in the current Event System, to 28 events with the Asynchronous Event System. Performance improvements range from 6.5 - 1.6 times the original Event System for event distribution 90nm and 350nm technology respectively. Future work will include development of a physical model and tools to handle effective asynchronous synthesis and testing. It must also be focused on how FLASH memory can be used to configure the switch blocks run-time, for fully flexible routing.

REFERENCES

[1] Aditya A. Aggarwal and David M. Lewis. Routing Architectures for Hierarchical Field-Programmable Gate Arrays. *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 475–478, October 1994.

[2] Mehrdad Najibi Atabak Mahram and Hossein Pedram. An Asynchronous FPGA Logic Cell Implementation. *International Conference on Design & Technology of Integrated Systems in Nanoscale Era*, pages 176–179, March 2007.

[3] Shekar Borkar. Design Challenges of Technology Scaling. *IEEE Micro*, 19:23–29, August 1999.

[4] Atmel Corporation. Preliminary AVR XMEGA A Manual. *Available at: http://www.atmel.com/dyn/products/datasheets.asp?family_id=607. Last updated: 04-2009, Cited: 28.05-2009. Doc Rev: G.*

[5] Marcos Ferretti and Peter A. Beerel. Single-track Asynchronous Pipeline Templates Using 1-of-N Encoding. *Proceedings of Automation and Test in Europe Conference and Exhibition*, pages 1008–1015, 2002.

[6] Scott Hauck. Asynchronous Design Methodologies: An Overview. *Proceedings of the IEEE*, 83:69–93, January 1995.

[7] Hans W. Eichel Jens U. Horstmann and Robert L. Coates. Metastability Behavior of CMOS ASIC Flip-Flops in Theory and Test. *IEEE Journal of Solid-State Circuits*, 24:146–157, February 1989.

[8] Lars S. Nielsen and Jens Sparsø. Designing Asynchronous Circuits for Low Power: An IFIR Filter Bank for a Digital Hearing Aid. *Proceedings of the IEEE*, 87:268–281, February 1999.

[9] Yu Cao Paul Friedberg and Ruth Wang et. Al. Modeling Within-Die Spatial Correlation Effects for Process-Design Co-Optimization. *Proceedings of the Sixt International Symposium on Quality Electronic Design (ISQED05)*, pages 516–521, March 2005.

[10] Jean-Baptiste Rigaud Quoc Thai Ho and Laurent Fesquet et Al. Implementing Asynchronous Circuits on LUT Based FPGAs. *Springer Link, Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream*, pages 36–46, January 2002.

[11] Rune André Bjørnerud. Event System Implementation for Microcontroller Circuits. Master's thesis, Norwegian University of Science and Technology (NTNU), June 2009.

[12] Ran Ginosar Rostislav Dobkin and Avinoam Kolodny. Fast Asynchronous Shift Register for Bit-Serial Communication. *Proceedings of the 12th IEEE International Symposium on Asynchronous Circuits and Systems (Asynch06)*, pages 118–127, March 2006.

[13] Steven Burns Scott Hauck, Gaetano Borriello and Carl Ebeling. MONTAGE: An FPGA for Synchronous and Asynchronous Circuits. *Field-Programmable Gate Arrays: Architectures and Tools for Rapid Prototyping*, pages 44–51, 1993.

[14] Alexander Borisovitch Smirnov. *Asynchronous Micropipeline Synthesis System*. PhD thesis, Boston University, 2009.

[15] Jens Sparsøand Steve Furber. *Principles of Asynchronous Circuit Design A Systems Perspective*. Springer Link, 2001. Available as pdf.

[16] John Teifel and Rajit Manohar. An Asynchronous Dataflow FPGA Architecture. *IEEE Transactions on Computers*, 53:1376–1392, November 2004.

[17] John Teifel and Rajit Manohar. Highly Pipelined Asynchronous FPGAs. *Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, pages 133–142, 2004.

[18] Tzyh-Yung Wuu and Sarma B. K. Vrudhula. A Design of a Fast and Area Efficient Multi-input Muller C-element. *IEEE Transactions on VLSI Systems*, 1:215–219, June 1993.

[19] Eslam Yahya and Marc Renaudin. QDI Latches Characteristics and Asynchronous Linear Pipeline Performance Analysis. *Springer Link, Lecture Notes in Computer Science*, pages 583–592, September 2006.

[20] Eslam Yahya and Marc Renaudin. Performance Modeling and Analysis of Asynchronous Linear-Pipeline with Time Variable Delays. *IEEE Electronics, Circuits and Systems, 14th IEEE International Conference*, pages 1288–1291, December 2007.