Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 270 (2025) 397–406

Procedia Computer Science

www.elsevier.com/locate/procedia

29th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems (KES 2025)

An Analysis and Implementation of the Switching Sequence Table Method

Piotr Czekalski[a,*], Mirosław Borowiecki[a], Bolesław Pochopień[a], Godlove Suila Kuaban[b]

[a]*Faculty of Automatic Control, Electronics and Computer Science, Silesian University of Technology, Akademicka 16, 44-100, Gliwice, Poland*
[b]*Institute of Theoretical and Applied Informatics, Polish Academy of Sciences, Baltycka 5, 44-100, Gliwice, Poland*

Abstract

This paper details the algorithm implementation of the Switching Sequence Table method in asynchronous sequential circuit Boolean switching system design. The algorithm is theoretically analysed in terms of its computational complexity and validated through actual experiments. It discusses the multidimensional problem of various parameters and their impact on performance. It then presents analytical evidence that reducing this multidimensional problem to a standard, one-dimensional problem is feasible. The authors consider method challenges when implementing computing modules for the Switching Sequence Table method.

Keywords: circuit design; automation control; logic circuit; asynchronous state machine; asynchronous system design; finite state machine

1. Introduction

The analysis and design of sequential circuits or systems is an essential but challenging part of courses such as Logic Systems Design, Digital Electronics and Theory of Logic Circuit, which are part of the Electrical and Electronics Engineering, Computer Engineering, Telecommunication Engineering, and Control Engineering academic curricula. A sequential logic circuit is one in which the outputs at any instant are a function of both the present inputs and the previous outputs [1]. Since sequential logic circuits must record the previous output state, they are composed of combinational logic blocks (made up of logic gates) and memory blocks (made up of flip flops) [2]. Therefore, a sequential logic circuit is a logic circuit with storage or ”memory” [3]. The use of video games [4] and software packages in teaching digital circuit design is already a popular approach, widely adopted in Science, Technology, Engineering, Mathematics, and Arts (STEMA) disciplines.

Asynchronous digital systems are devices with wired logic, presenting elementary memory capabilities and performing a fixed algorithm, usually quite limited, but still more advanced than combinational digital circuits. It is

* Corresponding author. Tel.: +48-32-237-2577.
E-mail address: piotr.czekalski@polsl.pl

1877-0509
10.1016/j.procs.2025.09.158

impossible to describe their working cycle with a timing or state transition diagram. The simplicity of such systems is essential for critical industrial applications when microprocessor-based systems are considered unreliable. Those systems originating from the 1950s are still widely used.

The authors in [5] demonstrated a project-based learning approach in which they organised graduate students to use an open-source platform to discuss and solve practical Integrated Circuit (IC) design problems in groups. The authors in [6] implemented a web-based software package that uses Huffman's algorithm to design and analyse asynchronous sequential logic circuits.

This paper presents a software implementation of the Switching System Table (SST) method, a tool for designing asynchronous digital systems using finite-state machines.

2. Asynchronous System Design and SST Method

2.1. SST Explained

The SST method was presented in detail in [7] and extended even more in [8][9][10][11][12][13][14] but is not very popular in the English language literature, so here we present essential concepts. A brief introduction to the SST method is also given in [15], where the author presents an approach towards self-checking, error-proof digital, asynchronous system design. A high-level algorithm is presented in Fig. 1.

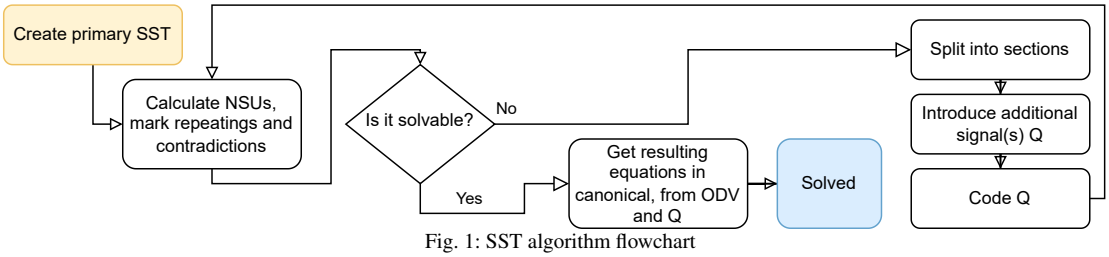


Fig. 1: SST algorithm flowchart

2.1.1. Primary SST

SST is a tabular representation of the timing diagram explicitly described by the switching sequence of the input and output signals. It can also be presented as a switching sequence or switching formula. In asynchronous finite state machines, only one signal can change at a time: the principle of the logical neighbourhood of the signal changes is strong. Switching sequence formula grammar has been presented in [6], and a sample sequence is present in Fig. 1.
In the primary SST, a row represents each input and output signal, and each row is assigned a unique value - an exponent of 2. The footer row contains the Numerical State of the Unit (NSU), a binary hash function that represents the current machine's internal state. The timing diagram illustrates a planar projection of the device's looped work cycle. SST can virtually start at any temporal location within the timing diagram. The most left Tact of all signals should be the same as the most right one to make a constant loop. The working cycle begins with the leftmost data column and ends when the loop closes. The following description references the Input Data Vector as $\overrightarrow{IDV}$ and the Output Data Vector as $\overrightarrow{ODV}$. A vector of internal states is referenced as $\vec{Q}$. $\overrightarrow{IDV}$ changes drive internal machine state that, in the case of the SST, is represented by $\overrightarrow{ODV}$ and eventually $\vec{Q}$. NSU is calculated using $\overrightarrow{IDV}$, $\overrightarrow{ODV}$ and $\vec{Q}$ vectors. For the primary SST, the following is true: $\left|\vec{Q}\right| = 0$.

2.1.2. Solvable vs Unsolvable SST

The first step is to evaluate whether any of the NSUs are repeating, and if so, whether the repetition is contradictory or non-contradictory. The contradiction appears when the signals composing $\overrightarrow{ODV}$ or $\vec{Q}$ present a different state for the same NSU value. In such a case, SST is unsolvable. To make it solvable, it is necessary to introduce additional signals into the $\vec{Q}$ vector, then control them appropriately to compose a closed loop of the working cycle and, this way, modify the binary mask to let NSU values present different (non-repeating) or non-contradictory repeating NSUs.

Introducing new digital signal(s) into the table requires splitting the table to introduce extra columns where $\vec{Q}$ signals are switched on and off.

2.1.3. SST Split Point Rules for the Unsolvable Case

Placing the split points is meaningful and should be done using the following rules. Assuming a split point splits SST into the two sections α and β, right after the Tact S, the following rules are valid:

- Split points should be introduced to split the regions with contradictory NSUs, so single sections α and β cannot contain contradictory NSUs.
- You can reuse sections alternatively if it solves the contradiction problem, e.g., α, β, then α and β, again.
- Split points cannot be located right after the contradictory state (NSU) column.
- It is safe to set the split point after the column with non-repeating NSU.
- Split points between two sections α and β, cannot be set after the Tact S contains non-contradictory repeating of the Tact that finishes section β or is present in any location in the section α.
  - The only exception is that one can be forced to split the SST immediately after the Tact S due to the first rule. In such a case, it is necessary to introduce as many splits between sections α and β as there are Tacts S repeating non-contradictory so that sections will be alternated. Everywhere there is a Tact S, there should be an ending of the section α and the beginning of the section β.

SST should be solvable once this process is completed and NSUs are recalculated in consideration of the introduction of $\vec{Q}$. If it remains unsolvable, then one of the rules has been broken, and the process should be rejected and repeated.

2.1.4. Regions and Their Coding

Once the table has been split into two or more regions, some regions may alternate, while others need unique coding. In the case, $\left|\vec{Q}\right| > 1$, nearby regions should be coded using logically binary neighbour values of the $\vec{Q}$.

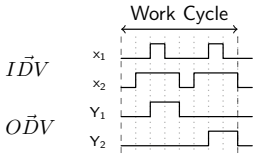
2.2. Multidimensionality in SST

A problem that is to be solved with the SST method is given with the following dimensions: number of Input Signals $\left|I\vec{D}V\right|$, number of Output Signals $\left|O\vec{D}V\right|$, number of Tacts n (discrete steps), complexity: a relation between switching signals, specific for the problem. When evaluating the algorithm's performance, it is essential to relate it to one dimension, namely, the best. In the SST method, it is possible to find upper and lower bounds for the problem complexity, thereby reducing it to a one-dimensional problem, as presented in Section 3.1.

2.3. Sample SST

The following section presents a sample problem and its solution using the SST method, illustrating the steps involved. The timing diagram is in Fig. ref timingdiagram1 and the corresponding Primary SST in Table 1. Switching Y_1 and Y_2 to high defines their "on-cycle" marked with arrows, and it is Tact 2 through 4 for Y_1 and Tact 8 through 10 for Y_2 respectively. Analysing "on-cycle" for both output signals, one can find that for Tacts 2 and 8, there are contradictory repeatings of the NSU value (in both equals 3) because it represents a different internal state of the system: in Tact 2, Y_1 is in its "on-cycle", and Y_2 is in its "off-cycle". In contrast, in Tact 8, it is the opposite. For this reason, it is necessary to split the table and introduce an additional binary internal signal, q. The division points introduced in the Primary SST were set after Tact 5 and after Tact 11, as present in the table 2. It inducts the insertion of two additional Tacts that control switching on and off of the extra, internal binary signal q in Tacts 5' and 11'.

The division (split) points introduced in the Primary SST were set after the Tact 5 and Tact 11, as present in the table 2. It introduces the insertion of two additional Tacts that control the switching on and off of the internal, binary signal q in Tacts 5' and 11', respectively. It makes SST solvable, so there are no contradictory or non-contradictory NSUs. Note, $\vec{Q}$ signal is included in contradiction evaluation along with $O\vec{D}V$.
SST method final solution describing internal state equations in a previous-step-next-step form, one can obtain quickly



x_2+x_1+Y_1+x_1-x_2-Y_1-x_2+x_1+Y_2+x_1-x_2-Y_2-

Fig. 2: Sample timing diagram and correlated switching sequence

Table 1: Primary SST

		Work Cycle												
Tact		**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**0**
$\mathbf{x_1}$	1	-		+		-				+		-		
$\mathbf{x_2}$	2	-	+				-		+				-	
$\mathbf{Y_1}$	4	-			+			-						
$\mathbf{Y_2}$	8	-									+			-
NSU		0	2	3_1	7	6	4	0	2	3_2	11	10	8	0

Table 2: SST with borders and additional state signal q

		Work Cycle													
Tact		**0**	**1**	**2**	**3**	**4**	**5**	**5'**	**6**	**7**	**8**	**9**	**10**	**11**	**11'**	**0**
$\mathbf{x_1}$	1	-		+		-					+		-			
$\mathbf{x_2}$	2	-	+				-			+				-		
$\mathbf{Y_1}$	4	-			+				-							
$\mathbf{Y_2}$	8	-										+				-
q								+							-	
NSU		0	2	3	7	6	4	20	16	18	19	27	26	24	8	0

via juxtaposing "on-cycle" and "off-cycle" NSUs in a canonical decimal form, for signals $\vec{Q}$ and $O\vec{D}V$, as on (1). Values are presented in the original order to facilitate easy reference to the SST (Table 2).

$$
\begin{aligned}
Y_1 &= \begin{cases} \sum & (3,7,6,4)_{x_1x_2Y_1Y_2q} \\ \prod & (0,2,20,16,18,19,27,26,24,8)^{1}_{x_1x_2Y_1Y_2q} \end{cases} \\
Y_2 &= \begin{cases} \sum & (19,27,26,24)_{x_1x_2Y_1Y_2q} \\ \prod & (0,2,3,7,6,4,20,16,18,8)_{x_1x_2Y_1Y_2q} \end{cases} \\
Q &= \begin{cases} \sum & (7,6,4,20,16,18,19,27,26)_{x_1x_2Y_1Y_2q} \\ \prod & (0,2,3,24,8)^{1}_{x_1x_2Y_1Y_2q} \end{cases}
\end{aligned}
\tag{1}
$$

2.4. SST vs Huffman

The SST method presents a one-step approach to obtaining a solvable solution (SST). The resulting state equations are obtained immediately for a selected class of problems without contradictions in the NSU. It is fixed to the structure; however, the output signals originate directly from the flip-flops' outputs, which constitute the device's memory block. On the other hand, Huffman's method allows for the design of either Moore's or Mealy's structures [6], which may

benefit from simpler memory block constructions.

$$T(r) = (r-1) + ... + (r-r) = \sum_{i=1}^{r}(r-i) = \frac{r^2 - r}{2} \tag{2}$$

3. The SST Algorithm and Challenges to Its Implementation

So far, there has been no software implementation of the SST method that is limitless in terms of the dimensions mentioned in Section 2.2. On the other hand, it is common to evaluate an algorithm's complexity in one dimension of the problem. It raises the question of correlation of upper and lower bounds among $\left|\vec{IDV}\right|$, $\left|\vec{ODV}\right|$ and n.

3.1. Reducing SST Problem Dimensionality

SST complexity generally relates to the number of changes and contradictory Tacts. The lower bound for the number of Tacts is limited by the total number of inputs and outputs, as each signal needs to change at least twice: $2 * n \geq \left|\vec{IDV}\right| + \left|\vec{ODV}\right|$. There is no upper bound, as an asynchronous sequential machine can be virtually infinitely complex. The primary SST structure also impacts algorithm complexity. When contradictory Tacts are located not so close to one another, it is easy to find split points, as they can be set close to the contradictory ones, which is best in non-repeating NSUs. The region merging process will not be time-consuming, and merging will occur only minimally. So, the lower bound is the case when there is just one contradictory pair, as in the example presented in this paper (Table 1). The upper bound of the complexity appears when there are many contradictory Tacts, and they are nested, one inside the other, e.g., in the following sequence: $a_1...b_1...c_1...c_2...b_2...a_2$. The maximum number of contradictory Tacts is less than half of the total number of Tacts n. The non-equality arises from the fact that even the most complex systems contain at least a couple of non-repeating Tacts; thus, these represent non-contradictory NSUs. The upper limit of the number of regions appearing after insertion of the split points is related to the number of contradictions, thus by the total number of Tacts n. The regions merging procedure takes longer if there are a complex number of split points; therefore, it is indirectly related to the number of contradictory states and can be upper-bounded by n. Regions coding to ensure logical neighbourhood can require the introduction of additional Tacts and eventually additional signals to the vector $\vec{Q}$ to handle non-logically neighbour transitions through the split point location of the neighbour regions. The worst-case scenario occurs when the region is near another region (alternating in the table). There is no direct relation to the n, but the number of regions cannot exceed the number of Tacts n. It does not describe explicit relations, however. To summarise, all steps of the SST solving process can be directly or indirectly (approximately) related to the total number of Tacts n.

3.2. Analytical Approach to the Computational Complexity

As presented in chapter 1, the solving process comprises several steps of complexity concerning n.
Building primary SST, given the switching formula, involves the following steps to be evaluated:

- Use of the RegEx *(\p{L}\d*[-\+])+* formula to decode $\vec{IDV}$ and $\vec{ODV}$ signals is (according to the [16] and [17]) of the linear complexity against the string length. In the case of the SST, the string length is limited by half of the total number of Tacts because each signal must be switched on and off. This step is of the linear complexity against n: $\Theta_1(n)$. Even if the RegEx formula contains an optional quantifier, it assumes exclusive tokens, reducing the back-reference problem to the marginal.
- Extraction of the unique $\vec{IDV}$ and $\vec{ODV}$ signals that is simple, linear loop, against n: $\Theta_2(n)$
- Sorting of the $\vec{IDV}$ and $\vec{ODV}$ signals and assigning binary masks to them. The most efficient sorting method is of the pessimistic complexity of $\Theta_3(o^2)$ and optimistic complexity of $\Omega_3(o * log(o))$, where $o = \left|\vec{IDV}\right| + \left|\vec{ODV}\right|$. As in the case of the asynchronous, sequential machine, the lower limit for the o is 2 (one input + one output), and the upper limit is $n/2$, the sorting complexity is in between $\Omega_3(1)$ and $O_3(n^2)$.

- Calculating of the NSUs and marking of the on-off regions is composed of the two nested iteration loops: the first, external iterator is linear against the number of Tacts n, multiplied by the number of outputs $\left|O\vec{D}V\right|$; thus its complexity is given by $\Theta_4(n * \left|O\vec{D}V\right|)$. Then, the internal loop runs over $O\vec{D}V$ signals directly related to the n, as the total number of input signals correlates with n. The complexity of this step can be estimated between $\Omega_4(n)$ linear, optimistic, and $O_4(n^2)$.
- The final step is to mark the repeating, contradictory, and non-contradictory states. It uses grouping to group all NSUs; groups are filtered to exclude non-repeating ones. Both actions are of linear complexity against n: $\Theta_5(n)$. The following requires comparing the r pairs in each of the g groups; thus, the number of comparisons needed is given by $T(r)$ as in formula (2). Then computational complexity is given by $\Theta_6(g * r^2)$. The number of groups g can equal at most the number of Tacts $n-1$ and at least (theoretically) 2. The number of groups g can vary between a meaningless constant value and a maximum of n (when SST is solvable); thus, g is linearly related to n. The amount of regions r is monotonically increasing with a decrease of the g, but this relation is not the opposite. With an increasing number of regions, there is also the growth of n because of the need to introduce the additional state signals $\vec{Q}$, so finally, $r < n$. The exact relation depends on the case and complexity of the timing diagram; still, it can be approached with the upper bound given as linear. Finally, for solvable SSTs, number of regions is $r = 1$, so optimistic computational complexity for this step is $\Omega_6(q * r^2) = \Omega_6(n * 1) = \Omega_6(n)$ and in pessimistic case ($r \to n$) it is $O_6(q * r^2) = O_6(1 * r^2) = O(n^2)$.

Finally, the SST creation table step's computational complexity, including contradictory and non-contradictory Tacts detection, is between $\Omega_I(n)$ and $O_I(n^2)$.

Finding borders and split points:

- Extraction of the unstable Tacts, containing contradictory NSU repeatings, uses a linear loop to iterate over all Tacts, so its complexity is of $\Theta_7(n)$. The number of such Tacts is referenced as s.
- Introducing borders concerning avoiding cycles uses the previous step and detects any contradictory repeating forwards and backwards. Nearby contradictory repeating can be at a fixed distance (1 Tact away) and eventually can be located farther away, but in any case, the distance is no further than $n-1$. Thus computational complexity is given by formula $\Theta_8(s)$ and limited with $\Omega_8(1)$ and $O_8(n)$.
  The following step is a procedure that introduces borders and split points. Once an unstable, contradictory Tact is identified in the previous activity, it is necessary to find nearby contradictory Tacts in both directions. The complexity of this procedure is linear with respect to n. So far, the dominating operation is the one that detects and extracts contradictory Tacts. Its complexity is determined by the number of contradictory repetitions t, linearly: $\Theta_9(t)$. The following activity involves a try-catch procedure to insert a split point between two contradictory Tacts. Optimistic complexity relates to the scenario where there is only one contradictory repeating, so it is $\Omega_9(1)$, and for the pessimistic case, we get $O_9(s * t)$.
  Considering that detecting s and t is sequential, we can refer to both using a generalised, upper limit as $\Theta_1 0t$.
- A process of identifying split points without respect to the closed cycle, according to the rules presented in section 2.1.3. It requires iterating over sub-vectors of NSUs located to the left and right of the considered contradictory Tact in the SST. The length of the sub-vector is referenced as u. Checking all of those rules in the pessimistic case yields $\Theta_1 1(u^2)$ and may need to be repeated in the worst case u times, as it involves shifts to the proposed split point u times, so finally, we obtain a complexity of $O_1 1(u^3)$. In the optimistic case, sub-vectors require just one scan, so their computational complexity is $\Omega_{11}(u)$.
  Then, we obtain $O_9(s * t) = O_9(s * u^3)$. Assuming the border case is when $u = n - 1$ (contradiction is most left Tact, and the limit of the sub-vector is the most-right or opposite situation), we obtain total pessimistic complexity for this step (not including closed cycle as presented below) given by formula $O_{12}(n^4)$.
- Inserting borders in the closed cycle to persist in a consistent work cycle is similar to those mentioned above. Still, its complexity is the opposite: the more complex the previous step, the simpler the following. We assume here the same complexity, and its pessimistic bound is $O_{13}(n^4)$.
- Introducing additional borders to avoid contradictions when facing non-contradictory states ending sections that appeared after splitting the Primary SST involves sorting and analysis. The sorting lower complexity bound

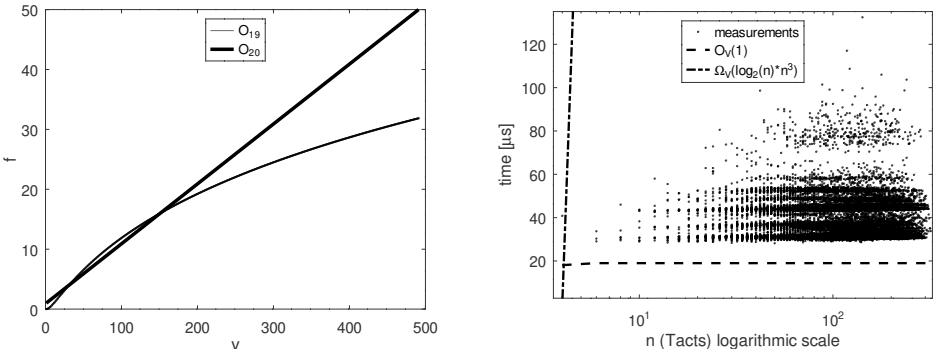
for split points is $\Omega_{14}(1)$ when there are just two split points (one contradictory NSU). The pessimistic case is limited by the number of Tacts in the SST, so it is $O_{14}(n^2)$.

- Next action is a preliminary assignment of the sections: it is linear against n concerning the number of contradictions, so the lower bound for complexity is $\Omega_{15}(1)$ and upper $O_{15}(n)$.
  The following action checks, along with rules in subsection 2.1.3 and in detail against the last rule on split points and repeating Tacts. This operation proposes an additional split point to ensure rules are fulfilled. It involves scanning any Tact against any other Tact in the section, which has a complexity of n^2. In the pessimistic case, this operation is executed for each contradictory state within and for each section. As all those relate directly to the n, the only known pessimistic complexity approach is $O_{16}(n^4)$. The same complexity applies to the aforementioned last rule and its exception, so it is $O_{16}(n^4)$.

In summary, the optimistic case is when there are no problems with split points, and insertions are done in one step, the complexity is of $\Omega_{II}(n)$, and the pessimistic case is $O_{II}(n^4)$.
Merging of the sections is a step in which regions are grouped to ensure $\left|\vec{Q}\right|$ is minimised. It is necessary to check for no contradictory NSUs inside to merge sections. As the number of Tacts is more than 2 for any reasonable asynchronous sequential circuit, in the pessimistic case, it involves the number of checks given by the formula $T(n) = \binom{n}{2} = \frac{n^2-n}{2}$. The algorithm behind this step has several optimisations, e.g., it removes checks for nearby sections; still, it does not impact the performance. So far, the only known upper bound for the complexity of this step is $O_{III}(n^2)$. An optimistic case appears when there are only two sections in the SST, so there is no possibility to merge them; thus, optimistic complexity is given simply by $\Omega_{III}(1)$.

Once borders are set up and regions are merged, it is necessary to binary code them, to switch on and off $\vec{Q}$ digital signals appropriately, and to ensure the logical neighbourhood of the adjacent regions. The coding algorithm utilises: a split-sequence list (of v items), and a merged regions list (of w items). To solve this step, it is necessary to build a graph of adjacent regions and, based on it, estimate boundaries as $\left|\vec{Q_a}\right| \in [\lceil log_2 w \rceil; \lceil log_2 v \rceil]$. Raw graph construction exhibits linear complexity for v. A graph relaxation is introduced whenever a node has several neighbours that exceed the current binary coding capacity, given as $2^{\left|\vec{Q_a}\right|}$. In the optimistic case, there will be no such nodes at all. In the worst-case scenario, it becomes the most significant factor in determining the overall computational complexity of the step. Its process is reorganising the graph to push the node over the neighbour one unless coding capacity is suitable, so it is linear against v. This process repeats once the node has up to $w-1$ neighbours. During a single iteration, up to $Q_b - 1$ nodes can be reorganised (shifted), where Q_b denotes the current value of $\left|\vec{Q_a}\right|$, and it grows when there is no possibility to relax graph nodes anymore due to the lack of coding capacity. The number of iterations for a single node is given by the formula $\frac{w-1}{\left|\vec{Q_b}\right|-1}$, so it is linear complexity vs w. Single node relaxation in the pessimistic case is then given by $O_{17}(w * v)$. When Q_b needs to be increased in the case mentioned above, building a whole raw graph from scratch and repeating the relaxation process is necessary. The maximum number of such cases is $log_2 c - log_2 w - 1$. This formula can be approximated with $log_2 v$ as in the pessimistic case, the number of regions before merging (v) will be of a magnitude higher than the number of merged regions w. Finally, the relaxation of the graph presents $O_{18}(log_2 v * v * w^2)$ complexity.
Binary coding requires an iterative process to visit all nodes and assign unique coding, ensuring the logical neighbourhood of the nearby nodes. The maximum number of iterations is then given as $log_2 v - log_2 w$. As mentioned before, $log_2 v >> log_2 w$, so that it can be estimated as $log_2 v$ iterations. The graph pathfinding algorithm in this step in the pessimistic case requires visiting x_b^2 nodes, where the upper bound of $x_b = log_2 v - 1$. Finally, the maximum number of nodes visited during the coding process is $(log_2 v - 1)^2$, so the pessimistic complexity is $O_{19}((log_2 v)^2)$. Additionally, a few nodes can be shifted if the coding capacity is exceeded during graph browsing. This operation is linear against v; in the worst case, it is $O_{20}(v)$. To estimate which of the two O_{19} and O_{20} is dominant, a function comparison drawing is helpful (Fig. 3 (a)). Assuming v is a reasonably large number, in theory, $v \to \infty$, a linear function brings a more pessimistic approach. Finally, there are $log_2 v$ repetitions, each containing v transitions between transition nodes with v checks on node insertion. This process defines pessimistic complexity as $O_{21}(log_2 v * v^2)$. Comparing O_{21} and O_{18}, it is necessary to assume a pessimistic approach when w is relatively large, and so its lower bound is $w^2 > v$, so $v > w$. Finally, the computational complexity of the whole coding process is limited between the optimistic case $\Omega_{IV}(v)$ and

$O_{IV}(log_2 v * v^3)$. As upper limit for v is n, then finally, algorithm complexity regarding n is limited in between given by $\Omega_V(v)$ $O_V(log_2 n * n^3)$.



(a) (b)

Fig. 3: (a) Linear and sq. of log2 function comparison, (b) Primary SST table building complexity, real vs theoretical

The final step in solving the SST is to introduce coding into the table that involves each NSU recalculation as new $\vec{Q}$ signals appear in the table. This operation is as simple as browsing the table and updating NSUs, so it is linear against n: $\Theta_{VI}(n)$. Analysing the presented SST solution steps, the total SST solving algorithm is $\Omega(n)$ in the optimistic case and $O(n^4)$ in the pessimistic one.

4. The Implementation of the SST Computing Module and Website

Implementation was done in a distributed model (frontend + backend). The backend features a switch for a rich response tailored to educators and a minimal response (raw, numerical results) for benchmarking purposes, and is stateless. The result is presented in canonical, decimal form. The frontend calls the Kazakov [18] logical function minimisation service to convert the result into equations. The use of Boolean function minimisation helps transform the outputs of the SST method, which come in a canonical, decimal form, such as those in (1), into equations. The SST computing module requires input in the form of a switching sequence, as presented in Fig. 2. A dedicated data generator was prepared to randomly generate a dataset of 25600 cases with variable $\left|I\vec{D}V\right|$ and n.

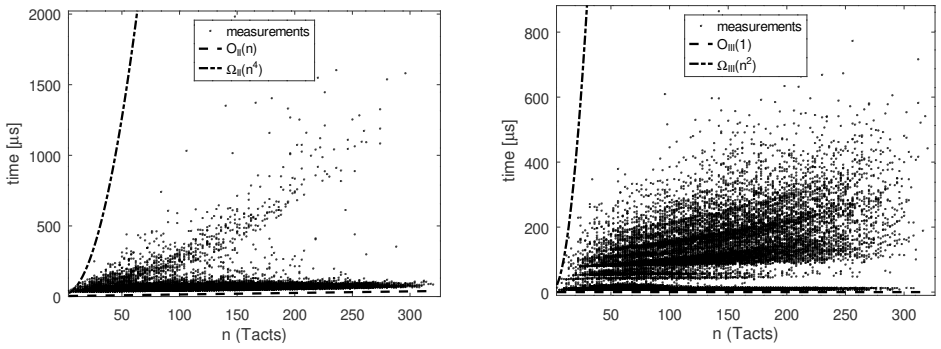
5. Model Verification

The following figures represent the cloud point of the averaged measurements (10 runs per sample) for a variety of input conditions, including variable n, $\left|I\vec{D}V\right|$ and $\left|O\vec{D}V\right|$. There are also theoretically inducted functions that represent lower and upper bounds for the computational complexity of each step of the SST-solving process.

Primary SST building (I): SST construction times for 25600 samples of different numbers of n, $\left|I\vec{D}V\right|$ and $\left|O\vec{D}V\right|$ (presented as related to n), are visualised in the form of cloud points in Fig. 3 (b). Theoretically, inducted lower and upper bound complexities are present with a solid and a dashed line on the same plot. One can observe that actual measurements confirm theoretically induced complexities, with a remark that the upper bound (pessimistic) complexity $\Omega_I(n^2)$ appears to be too broad; a majority of the samples exhibit a linear relation.

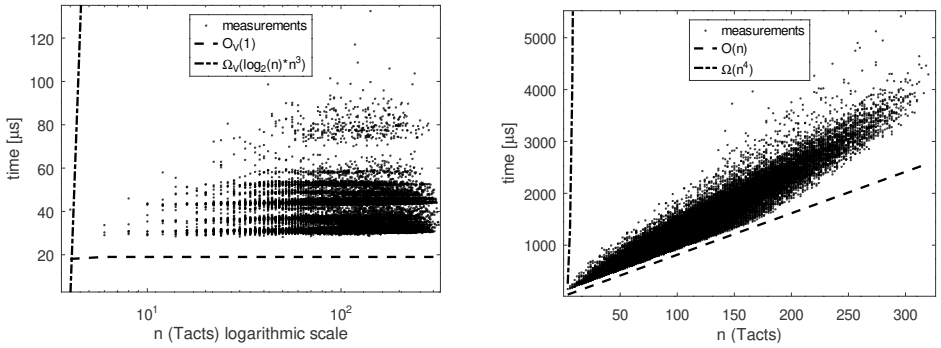
Split points detection (II): The computational complexity of the estimation of the borders (split points) algorithm was estimated to be in between $O_{II}(n)$ and $\Omega_{II}(n^4)$.
Representation of both functions and measurements in the form of the cloud point, as shown in Fig. 4 (a), shows that the theoretically induced complexity bounds were acknowledged with the experimental results.

Sections merging (III): Sections (regions) merging experimental results prove that the upper and lower bounds induced theoretically are correct for complexity. Cloud point distributes in between $O_{III}(1)$ and $\Omega_{III}(n^2)$, as present on the Fig. 4 (b).



(a) (b)

Fig. 4: (a) Split points and sections detection complexity real vs theoretical, (b) Sections merging complexity, based on the number of Tacts n



(a) (b)

Fig. 5: (a) Sections merging complexity based on number of Tacts n, (b) SST solving complexity (full algorithm) real vs theoretical

Coding and NSU updating (IV & V): The coding step was the most problematic regarding the theoretical induction of complexity, as the relation between several regions v and the number of Tacts n was not straightforward. For this reason, two benchmarks were performed, one against v and the other against n. It is the only step in the SST solving benchmarking that was benchmarked twice. In the case of the v, inducted lower bound complexity as $\Omega_{IV}(v)$ and upper pessimistic complexity given as $O_{IV}(log_2(v) * v^3)$, a sparse cloud point fits the area bordered by upper and lower bound functions. A cloud point in the case of the regions is sparse, as several areas v in the case of the majority of SST generated with the SST generator algorithm are no more than 25. Fig. 5 (a) presents complexity concerning the n. As one can observe, the assumed pessimistic, upper bound complexity, given by the formula $\Omega_V(log_2(n) * n^3)$, is far too strict and too steep. Authors expect that the inducted transition from v to n is the reason for the resulting overestimation of the pessimistic complexity induction. Still, this boundary is valid, even if too strict, compared to the cloud point. The NSU updating process is linear concerning n.

6. Summary

The SST algorithm has been used primarily in manual work since the 1950s, long before the advent of software implementations of the design algorithms and the PC era. SST's software implementation proved to be surprisingly complex, raising numerous questions, particularly regarding the boundaries of algorithm complexity and their measurement. The induction of the relation between the algorithm complexity and the number of Tact was a crucial step towards further analysis and experiments. Overall, algorithm complexity boundaries are determined by the chunks presented as their upper limits. Theoretically, it was inducted as $O(n)$ for the optimistic case and $\Omega(n^4)$ for the pessimistic case. However, in Fig. 5 (b), it is observable that $\Omega(n^4)$ is too strict and too steep for an upper bound, still limiting the cases. Experiments acknowledged the algorithm's analytical complexity and demonstrated the feasibility of dimension reduction.

Acknowledgements

This publication was supported by the Department of Computer Graphics, Vision, and Digital Systems under the statutory research project (Rau6, 2025), Silesian University of Technology (Gliwice, Poland). It was also partially supported by the Erasmus+ KA2 HED "IOT-OPEN.EU Reloaded" project no 2022-1-PL01-KA220-HED-000085090.

Appendix

Table 3: Terminology and abbreviations.

Term / Abbreviation	Description
Primary SST	A tabular representation of the switching sequence, without any additional signals, can be either solvable or non-solvable.
$\overrightarrow{IDV}$	Input Data Vector: state of all digital inputs of the system
$\overrightarrow{ODV}$	Output Data Vector: state of all digital outputs of the system
$\overrightarrow{Q}$	Additional Signals Vector: state of all internal signals, added during the solving process of the SST
NSU	Numerical State of the Unit: a hash function (binary mask) representing the current state of the system
Tact	A single column in the SST that corresponds to the current state of the system (including state of the $\overrightarrow{IDV}$, $\overrightarrow{ODV}$ and $\overrightarrow{Q}$)
Contradictory Tacts	A pair of Tacts that have the same NSU value, but represent different states (different values of $\overrightarrow{ODV}$ and $\overrightarrow{Q}$)

References

[1] A. Saha and N. Manna, *Sequential Logic circuits*. Hingham, Massachusetts: Infinity Science Press LLC, 2007.
[2] A. K. Singh, M. Tiwari, and A. Prakash, *Synchronous(Clocked) Sequential Circuits*. New Delhi: New Age International Limited, 2006.
[3] G. H. Mealy, "A method for synthesizing sequential circuits," *The Bell System Technical Journal*, pp. 1045–1079, 1955.
[4] M. Oren, S. Pedersen, K. L, and Butler-Purry, "Teaching digital circuit design with a 3-d video game: The impact of using in-game tools on students' performance," *IEEE Transactions on Education*, vol. 64, no. 1, pp. 24–31, 2021.
[5] X. Yang, "An approach of project-based learning: Bridging the gap between academia and industry needs in teaching integrated circuit design course," *IEEE Transactions on Education*, vol. 64, no. 4, pp. 337–344, 2021.
[6] P. Czekalski, K. Tokarz, and B. Pochopień, "A modern approach to the asynchronous sequential circuit synthesis," *Theoretical and Applied Informatics*, vol. vol. 26, no. No 1-2, pp. 25–37, 2014. [Online]. Available: http://journals.pan.pl/Content/118515/PDF/AModernApproachtotheAsynchronousSequentialCircuit.pdf
[7] J. Siwiński, *Układy przełączające w automatyce*. WNT, 1980.
[8] U. Stańczyk, K. Cyran, and B. Pochopień, *Theory of Logic Circuits Volume 2 - Circuit Design and Analysis*. Gliwice: Wydawnictwo Politechniki Śląskiej, 2007.
[9] H. Małysiak and Et Al., *Teoria automatów cyfrowych. Laboratorium*. Gliwice: Wydawnictwo Politechniki Śląskiej, 2002.
[10] H. Kamionka-Mikuła, "Optymalny sposób wprowadzania elementów dodatkowych do nierozwiązalnej tablicy kolejności łączeń." *Archiwum Automatyki i Telemechaniki*, vol. XXII, no. 4, pp. 41–53, 1977.
[11] H. Kamionka-Mikuła, "Algorytm wyznaczania miejsc zmian elementów dodatkowych w tablicy kolejności łączeń," *Archiwum Informatyki Teoretycznej i Stosowanej*, vol. 8, no. 3-4, pp. 26–38, 1997.
[12] H. Kamionka-Mikuła, H. Małysiak, and B. Pochopień, *Układy cyfrowe. Teoria i przykłady.* Gliwice: PKJS, 2004.
[13] H. Kamionka-Mikuła, H. Małysiak, and B. Pochopień, *Teoria układów cyfrowych, T. I. Układy kombinacyjne.* Wydawnictwo Politechniki Śląskiej, 2015.
[14] H. Kamionka-Mikuła, H. and Małysiak and B. Pochopień, *Teoria układów cyfrowych. T. II. Układy sekwencyjne.* Gliwice: Wydawnictwo Politechniki Śląskiej, 2018.
[15] H. Małysiak, "The synthesis of asynchronous sequential error detection circuits with the switching sequence table method," *Archiwum Informatyki Teoretycznej i Stosowanej*, vol. T. 10, z. 1-2, pp. 25–33, 1998.
[16] Microsoft, "MSDN - Backtracking in Regular Expressions," 2021. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/standard/base-types/backtracking-in-regular-expressions
[17] S. Fenner, D. Padé, and T. Thierauf, "The complexity of regex crosswords," *Information and Computation*, p. 104777, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0890540121000924
[18] W. D. Kazakov, "Minimization of logical functions of a large number of variables," *Automation and Telemechanics*, vol. 23, no. 9, pp. 1237–1242, 1962.