Mealy-to-Moore Transformation - A state stable design of automata

Mustafa Oezguel, Florian Deeg, Sebastian M. Sattler

Chair of Reliable Circuits and Systems, Friedrich-Alexander-Universität Erlangen-Nürnberg, Paul-Gordan-Str. 5, 91052 Erlangen, Germany

ARTICLE INFO

Article history:
Received: 27 October, 2017
Accepted: 02 December, 2017
Online: 10 December, 2017

Keywords :
Asynchronous Feedback
Functional Safety
Stabilization
Hazard-free
Race-free
Parallel de-composition
Dual-rail

ABSTRACT

The paper shows a method of transforming an asynchronously feedbacked Mealy machine into a Moore machine. The transformation is done in dual-rail logic under the use of the RS-buffer. The transformed machine stabilizes itself and is safe to use. The transformation is visualized via KV-diagrams and calculated with formulas. We will present three use-cases for a better understanding. To underpin the stated transformation a simulation is also presented.

1 Introduction

A Synchronous circuits entail several benefits. Firstly performance advantages, e.g. faster system performance, no clock synchronization, fewer power consumption and fewer electromagnetic emission. Secondly asynchronous circuits also have safety advantages, e.g. no clock skew. Asynchronous circuits can also lead to simplicity, by connecting different parts modularly [1, 2]. On the other hand asynchronous circuits provide risks, such as faults like hazards, glitches and so on [3]. Therefore it is very important to design asynchronous circuits correctly, especially in safety critical applications such as in automotive, where human lives are at stake. To prevent faults like hazards, a mealy-to-moore transformation will be presented, which creates a function stable asynchronous machine. The Mealy machine with the state transfer function \( \delta \) and the output function \( \lambda \), see figure 1 be given. Each Mealy machine can be transformed into an equivalent Moore machine [3] for example by increasing the arity of \( \delta \) by \( |x| \) and coding it with \( x, z' = (z, x) \) with \( \delta'(z', x) \mapsto (\delta(z, x), x) \), and the output function is only dependent on the new state variable \( z' \) with \( \mu : (z') \mapsto \mu(z') \).

By comparing the two machines the pros and cons can be shown. Therefore the idea of transforming the machines into each other to profit from the benefits of both is obvious. Mealy machines have the advantage of requiring less states since one state can produce a number of different outputs in combination with the input. A Moore machine’s state on the other hand only produces one output. A Mealy machine is also faster by reacting directly to the input. This feature however is not always wanted, since it can lead to undesired outputs (e.g. hazards, glitches) when the input is variable. A Moore machine is more stable in this regard, since it only indirectly reacts to input changes. The output only changes when transferring into the next state. Transforming a Mealy machine into a Moore machine is therefore useful in case a direct dependence on the input is to be avoided [2].

In the paper the transformation of a Mealy machine into a Moore machine is presented. The machine will be designed under the use of dual-rail logic and the RS-buffer, and will stabilize itself. Firstly the transformation is made by breaking the feedback up and nesting it in the RS-buffer, secondly by encoding the inputs as states, so that the output function is only dependent on pseudo states. With the method presented in this article, function stable sequential circuit parts can be seen as combinational logic and moved over other blocks at will. With this method, individual machines can be strung together to realize complex circuits for safety relevant applications.

2 Theory

This paper describes the underlying theory of the transformation and provides an illustrative example.

Figure 1 shows a fully asynchronous Mealy machine. The branches entering a node in the graph should end reflexively, so that only transient states are allowed which are conscious and triggered from the outside [2]. To consider all branches of the Mealy machine as locally reflexively

Corresponding Author: Mustafa Oezguel, mustafa.oezguel@fau.de

www.astesj.com 162
https://dx.doi.org/10.25046/aj020621
concluded, the feedback should be moved over $\lambda$, making the output $y$ the feedback. In order to set the correct state $z$ for the transfer function $\delta$, a function $\lambda^{-1}$ is realized. This function generates the reduced state $\bar{z}$ from the feedback $y$. This equivalent transformation is outlined in figure 2.

\[ \delta(\bar{z}, x) = \bar{z} \]
\[ \delta(\lambda^{-1}(y, x), x) = \lambda^{-1}(y, x) \]
\[ \lambda\left(\delta(\lambda^{-1}(y, x), x), x\right) = y \]
\[ \text{mit } \lambda\left(\lambda^{-1}(y, x), x\right) = y \]

The next stage of transformation shows at first sight no feedback. Only the RS-buffer, which is in $\bar{\delta}$, has a nested feedback. This transformation is outlined in figure 3. Transient states can be triggered via the input, while state stable states are being held via the RS-buffer. In order to set the correct state $\bar{z}$ to $\mu$, the function $\bar{\delta}$ must be well designed, which will be shown in the following sections.

\[ \bar{\delta}(\bar{z}, x) = \bar{z} \]
\[ \mu(\bar{z}, x) = y \]

2.1 Dual-rail logic

For the implementation in dual-rail logic [4], the fully asynchronous circuit from figure 1 will be divided in the 1- and 0-share. This is done by partitioning the state transfer function and the output function, see figure 4.

The functions $\delta$ and $\lambda$ are each w.l.o.g. realized in two blocks $\delta = (\delta_z, \delta_\bar{z})$ and $\lambda = (\lambda_z, \lambda_\bar{z})$. In order to guarantee this secure dual rail structure, RS-buffers are used. The mode of operation corresponds in a certain manner to the C-element [4, 5]. The function $\delta$ of figure 3 will be designed by appropriate coding according to the RS-buffer, showing a general structure as outlined in figure 5.

The stabilized 1-states, $\Lambda_\bar{z} \rightarrow \bar{z}$, are transformed into 1-outputs, $\Lambda_z \rightarrow y$. The corresponding Venn diagram of the stabilized 1-states in 1-outputs can be seen in figure 6. 1-states that appear as 0-outputs are declared as $\Lambda_\bar{z}$, 0-states which appear as 1-outputs are declared as $\Lambda_z$. The 1-partition is composed of:

\[ \Lambda_z \rightarrow \lambda_z(\delta_z) + \lambda_\bar{z}(\delta_\bar{z}) + \lambda_\bar{y}(\delta_\bar{y}) \]
The stabilized 0-states, $\Lambda_z$, are transformed into 0-outputs, $\Lambda_z$. The corresponding Venn diagram of the 0-states in 0-outputs is shown in figure 7. 0-states which appear as 1-outputs are declared as $\Lambda_z$, 1-states which appear as 0-outputs are declared as $\Lambda_z$. The 0-partition is composed of:

$$\Lambda_z = \Lambda_y(\delta_z) + \Lambda_y(\delta_z)$$

To understand the coding, first the RS-buffer has to be understood.

### 2.2 RS-Buffer

The RS-Buffer used and its circuit symbol are represented in figure 8 and 9 and its truth table is specified in table 1.

<table>
<thead>
<tr>
<th>R</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
<td>B</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1: Truth table of the RS-Buffer

The RS-buffer is used to synchronize a dual-rail signal by only letting a state transfer happen, if both incoming signals are disjoint (how it is desired) to each other. This means that there is no overlaying of signal, which could lead to fatal faults. The RS-buffer consists of the tri-state driver, which is the first state and the second stage the so-called babysitter. The driving structure now provides 3 functions. Firstly the setting signal, which means the output is in the logical 1-state. Secondly the resetting signal, which means the output is in logical 0-state and least the hold state which holds the last state at the output by keeping the signal in the babysitter and having a high impedance state at the tri-state. The babysitter consists of two complementary loops. The formula for the output $B$ is therefore $B := R(B \lor S) \lor S(B \lor R) = RS \lor RB \lor SB$.

### 2.3 Design of the reduced state transfer function $\delta$

As you can see the RS-buffer is coded via a 3-partitioning. There is the setting signal $S=[1]$, the resetting signal $R=[1]$ and the hold signal $RS=[00] \lor [11]$. The state transfer function of an automaton can in turn be split in 4 parts. Firstly
the transient 1-share, that means independent of the state the next state will be at logical 1, secondly the transient 0-share, that means output at logical 0 independent of the state, thirdly the state stable share of the state transfer function, that means the share, which is at constant input dependent on the value of the last state and the state instable state, which means a permanent change of the state at constant input, see figure 10. The resulting 3 parts, which are left, can now be easily compared to the 3-partition of the RS-buffer. The transient 1-share $\delta^1_z$ is realized by the setting signal of the RS-buffer, similar the transient 0-share $\delta^0_z$ is realized by the resetting signal and the state stable share is realized by the hold signal of the RS-buffer.

\[
\begin{align*}
\delta^1_x & = \delta^1(z = 1, x) \\
\delta^0_x & = \delta^0(z = 0, x) \\
\delta^s_x & = \delta^s(z = 1, x) \\
\delta^s_x & = \delta^s(z = 0, x)
\end{align*}
\]

This work only concentrates on function stable machines, so that state instable parts are not allowed. State instable parts can be deduced via

\[\delta^i \land \delta^s\]

Functions must not exhibit these parts. The other parts can be deduced by the following expressions

- Transient 1: $\delta^1_z = \delta^1 \land \delta^s$
- Transient 0: $\delta^0_z = \delta^0 \land \delta^s$
- State stable: $\delta^s \land \delta^s$

This leads to the general formula for $\delta^i$ for $n$ states:

\[
\begin{align*}
\delta^i_z & = \delta^1_z + \neg \delta^0_z \\
& = (\delta^1_z(z = 1, x) \land \delta^0_z(z = 0, x)) \\
& + \neg (\delta^0_z(z = 1, x) \land \delta^1_z(z = 0, x))
\end{align*}
\]

with $i = 0 \ldots n - 1$

2.4 Design of the output function

The detailed description of the Venn-diagrams given in figure 11 and 12 is reported in [6].
3 Use-Case

To get a better insight, three examples will be presented.

3.1 1-dimensional example

For \( x = (x_1, x_0) \), \( y = (y) \) and \( z = (z) \) the transformation of the Mealy machine \((X, Y, Z, \delta, \lambda)\) with the state transfer function and output function

\[
\delta(z, x) = x_1 x_0 \lor z x_0 \\
\lambda(z, x) = x_1 x_0 \lor z_1
\]

will be executed. The automaton before the transformation can be seen in figure 13.

![Figure 13: Mealy machine before the transformation](image)

3.1.1 Design of \( \delta \)

The KV-diagram of the state transfer function can be seen in figure 14.

![Figure 14: The state transfer function](image)

To code the RS-buffer correctly, firstly the KV-diagram will be compacted, see figure 15.

![Figure 15: Compacted KV-diagram of \( \delta \)](image)

The parts of the state transfer function can easily be seen and the KV-diagrams for \( \delta_+ \) and \( \delta_- \) can be deployed, see figure 16. For \( \delta_- \), 0 is coded as 1 and 1 is coded as 0 respectively, because of the dual-rail structure.

![Figure 16: \( \delta_- \) and \( \delta_+ \)](image)

For the next steps of the transformation, first the complement of \( \delta \) has to be calculated:

\[
\delta^c(x, z) = z_1 x_0 \lor x_0
\]

The parts of the state transfer function are:

\[
\begin{align*}
\delta_+ &= \delta(z = 1, x) = x_0 \\
\delta_- &= \delta(z = 0, x) = x_1 x_0 \\
\delta_+^c &= \delta^c(z = 1, x) = x_0 \\
\delta_-^c &= \delta^c(z = 0, x) = x_1 \lor x_0
\end{align*}
\]

It has to be checked, if there are any instable parts:

\[
\begin{align*}
\delta_+ \land \delta_-^c &= x_0 \\
\delta_+ \land \delta_- &= x_0
\end{align*}
\]

\( \delta_+ \land \delta_-^c \) is 0 means the function has no instable parts.

The functions \( \delta_+ \) and \( \delta_-^c \) can now be calculated via

\[
\begin{align*}
\delta_+ &= \delta_+ \land \delta_- = x_0 \\
\delta_-^c &= \delta_-^c \land \delta_+ = x_0
\end{align*}
\]

The coding for \( \delta \) of the RS-buffer is as follows:

\[
\begin{align*}
(r, s) \text{ Set:} & \quad (++1) = x_1 x_0 \text{ (transient 1-share)} = \delta_+ \\
\text{ Reset:} & \quad (1+) = x_0 \text{ (transient 0-share)} = \delta_-^c \\
\text{ Hold:} & \quad (**) = x_1 x_0 \text{ (state stable share)} = \delta_-^c
\end{align*}
\]

Now \( \delta \) can be derived:

\[
\begin{align*}
\delta &= \delta_+ + \lnot \delta_-^c \\
&= x_1 x_0 + \lnot x_0
\end{align*}
\]

3.1.2 Design of \( \lambda \)

The stabilized state transfer function has now been designed, now the output function must be designed. Therefore the output function has to be layed over the state transfer function and the parts have to be specified:

![Figure 17: Output function \( \lambda \)](image)
The parts are:

\[
\begin{align*}
A(z, x) &= \overline{x_0} x_1 x_0 \\
B(z, x) &= \overline{x_1} x_0 \lor \overline{x_1} \overline{x_0} \\
C(z, x) &= x_1 x_0 \\
D(z, x) &= x_1 \overline{x_0} \lor \overline{x_0}
\end{align*}
\]

This leads to the output function:

\[
\begin{align*}
y &= A \lor B = \overline{x_1} x_0 \lor \overline{x_1} \\
\overline{y} &= \overline{C} \lor \overline{D} = x_1 \lor \overline{x_0} \\
y &= y + \overline{y} = x_1 x_0 \lor \overline{x_1} + (x_1 \lor \overline{x_0})
\end{align*}
\]

The resulting automaton has the form of figure 19 with the determined functions \( y = A \lor B \) and \( \overline{y} = \overline{C} \lor \overline{D} \).

### 3.2 2-dimensional example

For \( x = (x_1, x_0) \), \( y = (y) \) and \( z = (z_1, z_0) \) the transformation of the Mealy machine \((X, Y, Z, \delta, \lambda)\) with the state transfer functions and output function

\[
\begin{align*}
\delta_{0}(z, x) &= \overline{x_0} \lor z_1 x_1 \\
\delta_{1}(z, x) &= \overline{x_0} x_1 \lor z_1 x_1 x_0 \\
\lambda(z, x) &= z_1 x_1 \lor \overline{z_0} \overline{x_1} x_0
\end{align*}
\]

will be executed. The machine before the transformation is outlined in figure 20.

![Figure 20: Automaton before the transformation](image)

#### 3.2.1 Design of \( \delta_{0} \)

The KV-diagram of figure 21 will be lossless compacted given by the pointers in the diagram to the diagram of figure 22.

![Figure 21: KV-diagram of \( \delta_{0} \)](image)

![Figure 22: compacted KV-diagram of \( \delta_{0} \)](image)

The KV-diagram of figure 21 will be lossless compacted given by the pointers in the diagram to the diagram of figure 22.

With the compacted KV-diagram the RS-buffer for \( \delta_{0} \) can be coded by specifying the functions \( \overline{\delta}_{0} \) and \( \overline{\delta}_{0} \), see

![Figure 19: Transformed Moore](image)

The truth table of the automaton:

<table>
<thead>
<tr>
<th>#</th>
<th>( z )</th>
<th>( x_1 )</th>
<th>( x_0 )</th>
<th>( \delta )</th>
<th>( \overline{\delta} )</th>
<th>( \delta_\lambda )</th>
<th>( A )</th>
<th>( B )</th>
<th>( \overline{C} )</th>
<th>( D )</th>
<th>( A \lor B )</th>
<th>( \overline{C} \lor \overline{D} )</th>
<th>( \overline{y} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>1</td>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>*</td>
<td>0</td>
<td>1</td>
<td>*</td>
<td>1</td>
<td>*</td>
<td>1</td>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>*</td>
<td>1</td>
<td>0</td>
<td>*</td>
<td>*</td>
<td>1</td>
<td>*</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>*</td>
<td>*</td>
<td>1</td>
<td>*</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>*</td>
<td>1</td>
<td>0</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>1</td>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>*</td>
<td>1</td>
<td>1</td>
<td>*</td>
<td>1</td>
<td>*</td>
<td>1</td>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>*</td>
<td>*</td>
<td>1</td>
<td>0</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>*</td>
<td>1</td>
<td>0</td>
<td>*</td>
<td>*</td>
<td>1</td>
<td>*</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2: Partial truth table of the resulting automaton
The partial function \( \overline{\delta}_{\tau_0} \) delivers the 1’s of \( \delta_{\tau_0} \) as 1’s, while \( \overline{\delta}_{\tau_0} \) delivers the 0’s of \( \delta_{\tau_0} \) as 1’s. In this state transfer function, there is no state stable part only transient parts. Making the stable \( \overline{\delta}_{\tau_0} \) a normal dual-rail structure.

State instable parts are not allowed in the state transfer function:

\[
z_0 \delta_{\tau_0} \land z_0 \overline{\delta}_{\tau_0} = (\overline{z_0} \lor z_1 x_1) \land (\overline{z_0} x_0 \lor \overline{x_1} x_0) = 0
\]

Now the resulting \( z_{\delta_0} \) can be calculated:

\[
\begin{align*}
\delta_{\tau_0}(z, x) &= z_0 \delta_{\tau_0} \land \overline{z_0} \delta_{\tau_0} = \overline{z_0} \lor \overline{x_1} x_1 \\
\overline{\delta}_{\tau_0}(z, x) &= z_0 \overline{\delta}_{\tau_0} \land \overline{z_0} \overline{\delta}_{\tau_0} = z_0 x_0 \lor \overline{x_1} x_0 \\
z_{\delta_0} &= \delta_{\tau_0} + \overline{\delta}_{\tau_0} = \overline{z_0} x_0 \lor \overline{x_1} x_1 \\
z_{\overline{\delta}_{\tau_0}} &= \overline{z_0} x_0 \lor \overline{x_1} x_1 + (z_0 x_0 \lor \overline{x_1} x_1)
\end{align*}
\]

### 3.2.2 Design of \( \overline{\delta}_{\tau_1} \)

For \( \overline{\delta}_{\tau_1} \) the same approach applies. First the KV-diagram, see figure 24 will be compacted:

Now the functions \( \overline{\delta}_{\tau_1} \) and \( \overline{\delta}_{\tau_1} \) can be deduced:

\[
\begin{align*}
\delta_{\tau_1}(z, x, y) &= (z_1, z_0, y) \\
(001) &= A \\
(011) &= B \\
(101) &= C \\
(111) &= D
\end{align*}
\]

No instable parts, so the RS-buffer can be coded for \( \delta_{\tau_1} \) by determining the functions \( \overline{\delta}_{\tau_1} \) and \( \overline{\delta}_{\tau_1} \):

\[
\begin{align*}
\delta_{\tau_1}(z, x) &= z_1 \delta_{\tau_1} \lor z_0 \delta_{\tau_1} = z_0 x_1 x_0 \\
\overline{\delta}_{\tau_1}(z, x) &= z_1 \overline{\delta}_{\tau_1} \lor z_0 \overline{\delta}_{\tau_1} = \overline{z_0} x_1 x_0 \\
z_{\tau_1} &= \delta_{\tau_1} + \overline{\delta}_{\tau_1} = \overline{z_0} x_1 x_0 + (z_0 x_0 \lor \overline{x_1} x_0)
\end{align*}
\]

### 3.2.3 Design of \( \lambda \)

The KV-diagram of \( \lambda \) can be seen in 27. Because of the two-dimensional state, \( \lambda \) has to be divided into eight parts. They exist as follows:

\[
\begin{align*}
(\lambda_1, \lambda_0, y) &= \overline{\lambda}_1 \\
(001) &= A \\
(011) &= B \\
(101) &= C \\
(111) &= D
\end{align*}
\]

The parts of \( \lambda \) are set up in the KV-diagram, see figure 27.
The output functions $y$ and $\overline{y}$ can be inferred from this:

$$y = A \lor B \lor C \lor D$$

$$\overline{y} = E \lor F \lor G \lor H$$

The resulting automaton can be seen in [30]. As you can see in table 3, situation 7 leads to a cycle, by stepping to configuration 11, which steps back to configuration 7, and so on. So to have a state stable design in multistate machines, it is also necessary to regard the transitions between the states. A rule has to be found, to filter these race conditions. In the following transformation the automaton of subsection b is stabilized, by changing $\delta_{z_1}$.

### 3.3 Stable 2-dimensional example

The state transfer function $\delta_{z_1}$ has now been changed, to make it a stable automaton. For $x = (x_1, x_0)$, $y = (y)$ and $z = (z_1, z_0)$ the transformation of the Mealy machine $(X,Y,Z,\delta, \lambda)$ with the state transfer functions and output function

$$\delta_{z_1}(z,x) = z_0 \lor z_1 x_1$$

$$\lambda(x) = z_1 x_1 \lor z_0\overline{x_0}$$

will be executed. The machine before the transformation is outlined in figure [29].

![Figure 29: Automaton before the transformation](image)

### 3.3.1 Design of $\delta_{z_0}$

The KV-diagram of figure [31] will be lossless compacted given by the pointers in the diagram to the diagram of figure [32].

![Figure 32: KV-diagram of figure 31](image)

| # | $z_1$ | $z_0$ | $x_1$ | $x_0$ | $\delta_{z_1}$ | $\delta_{z_0}$ | $\overline{\delta}_{z_0}$ | $\overline{\delta}_{z_1}$ | $\overline{\delta}_z$ | $\overline{\delta}_y$ | $\lambda$ | $A$ | $B$ | $C$ | $D$ | $E$ | $F$ | $G$ | $H$ | $y$ | $\overline{y}$ |
| 0 | 0 | 0 | 0 | 0 | 1 | * | 1 | 0 | 1 | * | 1 | 0 | * | * | * | * | * | * | 1 | * | * | * | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | * | 1 | 0 | 1 | * | 1 | 0 | * | * | * | * | 1 | * | * | * | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 | 0 | * | * | 0 | 1 | * | 1 | 0 | * | * | * | * | 1 | * | * | * | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 | 0 | 0 | * | 1 | 0 | 1 | * | 0 | 0 | * | * | * | 1 | * | * | * | 1 | 0 |
| 4 | 0 | 1 | 0 | 0 | 0 | 1 | * | 1 | 0 | 1 | * | 1 | 0 | * | * | 1 | * | * | * | 1 | 0 |
| 5 | 0 | 1 | 0 | 1 | 0 | 0 | * | 1 | 0 | 1 | * | 0 | 0 | * | * | * | 1 | * | * | * | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 | 0 | 1 | * | * | 0 | 1 | * | 1 | 0 | * | * | * | 1 | * | * | * | 1 | 0 |
| 7 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | * | 1 | 0 | 1 | * | 0 | 0 | * | * | * | * | 1 | * | * | * | 1 | 0 |
| 8 | 1 | 0 | 0 | 0 | 0 | 1 | * | 1 | 0 | 1 | * | 1 | 0 | * | * | * | * | 1 | * | * | * | 1 | 0 |
| 9 | 1 | 0 | 0 | 1 | 0 | 0 | * | 1 | 0 | 1 | * | 0 | 0 | * | * | * | * | 1 | * | * | * | 1 | 0 |
| 10 | 1 | 0 | 1 | 0 | 1 | * | 1 | 1 | 0 | * | 1 | 1 | 0 | * | * | * | * | 1 | * | * | * | 1 | 0 |
| 11 | 1 | 0 | 1 | 1 | 1 | 0 | * | 1 | 0 | 1 | * | 1 | 1 | 0 | * | * | * | * | 1 | * | * | * | 1 | 0 |
| 12 | 1 | 0 | 0 | 0 | 1 | * | 1 | 0 | 1 | * | 1 | 1 | 0 | * | * | * | * | 1 | * | * | * | 1 | 0 |
| 13 | 1 | 1 | 0 | 0 | 0 | 1 | * | 0 | 0 | * | 1 | 0 | 0 | * | * | * | 1 | * | * | * | 1 | 0 |
| 14 | 1 | 1 | 1 | 0 | 0 | 1 | * | 1 | 1 | * | 1 | 1 | 0 | * | * | * | * | 1 | * | * | * | 1 | 0 |
| 15 | 1 | 1 | 1 | 1 | 1 | 1 | * | 1 | 1 | * | 1 | 1 | 0 | * | * | * | * | 1 | * | * | * | 1 | 0 |

Table 3: Truth table of the transformed automaton
Figure 30: Resulting automaton

Figure 31: KV-diagram of $\delta_0$

Figure 32: compacted KV-diagram of $\delta_0$

With the compacted KV-diagram the RS-buffer for $\delta_0$ can be coded by specifying the functions $\delta_0$ and $\overline{\delta}_0$, see figure 33.

State instable parts are not allowed in the state transfer function:

$$\tau_0 \delta_0 \land \tau_0 \overline{\delta}_0 = (\overline{x}_0 \lor z_1 x_1) \lor (\overline{z}_1 x_0 \lor \overline{x}_1 x_0) = 0$$

Now the resulting $z_0$ can be calculated:

$$\delta_0(z, x) = \tau_0 \delta_0 \land \tau_0 \overline{\delta}_0 = \overline{x}_0 \lor z_1 x_1$$

$$\overline{\delta}_0(z, x) = \tau_0 \overline{\delta}_0 \land \tau_0 \overline{\delta}_0 = z_1 x_0 \lor \tau_1 x_0$$

$$z_0 := \delta_0 + \overline{\delta}_0$$

$$z_0 := \overline{x}_0 \lor z_1 x_1 + \overline{\tau}_1 x_0 \lor \overline{x}_1 x_1$$
3.3.2 Design of $\delta_{z_1}$

![KV-diagram of $\delta_{z_1}$](image)

Figure 34: KV-diagram of $\delta_{z_1}$

For $\delta_{z_1}$ the same approach applies. First the KV-diagram, see figure 34 will be compacted:

![Compacted KV-diagram of $\delta_{z_1}$](image)

Figure 35: compacted KV-diagram of $\delta_{z_1}$

Now the functions $\bar{\delta}_{z_1}$ and $\delta_{\bar{z}_1}$ can be deduced:

![KV-diagrams for $\delta_{z_1}$ and $\delta_{\bar{z}_1}$](image)

Figure 36: KV-diagrams for $\delta_{z_1}$ and $\delta_{\bar{z}_1}$

First $\delta_{z_1}$ is checked for instable parts:

$$\tau_{z_1} \delta_{z_1} \land \bar{z}_1 \bar{\delta}_{z_1} = z_{0}x_{1}x_{0} \land \bar{x}_1 = 0$$

No instable parts, so the RS-buffer can be coded for $\delta_{z_1}$ by determining the functions $\bar{\delta}_{z_1}$ and $\bar{\delta}_{\bar{z}_1}$:

$$\delta_{z_1}(z,x) = z_{1} \delta_{z_1} \land \bar{z}_1 \bar{\delta}_{z_1} = z_{0}x_{1}x_{0}$$

$$\bar{\delta}_{z_1}(z,x) = \bar{z}_1 \bar{\delta}_{z_1} \land \tau_{z_1} \delta_{z_1} = \bar{x}_1$$

$$z_{1} := \delta_{z_1} + \neg(\bar{\delta}_{z_1})$$

$$\bar{z}_{1} := z_{0}x_{1}x_{0} + \neg(\bar{x}_1)$$

3.3.3 Design of $\lambda$

The KV-diagram of $\lambda$ can be seen in 35. Because of the two-dimensional state, $\lambda$ has to be divided into eight parts.

![KV-diagram of $\lambda$](image)

Figure 37: KV-diagram of $\lambda$

The parts of $\lambda$ are set up in the KV-diagram, see figure 36.

![Parts of $\lambda$](image)

Figure 38: Parts of $\lambda$

The output functions $y$ and $\bar{y}$ can be inferred from this:

$$y = A \lor B \lor C \lor D$$

$$y = \bar{E} \lor \bar{F} \lor \bar{C} \lor \bar{H}$$

$$y = \bar{x}_1x_0 \lor z_1z_0x_0 \lor z_0x_1x_0 \lor z_1x_1x_0 \lor z_1z_0x_1x_0$$

$$y = \bar{x}_1x_0 \lor \bar{z}_1x_1 \lor \bar{z}_0x_1$$

4 Test of Theory

To confirm the theory presented in the article, the state transfer function of the one dimensional example was analyzed making it a Medvedev. To simulate the circuit it was modelled with NAND gates and simulated in LT-Spice. The modeled NAND structure then was realized on the circuit board and the output was captured via an oscilloscope. The Medvedev machine $\{X, Y, Z, \delta, 1\}$ with $x = (x_1, x_0), y = (y)$ and $z = (z)$ with the state transformation function and output function be given:

$$y = A \lor B \lor C \lor D$$

$$y = \bar{E} \lor \bar{F} \lor \bar{C} \lor \bar{H}$$

$$y = \bar{x}_1x_0 \lor z_1z_0x_0 \lor z_0x_1x_0 \lor z_1x_1x_0 \lor z_1z_0x_1x_0$$

$$y = \bar{x}_1x_0 \lor \bar{z}_1x_1 \lor \bar{z}_0x_1$$
The table and circuit:

Table 4: Truth table of the transformed automaton

| # | $z_1$ | $z_0$ | $x_1$ | $x_0$ | $\delta_{z_1}$ | $\delta_{z_0}$ | $\delta_{x_1}$ | $\delta_{x_0}$ | $\lambda$ | $A$ | $B$ | $C$ | $D$ | $E$ | $F$ | $G$ | $H$ | $y$ | $\bar{y}$ | $y$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | * | 1 | 0 | 1 | * | 1 | 0 | * | * | * | * | 1 | * | * | * | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | * | 1 | 0 | 1 | 0 | 0 | * | * | * | * | 1 | * | * | * | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 | 0 | 1 | * | * | 0 | 1 | * | 1 | 0 | * | * | * | * | 1 | * | * | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 | 0 | 0 | * | * | 0 | 1 | * | 1 | 0 | 0 | * | * | * | * | 1 | * | * | 1 | 0 |
| 4 | 0 | 1 | 0 | 0 | 0 | 1 | * | * | 1 | 0 | * | 1 | 0 | 1 | * | 1 | * | * | 1 | * | * | 1 | 0 |
| 5 | 0 | 1 | 0 | 1 | 0 | 0 | * | 1 | 0 | * | 1 | 0 | 0 | * | * | * | 1 | * | * | 1 | 0 |
| 6 | 0 | 1 | 0 | 0 | 0 | 1 | * | * | 0 | 1 | * | 1 | 0 | 0 | * | * | * | * | 1 | * | * | 1 | 0 |
| 7 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | * | 1 | 0 | 1 | * | 1 | 0 | 0 | * | * | * | * | 1 | * | * | 1 | 0 |
| 8 | 1 | 0 | 0 | 0 | 1 | * | 1 | 0 | 1 | * | 1 | 0 | 0 | * | * | * | 1 | * | * | 1 | 0 |
| 9 | 1 | 0 | 0 | 0 | 0 | * | 1 | 0 | * | 1 | 0 | 0 | 0 | * | * | * | 1 | * | * | 1 | 0 |
| 10 | 1 | 0 | 1 | 0 | 1 | 1 | * | 1 | 1 | * | 1 | 1 | 1 | * | * | 1 | * | * | 1 | 1 |
| 11 | 1 | 0 | 1 | 0 | 1 | 1 | * | 1 | 1 | * | 1 | 1 | 1 | * | * | 1 | * | * | 1 | 1 |
| 12 | 1 | 0 | 1 | 0 | 1 | 0 | * | 1 | 0 | * | 1 | 0 | 0 | * | * | * | * | 1 | * | * | 1 | 0 |
| 13 | 1 | 1 | 0 | 1 | 0 | 0 | * | 1 | 0 | * | 1 | 0 | 0 | * | * | * | 1 | * | * | 1 | 0 |
| 14 | 1 | 1 | 0 | 0 | 0 | 1 | * | 1 | 1 | * | 1 | 1 | 1 | * | * | 1 | * | * | 1 | 1 |
| 15 | 1 | 1 | 1 | 1 | 1 | 1 | * | 1 | 1 | 1 | * | 1 | 1 | 1 | * | * | 1 | * | * | 1 | 1 |

Figure 39: Resulting automaton
\[ \delta(z, x) = z x_0 x_1 \lor x_1 \]

The KV-Diagramm:

\[
\begin{array}{cccc}
& & 0 & 1 \\
\delta & 0 & 0 & 0 \\
& 1 & 1 & 1 \\
\end{array}
\]

Figure 40: KV-diagram of \( \delta \)

The table:

<table>
<thead>
<tr>
<th>( z )</th>
<th>( x_1 )</th>
<th>( x_0 )</th>
<th>( \delta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 5: Truth table of the automaton

The graph of the automaton:

Figure 41: Graph of the automaton

NAND gates have been used to realize the structure. Therefore the DNF was transformed into a NAND structure by twice negating the DNF:

\[ \delta(z, x) = z x_0 x_1 \lor x_1 = \overline{z x_0} x_1 \lor x_1 \]

Figure 42: machine before the transformation

Figure 43: Different view of the automaton

Figure 44: Race condition

Figure 45: Race \( z = 0 \)

Figure 46: Race \( z = 1 \)

To simulate the race condition, the circuit was modelled and simulated in LT Spice. The simulation shows a race.

Figure 47: Simulated race situation
The circuit was realized in reality and the race was captured via an oscilloscope.

Then the transformation, which was presented in this journal, was made leading to the following structure, which does not have any contending feedbacks anymore. The only feedback this structure needs is nested in the RS-Buffer.

This structure does not show a race problem. To prove this statement the race condition of the old automaton has been set to the inputs. In figure 50 the state \( z \) and the rising edge of \( x_1 \) is shown. It’s clearly to see that the state doesn’t change, not even a fluctuation of the signal can be recognized. Therefore the transformed machine is determined and stable.

5 Conclusion and Future Work

An asynchronous function stable Mealy was transformed into a state stable Moore under the use of the RS-buffer with suitable coding. The transformation provides general asynchronous design rules, e.g. the isolation of critical situations via the hold function. Due to the stabilisation by RS-buffer with the hold function, no races will be propagated through the circuit. In this journal three examples have been presented, a 1-dimensional and two 2-dimensional. Although the stable 2-dimensional transformation has produced an asynchronous feedback, this feedback leads either to a confirmation or a hold of the RS-buffer. The feedback has to be seen as a starting condition which gets triggered by the input and a change of the feedback will only affect the RS-buffer if the input changes. This design guideline can be used for all circuit designs, to create a stable, reliable system. The goal of the design is to remove the outer feedback and nest it in the RS-buffer. The inputs have been coded as pseudo states and stabilized via the RS-buffer. The output function has also been designed in dual-rail. This leads to a circuit, that looks at first sight as a combinational circuit. With this method you can simplify complex automata and combine them. The results have to be simulated to get a detailed look at the behavior of the transformed Moore. As it was shown, one dimensional machines get stabilized by this design, but for more dimensional machines it is necessary to design stable functions and regard the transitions between the states, not only concentrating on one state. So a rule has to be found, which can check for instabilities between states and stabilize machines in these constellations. Additionally the formula for the Transformation can be programmed, to create an automated process of transformation. Additionally the output function \( \lambda \) has to be checked for functional Hazards and adapted to guarantee that they are no longer existing.

References


