Asynchronous Design

Florian Deeg, Jie Zhu, Sebastian M. Sattler
Chair of Reliable Circuits and Systems
Friedrich-Alexander-Universität Erlangen-Nürnberg
Paul-Gordan-Str. 5, 91052 Erlangen, Germany
Email: {florian.deeg,jie.zhu,sebastian.sattler}@fau.de

***Abstract*—The paper discusses the challenges and solutions to asynchronous design and suggests a secured design by stabilizing all states with RS-Buffer in Dual-Rail.**

I. Motivation

Although synchronized circuits are state-of-the-art, the consideration of asynchronous circuits with various advantages (lower power consumption, better system performance, no clock skew problems) has become increasingly important in recent years. [1] An essential feature of asynchronous circuits in contrast to synchronized circuits is that they are controlled without a global clock edge, which is why errors, such as hazards, glitches and races, can occur.

II. Hazards

Hazards describe the possibility that a short-term fault may affect an electronic circuit. If a hazard actually appears at the output of a circuit, it is called a hazardfehler. Hazards are distinguished in principle in their origin in functional and structural hazards and in their effect in static or dynamic hazards. [2]

A. Functional Hazards

Functional hazards are potential errors at the output of a circuit, if at least two variables do not change at the same time and an intermediate error signal occurs (due to the function, hence the name). Function hazards are caused by wrong switching of at least two input signals.

1) Static Functional Hazards: Static functional hazards retain their value at the end and may briefly produce false signals, see Fig. 1.

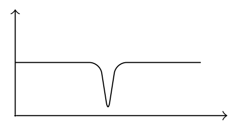

Fig. 1: Static 1-Hazard

2) Dynamic Functional Hazards: Dynamic functional hazards are potential false signals that can occur during a transition of an output signal if at least three input signals change, see Fig. 2. Every dynamic functional hazard also contains a static functional hazard, so one must distinguish between static and dynamic. There may be a static functional hazard error that is not contained in any dynamic functional hazard error.

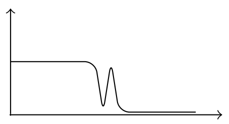

Fig. 2: Dynamic Functional Hazard

3) Prevention of Functional Hazards: Functional hazards can now be avoided by one-step changing an input signal. This is illustrated by the partial automaton in Fig. 3. Its

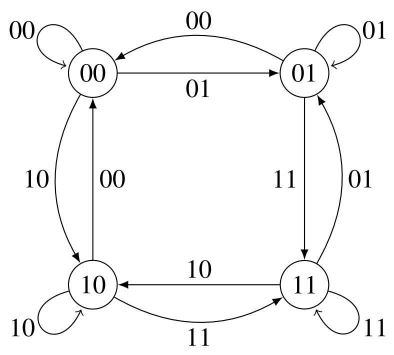

Fig. 3: One-Step Encoded Automaton (Graph)

representation as a phase list (table of values) is given in Tab. I, its representation as a multiset (KV diagram) in Fig. 4. From

i	z_1	z_0	x_1	x_0	z_1	z_0
0	0	0	0	0	0	0
1	0	0	0	1	0	1
2	0	0	1	0	1	0
3	0	0	1	1	$*$	$*$
4	0	1	0	0	0	0
5	0	1	0	1	0	1
6	0	1	1	0	$*$	$*$
7	0	1	1	1	1	1
8	1	0	0	0	0	0
9	1	0	0	1	$*$	$*$
A	1	0	1	0	1	0
B	1	0	1	1	1	1
C	1	1	0	0	$*$	$*$
D	1	1	0	1	0	1
E	1	1	1	0	1	0
F	1	1	1	1	1	1

TABLE I: Phase List of the Automaton

this, for example, the unary z-variables $z_1 = Z_1^1$ and $z_0 = Z_0^1$

can be resolved from the logic state functions $\delta_z = (\delta_{z_1}, \delta_{z_0})$ of Equ. (1)-(2):

$$\delta_{z_1}(z,x) = z_1x_1 \vee \bar{z}_0x_1\bar{x}_0 \vee z_0x_1x_0 \quad (1)$$

$$\delta_{z_0}(z,x) = z_0x_0 \vee z_1x_1z_0 \vee \bar{z}_1\bar{x}_1x_0 \quad (2)$$

Following a two-level implementation of gates functional hazards aren't excited anymore. During transitions any signals only change in one variable at a time. Asterics $(*)$ are forbidden assignments [3] and implemented by Don't Cares in Fig. 5.

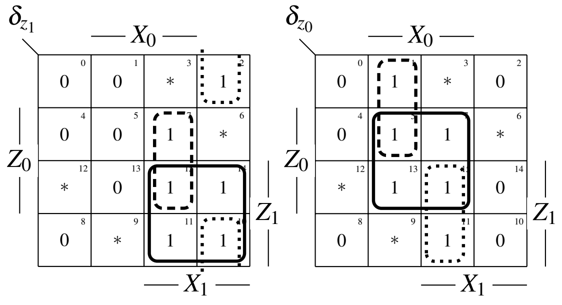

Fig. 4: KV-Diagram of δ_{z_1} and δ_{z_0}

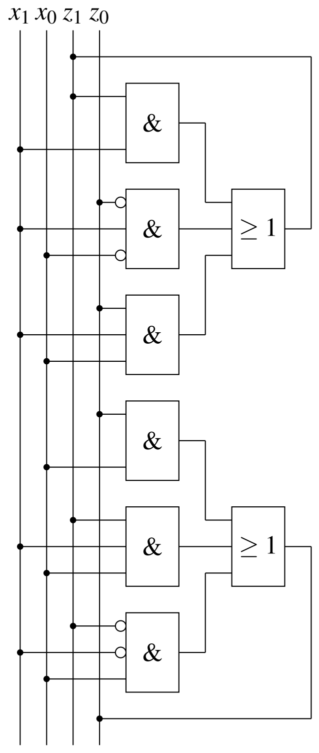

Fig. 5: Digital Circuit of the Automaton

Functional hazards can also be avoided by synchronizing the input signals through e.g. D-Flipflops as shown in Fig. 6. Thus input signals are always applied to the circuit simultaneously with the change of clock (here the positive change from 0 to 1). The change of a state is thus effected by all input signals simultaneously in parallel, i.e. synchronized with the positive edge of the clock.

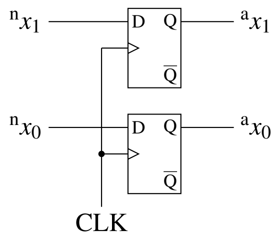

Fig. 6: Synchronization of Inputs (x_1, x_0)

B. Structural Hazards

Structural hazards describe, as the name implies, the possibility of a short-term false signal due to differences in time delay in a circuit, i.e. due to different delays between paths of a circuitry. As an example, a structure which implements the z-variable (function z) from Fig. 7 is presented in Fig. 8. Since the respective parts of the circuit are not overlapped designed, a structural hazard can be created by switching from input $7 = [111]$ to input $5 = [101]$, if $x_1 = [0]$ and $\bar{x}_1 = [0]$ are present for a short time. This can happen due to the faster switching of x_1 over $\bar{x}_1$.

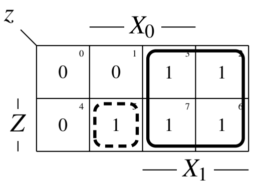

Fig. 7: Example KV-Diagram

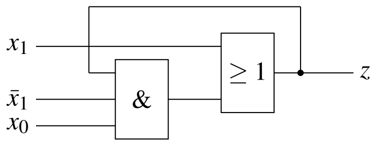

Fig. 8: Structural Hazardous Implementation

1) Prevention of Structural Hazards: The avoidance of such hazards is done by overlapping the prime blocks. With the help of additional circuitry (additional paths) a hazard-free implementation of equal but different signals, i.e. 1's or 0's, is achieved. They must be "counted" overlapping as shown in Fig. 9, their implementation is given in Fig. 10.
Structural hazards also show no effect at the output when designing in Dual-Rail with the RS-Buffer. This design method is explained in more detail in the following chapter. However, one should first concentrate on the overlapping prime blocks to avoid structural hazards and thus obtain a safe circuit.

III. Glitches

Glitches describe short term false signals, which can occur e.g. by irradiation of charged particles into paths of electrical circuits. [4] We assume here hazard-similar short-term false signals to discuss the avoidance of such glitches.

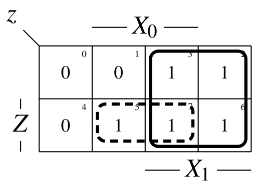

Fig. 9: Overlapping 1-Function Blocks

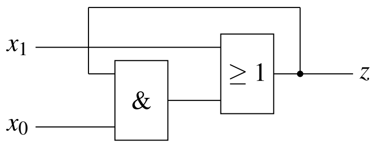

Fig. 10: Structural Hazard-Free Implementation

1) Prevention of Glitches: The RS-Buffer (RSB) and the Dual-Rail Logic (DRL) make it possible to realize a SEU-hard circuit. For a better understanding, the RSB as well as the design in DRL are briefly discussed. [5] An RS-Buffer consists of a tri-state driver and a so-called babysitter. As shown in Fig. 11, the tri-state data path is the first stage of the RSB. It consists of two PMOS and NMOS transistors arranged in series. The signals $\bar{R}$ and S are each connected to a PMOS and a NMOS pin. The structure exhibits the property that the transistors arranged in series are only switched on at $\bar{R} = S$. Otherwise the tri-state is in a high-impedance state. In the case of the unexpected invalid input combinations $\bar{R}S = [01]$ or $R\bar{S} = [10]$, the babysitter's task is to keep the current state awake until a new valid input combination is switched through via the tri-state. The second stage of the RSB, that is the control path - the so-called babysitter, consists of a two-stage, fed back inverter chain, in which the feedback inverter is realized with a long-channel transistor. A lower driver capability is implemented, that it can always be overruled by a signal driven by the tri-state. The truth table for the output B of the RSB is given in Tab. II, its formal description is $B = \bar{R}S \vee \bar{R}B \vee SB$.

S	R	B	Comment
0	0	B	Hold
0	1	0	Reset
1	0	1	Set
1	1	B	Hold

TABLE II: Truth Table of the RS-Buffer

For the purpose of a consistent and structural analysis of asynchronous circuits, the symbol of the RS-Buffer (RSB) has been chosen as shown in Fig. 12. With the help of the RSB, a state can now be maintained for unexpected input assignments. It is therefore possible to intercept unexpected signals, i.e. to muffle them. For the successful use of an RSB in asynchronous circuitry, however, it is mandatory to feed them by a Dual-Rail Logic (DLR). In this logic, the ones (1-signals) are propagated along the positive rail, the zeros (0-signals) are propagated along the negative rail. BUT, at gate level (block diagram level) the ones are understood as a set and the zeros are understood as a reset. Thus, in propositional logic in Dual-Rail with RS-Buffer

- the 1's of a logical function are coded as 1's,
- the 0's of a logical function are coded as 1's.

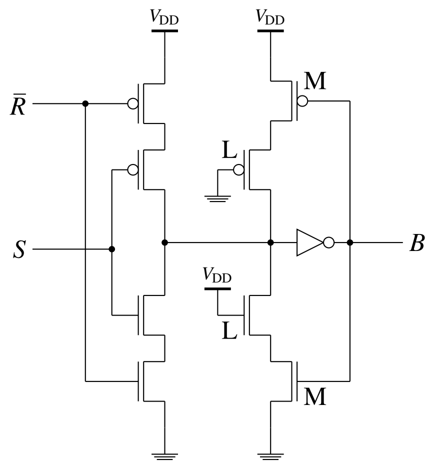

Fig. 11: Schematic of the RS-Buffer

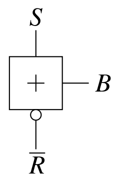

Fig. 12: Circuit Symbol of the RS-Buffer

The example of Fig. 3 is now provided in Fig. 13 in DRL with RSB. With 1's and 0's of both rails being coded as 1's in propositional logic, binary functions are given in Equ. (3)-(6).

$$\delta_{z_1} = z_1x_1 \vee z_1\bar{z}_0x_0 \vee z_0x_1x_0 \vee \bar{z}_0x_1\bar{x}_0 \tag{3}$$

$$\bar{\delta}_{z_1} = \bar{z}_1\bar{x}_1 \vee z_0\bar{x}_1 \vee \bar{z}_1\bar{z}_0x_0 \vee z_1\bar{x}_1\bar{x}_0 \vee \bar{z}_1z_0\bar{x}_0 \tag{4}$$

$$\delta_{z_0} = z_0x_0 \vee \bar{z}_1\bar{x}_1x_0 \vee \bar{z}_1z_0x_1 \vee z_1x_1x_0 \tag{5}$$

$$\bar{\delta}_{z_0} = \bar{x}_1\bar{x}_0 \vee z_1\bar{x}_0 \vee z_1\bar{z}_0\bar{x}_1 \vee \bar{z}_1\bar{z}_0x_1 \tag{6}$$

IV. Races

Races in a circuitry are races on a feedback line (between their signals on a single line or more than a single line) and thus dependencies on the change of a state, i.e. on state variables.

A. RS-Latch as an Example for Races

To take a closer look at races, an example structure is first considered, see Fig. 14.

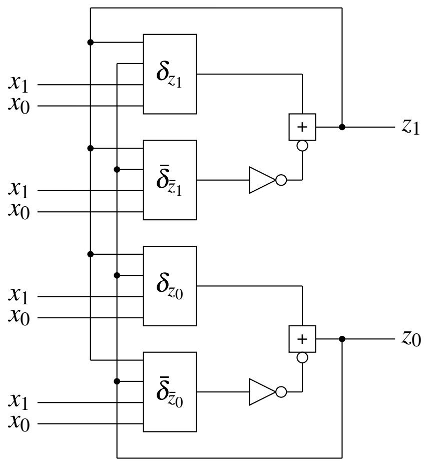

Fig. 13: Glitch-Free Dual-Rail Logic

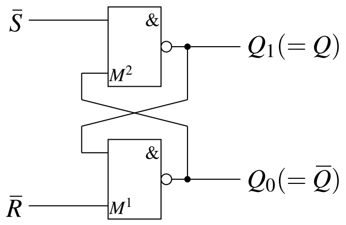

Fig. 14: Low-Active RS-Latch

A race can be triggered when $\bar{S}\bar{R}$ switches from [00] to [11] as in Fig. 15.

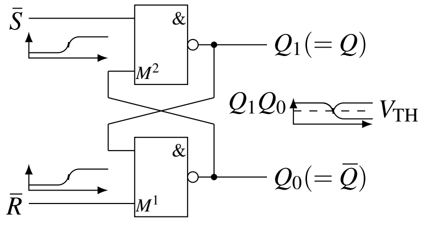

Fig. 15: Example of a Race

In this example, the output Q_0 wins, because it first switches to Low and thus turns the other output to High. This state then stabilizes itself. It could also come to the assignment $(Q_1, Q_0) = [01]$, but never to the oscillation. Since the subsequent state of the race constellation cannot be predicted, this assignment is numbered $[* \; *]$.

B. Characteristics of Races

Races describe undetermined signal superpositions of at least two signals on at least one feedback line. The destructive signal superpositions are caused by the change of at least two state variables. If the output depends on the propagation of both signals, it is called a race.

C. Prevention of Races

To avoid races, asynchronous automata are designed one-step incremental [6]. Thus, only one state variable may change at a time during a state transition. The example from chapter II shows an automaton, that is completely encoded one-step.

V. Example of an Asynchronous Automaton

A functional hazard-free automaton for at-speed direction recognition is considered, see Fig. 16. Only a single input can change at each transition. This is called correct-by-construction.

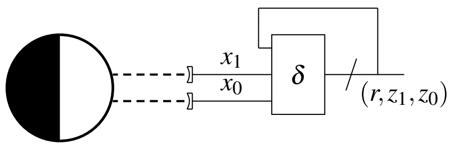

Fig. 16: Direction Recognition at a Rotating Object

In order to obtain the race-free implementation of the circuit in Fig. 17, the machine is one-step encoded in its states. It means that only one state variable changes during the time of a transition. To realize a glitch-free circuit, the unused edges,

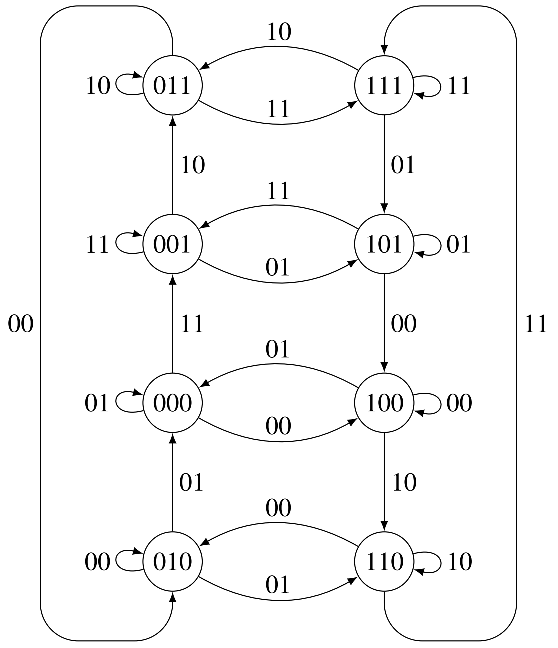

Fig. 17: One-Step Encoded Machine

which cannot occur, will be encoded as "hold", meaning the machine will maintain the last state. Using the KV-diagrams

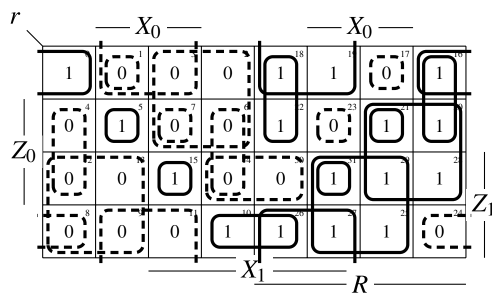

Fig. 18: KV-Diagram of $r = (R, \bar{R})$

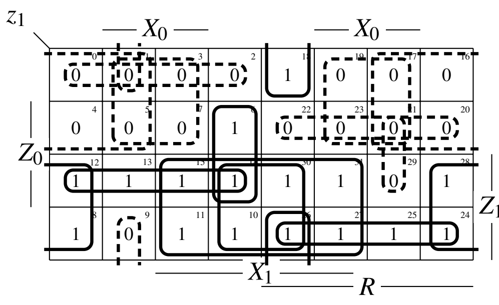

Fig. 19: KV-Diagram of $z_1 = (Z_1, \bar{Z}_1)$

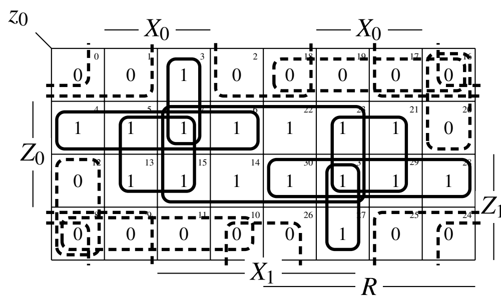

Fig. 20: KV-Diagram of $z_0 = (Z_0, \bar{Z}_0)$

of the unary variables R, $\bar{R}$, Z_1, $\bar{Z}_1$, Z_0 and $\bar{Z}_0$, see Fig. 18 to 20, the state transfer functions can be set up in Eq. (7)-(12).

$$r = rz_1x_0 \vee rz_0\bar{x}_1 \vee r\bar{z}_1\bar{x}_0 \vee r\bar{z}_0x_1 \vee \bar{z}_1z_0\bar{x}_1x_0 \vee z_1z_0x_1x_0 \vee z_1\bar{z}_0x_1\bar{x}_0 \vee \bar{z}_1\bar{z}_0\bar{x}_1\bar{x}_0 \tag{7}$$

$$\bar{r} = \bar{r}z_1\bar{x}_1 \vee \bar{r}\bar{z}_1x_1 \vee \bar{r}\bar{z}_0x_0 \vee \bar{r}z_0\bar{x}_0 \vee z_1z_0x_1\bar{x}_0 \vee \bar{z}_1z_0x_1x_0 \vee z_1\bar{z}_0\bar{x}_1\bar{x}_0 \vee \bar{z}_1\bar{z}_0\bar{x}_1x_0 \tag{8}$$

$$z_1 = z_1x_1 \vee z_1\bar{x}_0 \vee \bar{r}z_1z_0 \vee rz_1\bar{z}_0 \vee \bar{r}z_0x_1\bar{x}_0 \vee r\bar{z}_0x_1\bar{x}_0 \tag{9}$$

$$\bar{z}_1 = \bar{z}_1\bar{x}_1 \vee \bar{z}_1x_0 \vee r\bar{z}_1z_0 \vee \bar{r}\bar{z}_1\bar{z}_0 \vee rz_0\bar{x}_1x_0 \vee \bar{r}\bar{z}_0\bar{x}_1x_0 \tag{10}$$

$$z_0 = z_0x_1 \vee z_0x_0 \vee rz_1z_0 \vee \bar{r}\bar{z}_1z_0 \vee rz_1x_1x_0 \vee \bar{r}\bar{z}_1x_1x_0 \tag{11}$$

$$\bar{z}_0 = \bar{z}_0\bar{x}_1 \vee \bar{z}_0\bar{x}_0 \vee r\bar{z}_1\bar{z}_0 \vee \bar{r}z_1\bar{z}_0 \vee r\bar{z}_1\bar{x}_1\bar{x}_0 \vee \bar{r}z_1\bar{x}_1\bar{x}_0 \tag{12}$$

The resulting dual-rail implementation of the machine is given in Fig. 21.

VI. Conclusion and Future Work

In order to obtain delay-insensitive circuits, the one-step encoding of both the input and output is important. If and only if one input or state variable changes at a time, time difference of signals no longer plays a role. In order to make the paths through a circuit additionally hardened against structural hazard errors, prime blocks (for maxterms and minterms) must be selected overlapping, so that differences in path propagation time disappear. In the future, high-performance algorithms will be required for the one-step encoding of (state) variables as well as for high fan-in cascaded circuitry in order to avoid any kind of structural hazards. This will create new possibilities to emulate, design and protect programs, circuits and systems against delay.

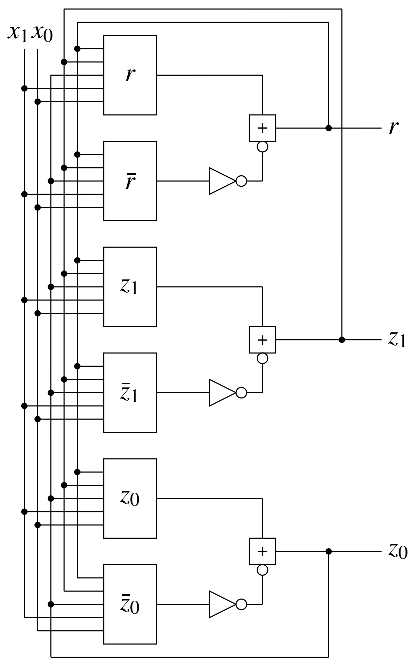

Fig. 21: Glitch-Free Dual-Rail Implementation

References

[1] Carroll, C.: Asynchronous finite state machine design: A lost art? ASEE, 2006.

[2] Zander H.: Logischer Entwurf binärer Systeme. VEB Verlag Technik Berlin, 1989.

[3] Wuttke H.-D.; Henke K.: Schaltsysteme, Eine automatenorientierte Einführung. Pearson München, 2003.

[4] Karnik, T:, Hazucha, P.:, Patel, J.: Characterization of Soft Errors Caused by Single Event Upsets in CMOS Processes. IEEE Trans. on Depend. and Sec. Comp., Vol. 1, No. 2, Apr-Jun 2004.

[5] Özgül, M.: Deeg, F.:, Sattler, S.M.: Mealy-to-Moore Transformation - A state stable design of automata. ASTES Jorunal, Vol. 2, Issue 6, Page No 162-174, 2017.

[6] Elbably, M.E.: State Machine Transition to avoide the Race Conditions in Asynchronous Sequential Logic Circuits. 7th Nat. Radioscience Conf.,Feb. 22-24,2000.