

Solving the Fundamental Problem of Digital Design – A Systematic Review of Design Methods

Martin Delvai, Andreas Steininger, Wolfgang Huber
Vienna University of Technology
email: {delvai,steininger,huber}@ecs.tuwien.ac.at

Abstract

As the synchronous design paradigm is approaching its limits asynchronous design has been celebrating a comeback during the last decade. Researchers have developed and are still developing circuit structures and design methods, which seem to be quite different. A closer look, however, reveals that in one or the other way all methods contribute to solving the same fundamental design problem.

In this paper we use a simple circuit model to figure out what the fundamental design problem actually is and to highlight its roots. We show how each of the related sub-problems can be conceptually solved in the time domain and in the information domain. Having this model in mind we finally develop a common framework to classify the most popular asynchronous design methods and figure out which sub-problem they actually solve and in which respect they differ from each other.

1. Introduction

Asynchronous design approaches have been studied for many decades, and a lot of publications exist in this field (see [4][3][1], for example). Most of them are dedicated to theoretical properties and abstract design techniques, while relatively few are concerned with implementation issues. Being hardware designers we soon found out that a comparison of the existing design techniques, however, looks completely different, as soon as implementation issues are taken into account as well. Moreover, we soon experienced that using one or the other method alone does not solve the whole design problem. It is important to understand which issues are being solved by a particular method and which are not. And for this purpose it is vital to identify what the problem of digital design actually is about. Although this question may sound trivial in the first place, it is

hard to find one competent and comprehensive answer to it in the literature.

Therefore the contribution we want to make in this paper is not a new technique, but rather a different – namely hardware-related and systematic – view on existing techniques. In contrast to existing publications [10][1] where the authors provide an “all-in one” design solution, we analyze and highlight all subproblems separately. After briefly providing the required terminology in Section 2 we identify in Section 3 the fundamental problems of digital design that make the employment of specific design techniques necessary from a hardware point of view. In Section 4 we identify possible conceptual approaches to overcome these problems. In Section 5 we study the synchronous design paradigm and the most relevant clockless design methods in the light of these strategic considerations. This enables us to analyze the basic solution principles they employ and to compare apparently different methods. Finally Section 6 concludes the paper.

2. Terminology

2.1. Signal model

We call the input of a Boolean logic function an *input vector*. It is constituted by a number of *signals* – one for each input. The Boolean logic function defines a specific mapping from the input vector to an output signal. This mapping is implemented by a *logic function unit*. Often several Boolean logic functions are applied to the same input vector in parallel, creating several output signals with a common semantic context (datapath elements like adder, e.g.). We use the term logic function unit in a broader sense to describe the implementation of this set of Boolean functions as well.

We call the smallest unit of information conveyed on a signal a *bit*, and the (consistent) vector of bits conveyed on an input vector a *data word*. A signal

can be physically represented by one or more *rails*, whose *logic levels* define the signal's *logic state*. The two mandatory logic states of a signal are "high (HI)" and "low (LO)", but states such as "NULL", "illegal" or "in transition" are conceivable and sometimes used as well. A *signal-level code* relates the logic levels of the rails – viewed as a vector that represents a signal – to the logic state of the corresponding signal. For the digital rails we consider here the logic level may be either "0" or "1". In the conventional single-rail encoding a signal is represented by only one rail whose logic level is directly mapped to the signal state.

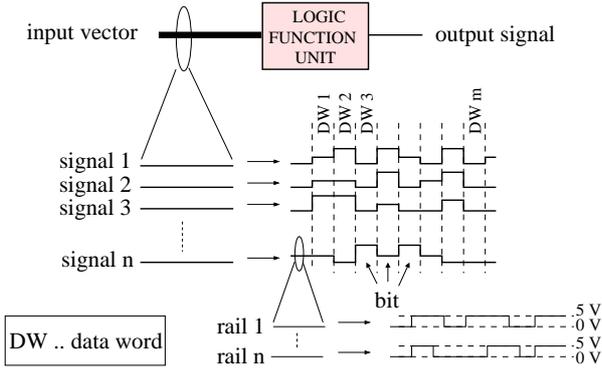


Figure 1. Terminology

We call an input vector *consistent* at instant t_i , if the states of all its signals belong to the same context at instant t_i , i.e. if they represent one single valid data word, and inconsistent otherwise. We also call the involved signals consistent under this condition. We call a signal *valid* at instant t_i , if its state at instant t_i is the stable result of a logic operation performed on a consistent input vector, and invalid otherwise.

2.2. Data flow model

From the point of view of information flow every logic function unit FU is preceded by a data source SRC that provides its input vector, and followed by a data sink SNK that further processes its output signal or vector (maybe in context with the outputs of other logic function units) as illustrated in Figure 2. Both data sink SNK and source SRC represent an abstraction of the remaining circuit and may internally consist of further logic function units. We call an output bit b_y of FU *consumed* by SNK at t_i , if b_y is still properly considered in the flow of information in SNK , regardless of whether b_y is overwritten by a subsequent bit b_z after t_i or not. Usually consumption implies the transfer of the information to some storage element.

We call an information flow *lossless*, if all pertaining

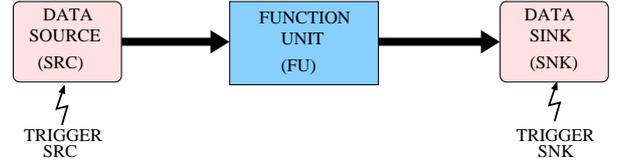


Figure 2. Circuit model

bits are properly consumed at all instances (i.e. points in time) t_i . We call a signal path P_k lossless, if the information flow along P_k is lossless. To guarantee this, SRC and SNK have to be coordinated by appropriate trigger events.

2.3. Timed data flow model

Considering the temporal relations and delays involved in the data transfer between SRC and SNK , we have to extend our model by timing issues. Figure 3 illustrates this model.

Upon a source trigger event $TRG_{SRC,x}$ at instant $t_{TRG SRC,x}$ a data word $DW_{SRC,x}$ is requested from SRC . As soon as SRC is ready, it will react by issuing $DW_{SRC,x}$ which will – after some delay – become visible and consistent at the output of SRC at instant $t_{issue,x}$. We call the interval between trigger event ($t_{TRG SRC,x}$) and actual visibility of the consistent data word $DW_{SRC,x}$ at the output ($t_{issue,x}$) the *issue delay* Δ_{issue} . Next $DW_{SRC,x}$ propagates to the logic function unit FU where it is processed. The corresponding result, $DW_{FU,x}$, propagates from the output of FU to SNK , passing SNK 's input logic, until it is finally available as a consistent data word $DW_{SNK,x}$ within the sink and hence ready for storage at instant $t_{SNKrdy,x}$. We subsume all these delays between $t_{issue,x}$ and $t_{SNKrdy,x}$ in the *processing delay* $\Delta_{process}$. At some point in time $t_{SNKtrg,x} > t_{SNKrdy,x}$ the sink trigger $TRG_{SNK,x}$ event will – after some inherent delay – cause $DW_{SNK,x}$ to be actually consumed at instant $t_{consume,x}$. We call the interval between $t_{SNKrdy,x}$ and $t_{consume,x}$ the *consumption delay* $\Delta_{consume}$ and the interval between $t_{SNKtrg,x}$ and $t_{consume,x}$ the *sink trigger delay* $\Delta_{SNKtrig}$.

At instant $t_{issue,x+1} > t_{consume,x}$ it is safe to trigger the next data word $DW_{SRC,x+1}$ to be issued by SRC . We call the delay until this actually occurs (i.e. the interval between $t_{consume,x}$ and $t_{issue,x+1}$) the *cycle delay* (Δ_{cycle}). Notice that $DW_{SNK,x}$ does not necessarily become invalid immediately at $t_{issue,x+1}$ but only after $DW_{SRC,x+1}$ has propagated through FU to SNK . We describe this conservation of the previous data word by an *invalidity delay* ($\Delta_{invalidity}$). As a consequence the system designer may decide to choose a *negative cycle*

delay, thus increasing throughput by issuing the next data word $DW_{SRC,x+1}$ already before the current data word $DW_{SNK,x}$ has been consumed. Notice that all delays may vary with implementation, operating conditions and even with data and hence some margins have to be considered in the timing.

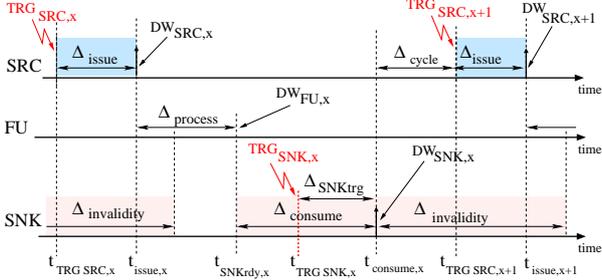


Figure 3. Timed circuit model

3. The Fundamental Design Problem

The fundamental problem of digital logic design can be subsumed as follows: *Ensure a lossless information flow in the system.* Under this fundamental constraint systems are typically optimized for maximum information throughput and implementation efficiency. In order to achieve these aims the designer has to coordinate the triggers of source and sink appropriately. In context with the timed data flow model presented above, this implies the following:

- (i) The *sink trigger* $TRG_{SNK,x}$ must not be activated before $t_{SNKrdy,x}$ (ensure losslessness). Less formally speaking this means that a new data word may only be captured by the sink after it has become consistent. To achieve maximum throughput capturing should, however, occur as soon after $t_{SNKrdy,x}$ as possible. As a consequence, every design method must allow a judgement of validity and consistency of a data word in one way or the other (**fundamental requirement 1**).
- (ii) The *source trigger* $TRG_{SRC,x}$ can safely be activated after $t_{consume,x}$ to guarantee losslessness, which means that the next data word may be issued only after the previous has been consumed. For maximum throughput it is desirable to place the trigger right after $t_{consume,x}$ or even prior to this instant (negative cycle delay). With respect to the design method this requires the existence of some kind of information feedback from the sink to the source (**fundamental requirement 2**).

Figure 4 illustrates these requirements. In prac-

tice requirement (2) has turned out to be relatively easily fulfilled by an appropriate circuit structure (micropipeline, e.g.), while the assessment of validity and consistency (condition (1) above) is a notorious problem that we will analyze more closely in the following.

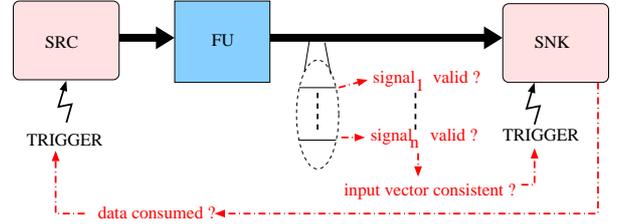


Figure 4. Fundamental design problem

3.1. Formal incompleteness of Boolean logic

Boolean logic defines functions on a high abstraction level. In essence a Boolean function is a time-free mapping (truth table, e.g.) from an input vector to an output signal. The output continuously reacts to any change of the input – there is no such thing as a trigger. This implies that only consistent data words are applied to the logic function. In fact Boolean logic does neither provide any means for expressing or considering validity or consistency nor for expressing temporal relationships. For this reason Boolean logic is called “formally incomplete” in [2]. Unfortunately, due to signal delay and signal skew, none of the above assumptions is fulfilled in a physical circuit implementation.

In conclusion, Boolean logic does not solve any of the fundamental requirements and hence does not contribute to solving the fundamental design problem in the first place. Still, Boolean logic is the established way of describing logic operations. All design methods have to compensate for this shortcoming in one way or another. In section 4 we will analyze how different design styles solve this problem. Before we do this we will analyze the roots of the problem in more detail.

3.2. Signal delay

Two constituents of signal delay are commonly distinguished, namely gate delay and interconnect delay. While gate delay is mainly determined by technology and fan-out, interconnect delay depends on many parameters that are specific to a given signal path, such as drive strength of the sender’s output, capacitance and resistance of potential switch elements or vias along the wire, length and physical arrangement of the particular wire, and capacitance of the connected inputs. In addition, overall signal delay is a function of the operating

conditions (supply voltage, temperature). As a consequence the time it takes an output to become valid is non-zero, which is contradictory to the assumptions made by the Boolean logic.

3.3. Signal skew

As a consequence of the uncertainties with respect to signal delays no pair of signals will exhibit exactly the same delay. The difference of delays within signals of the same input vector is called *skew*. Notice that by definition skew distorts the temporal relations between signal transitions. As a result the assumption that the transition from one data word to the next one will occur at once (as implied by the continuous, time-free definition of a Boolean logic function) is unrealistic. The edges on the individual rails will rather arrive sequentially, creating inconsistent intermediate signals and input vectors that (temporarily) cause invalid outputs. In this sense the skew disproves the validity and consistency assumption made by the Boolean logic.

4. Strategic Options

In Section 3 we pointed out that it is an essential task of every digital design method to ensure that only consistent and valid data is consumed by the data sink and that the source is synchronized to the sink such that no data is getting lost. In the following we will identify two basic domains where this can be done. Notice, that it is not required to solve all aspects of the fundamental design problem in one domain – mixed solution are also possible.

4.1. Time domain

Having figured out *timing* issues – namely delay and skew – as one root of the fundamental design problem, one consequent solution is to compensate for their undesired effects directly in the time domain.

Concerning the validity and consistency requirement we can simply determine all relevant delays between $t_{TRG_SRC,x}$ and $t_{SNK_rdy,x}$. The sum of these delays constitutes the minimum time we have to wait after $TRG_{SRC,x}$ until we can safely apply $TRG_{SNK,x}$:

$$t_{consume,x} \geq t_{TRG_SRC,x} + \Delta_{issue,x} + \Delta_{process,x} \quad (1)$$

Likewise we can relate $TRG_{SRC,x+1}$ to $TRG_{SNK,x}$:

$$t_{TRG_SRC,x+1} \geq t_{consume,x} + \Delta_{cycle} \quad (2)$$

Like above Δ_{cycle} must be determined by means of a delay analysis of a given implementation.

Based on this strategy we can use two different approaches for controlling the triggers (see Figure 5):

1. The use of coupled *timers* that – started with the one trigger event (source or sink) – generate the respective other trigger event (sink or source) after the appropriate delay.
2. The use of a common time reference from which periodic triggers for both, *SRC* and *SNK* are derived with an appropriate phase difference.

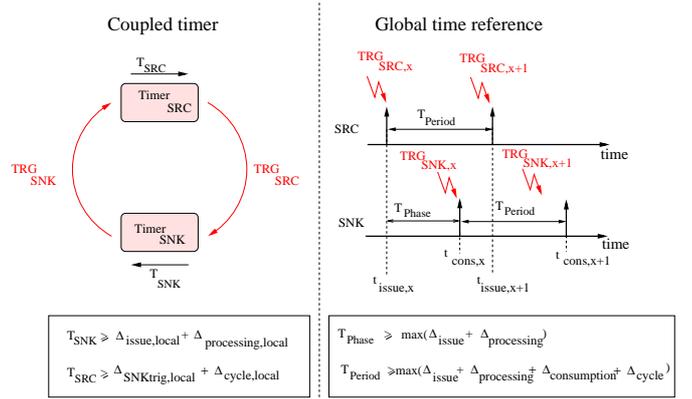


Figure 5. Solutions in the time domain

These approaches are capable of solving the fundamental design problem on all levels, since all delays have passed and the circuit is stable at the trigger instants. In some sense we have thus overcome the formal incompleteness of Boolean logic by condensing the missing information on validity and consistency into the timer settings (T_{SNK} , T_{SRC} , or T_{period} , respectively; further referred to as T_*) and using dedicated control signals to convey this information between source and sink. However, there is a substantial difference between those approaches: While the coupled timers are matched to *local* delays between each source/sink pair, the use of a global time reference necessitates to consider the *entire* circuit for calculating T_{period} and T_{phase} . Notice, however, that both approaches *postulate* that the input vector will be consistent and valid after a calculated delay, in fact have no means to *directly* assess consistency and validity. As a result the determination of the timer settings T_* becomes a crucial issue. Two essentially contradicting arguments guide their choice:

1. *Restrictive assumptions*: It is not possible to determine any finite value for T_* without making assumptions on the implementation. The higher T_* the fewer assumptions must be made and the fewer

restrictions apply to the implementation and the safer we can assume our losslessness property.

2. *Performance*: Obviously T_* has a negative impact on throughput in terms of data words per second. In order to minimize the resulting performance degradation, a minimal T_* should be strived for.

So ultimately the choice of T_* turns out to imply a tradeoff between performance and assumptions that have to be made on (and finally be met by) the implementation. Many models and techniques exist that allow to determine T_* for a given circuit topology and technology. However, since delay and skew depend on many parameters, an "aggressive" choice of T_* towards maximum performance compromises the robustness of the circuit.

4.2. Information domain

Alternatively we can tackle the other root of the problem, namely the formal incompleteness of Boolean logic. Different methods are available here to enforce the different fundamental requirements:

Validity: Recall from Section 2.1 that a signal is called valid if it is the stable result of a Boolean function performed on a consistent data word. There are several possibilities to judge on the validity of a signal: (i) **Ensuring continuous validity:** If we can manage to build the logic function unit in such a way that it produces only valid outputs, judgement of the output signal's validity becomes trivial. A function unit of this type must change its output only in response to a consistent input word (Notice that ensuring continuous validity does not enforce continuous consistency, since the combination of valid signals pertaining to a different context does not yield a consistent data word). To this end it must (a) be able to judge on the consistency of the input word and (b) hold the last valid output signal during transient phases of inconsistent inputs. This obviously requires some kind of storage element for each functional logic unit.

Even with an input perfectly changing from one consistent state to the other, skew within the function unit may cause invalid transient spikes at the output signal. Therefore special care must be taken for the design of the function unit. This causes a trade-off with respect to the partitioning of a circuit into functional blocks: A coarse-grained partitioning into few logic function units saves storage elements, while a fine-grained partitioning facilitates better control of skew effects.

If the signal is composed of more than one rail, continuous consistency in the rail domain is a necessary

condition for continuous validity in the signal domain. This can be ensured by the employment of a grey-code on the rail level, e.g.

(ii) **Extending the signal code** Another approach to make validity visible is to establish a more comprehensive alphabet than the binary Boolean logic (by using more than one rail per signal, e.g.) and to define a subset of all expressible codewords, which are considered as "valid". In contrast to the previous approach, direct transition from one valid codeword on the rail-level to the next is not mandatory, (invalid) intermediate states are allowed, since they can be identified. In other words, if a valid codeword has been reached after a number of single transitions on k of n rails of a signal, there must be no other valid codeword that can be reached by transitions on the remaining $n - k$ rails. This allows us to unambiguously identify when a codeword is complete, irrespective of the order in which the transitions occur. The transition to the next codeword must include another transition on at least one of the k rails. The same condition – though in a different formulation – has been given in [12].

(iii) **Current sensing:** This method exploits the fact that transient effects in a circuit are associated with current flow. Unfortunately, however, the reverse is not necessarily true: The lack of dynamic current flow is indeed a reasonable indication that the inputs are stable (and hence consistent?) and the output is stable and hence valid. Without any restrictions on the delays, however, it may well happen that one slow rail transition arrives after the circuit has been considered stabilized. Another problem with this method is the lack of an event separating two successive identical data words, which substantially complicates consistency judgement. Finally the inclusion of analog circuitry for the current sensors causes additional technological efforts.

Consistency: Imagine the situation depicted in figure 6: SNK has an input vector composed by two signals, each of which carrying a valid codeword in the rail domain. This does not necessarily imply that the input vector is consistent, the bits on the signal could even belong to different contexts. Notice, validity does not imply consistency, but consistency requires validity.

To judge consistency, a circuit must be able to differentiate between two consecutive bits carried on a signal line, even if they hold the same information. In other words we have to choose a signal level code which relates information to context. So in order to be applicable for our purpose, a coding scheme must meet two

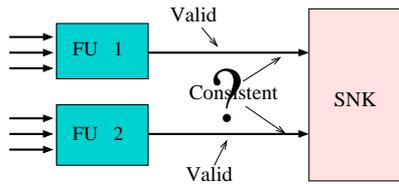


Figure 6. Validity vs. Consistency

conditions:

Consistency Condition 1: Existence of transitions
 There must be at least one signal transition between any two successive code words. While this naturally happens in transition based coding schemes, it requires special efforts to ensure a transition between two successive identical data words in state based coding schemes. A usual solution is to introduce a "neutral" code word (like all zero, e.g.) between any two data words in a "return to zero" manner.

Consistency Condition 2: Membership to contexts
 As can be seen in figure 3, two data waves (belonging to a different context) will transiently coexist between SRC and the associated SNK: There is a finite interval when the new data wave has already been issued and is propagating through the FU, but the previous one is still valid at the SNK's input. This procedure is properly synchronized by the trigger control. Would we admit more data waves between SRC and SNK we would lose control of them and in particular not be able to prevent one data wave from catching up with its predecessor (unless this is ensured by timing assumptions). As a consequence, if our basic requirement is to be able to distinguish data waves with different context, we normally come along with two disjoint code sets on the rail level, which allows us to unambiguously assign every bit to one of two data waves.

Losslessness: As already outlined the losslessness property requires us to provide the data source with information on when new data can be issued and the data sink with information on when data can be consumed. The latter can be achieved by checking consistency and validity of the input vector. The source trigger can only be derived from information explicitly provided by the data sink such as a control signal acknowledging the consumption of the previous data word. Since there is only one single bit of information required on this backward path, there is no potential for skew effects. However, the consumption of a data word can usually not be directly measured, which gives rise to conceptually weak compromises in this respect.

From a higher level of abstraction we can consider the logic function unit as part of the data source/sink

and map the lossless requirement to communication process problem.

Obviously there must be an agreement between source and sink, in which way data is transmitted over the communication channel - a so-called communication protocol. Basically we can distinguish between a 2-phase protocol and a 4-phase protocol. In contrast to the 2-phase protocol, the 4-phase protocol returns back to its "neutral state", after each communication cycle. (see figure 7)

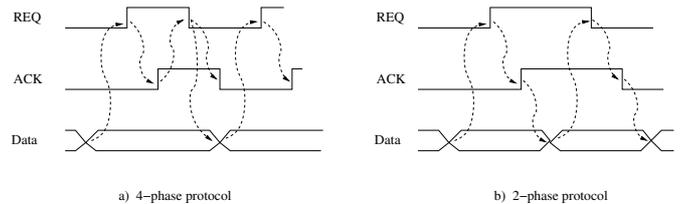


Figure 7. 2-phase vs. 4-phase protocol

Furthermore we have to distinguish between *push channel*, where the data source is the active party, and *pull channel*, where the data source reacts on requests of the data sink. A detailed description of communication mechanism with respect to asynchronous circuits can be found in [9]

4.3. Hybrid solution

It is not necessary to solve the fundamental design problem in one domain only. Quite on the contrary, many design approaches are based on a hybrid solution. Huffman codes [4] or micropipelines [11], e.g., solve only a part of the fundamental design problem and only their combination with other methods yields the desired result.

Furthermore, we have to take into consideration different abstraction levels of a circuit implementation. Until now we have talked only about abstract functional logic blocks, disregarding whether we are considering a simple inverter built from two transistors or a complex ALU. The distinction between abstraction levels is important because several design approaches solve the fundamental design problem not only in different domains but also on different abstraction levels. In most cases library cells, such as AND, OR, latches, ..., are implemented by making timing assumptions e.g. isochronic fork [5] or fast local feedbacks [2], since it is quite easy to consider timing assumption within such atomic cells and yields a more efficient implementation in terms of speed and silicon area. On a higher level it is assumed that the basic gates fulfill specific requirements with respect to consistency and validity. This

allows us to build circuits, such as an ALU, for which the fundamental design problem is entirely solved in the information domain (on this higher level of abstraction).

5. Design Techniques

Obviously this paper cannot cover all existing design techniques. Instead, we will analyze some popular approaches in the light of our system model and show how they can be compared.

5.1. Synchronous Approach

The synchronous approach solves all subproblems concerning the fundamental design problem in the time domain. It employs a unique control rail, the clock signal, to indicate validity, consistency and lossless at the same time. At every active edge of the clock all signals have to be consistent and valid by definition and therefore ready to be consumed. Due to the fact that data sources get the same clock signal as data sinks, the active clock edge signalizes also the point in time where new data can be issued. In this way the regulation of the data flow is also strictly based on time and occurs without feedback (see figure 8). By assuming that all data sources and sinks get a common global time reference from the clock signal 4.1, it is implied that all these components actually get the rising and falling edges at the same time. However, since skew and delay effect also affect the clock signal, this claim is not justified for deep sub-micron technologies. Quite on the contrary, [6] predicts that in the near future only a small percentage of the die will be **reachable** during a single clock cycle.

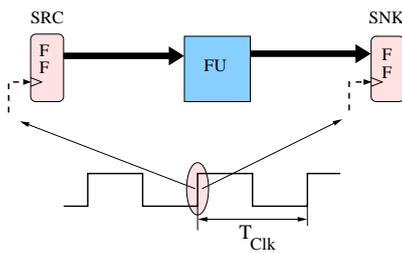


Figure 8. Synchronous design approach

Notice, using the same clock signal leads to an implicit synchronization between data sources and data sinks. This is an extremely elegant method to solve the fundamental design problem. However, there is no immediate relation to consistency/validity of signals or rails and the clock signal, it is just a strictly periodic

and time driven control signal. In essence, an indirect conclusion from time consistency is made as outlined in 4.1.

The minimum distance between active clock edges T_{Clk} is defined by $\Delta 1$. $\Delta 2$ is set to zero due to the fact that at the moment, where the data is consumed, new data is issued. This implies a further assumption, namely that data sources and sinks are ready to perform their operation immediately after a command is received.

5.2. Bundled Data Approach

The basic concept of **bundled-data** [10] is to arrange several (data-)signals in a group and to use a common control signal, which signalizes validity and consistency of this (data-)signals. This supposes that the data path is at least as fast as the control path. Therefore, to maintain the correct behavior matching delays are usually inserted in the control signal path. In this sense bundled-data solves consistency and validity in the time domain. Although the control signal can be used as trigger for data sinks, the bundled data approach does not provide any means for data flow control. This requirement has to be fulfilled by other methods.

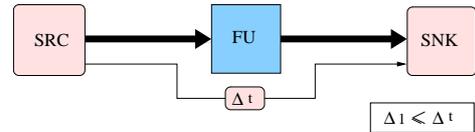


Figure 9. Bundled-data design approach

As illustrated in figure 9 consistency and validity are ensured in a similar manner as in the synchronous approach. The main difference is that the bounded delay approach allows the designer to define an individual delay for each data source/sink pair, while the clock signal is used for the whole circuit.

5.3. Micropipeline Approach

Basically **micropipelines** (as introduced by [11], see figure 10) are a means for structuring complex logic designs in general and datapath designs in particular. In contrast to the synchronous pipelines they employ local handshake signals between any two pipeline stages to interlock the inter-operation between the individual stages such that the speed of data flow can be adapted to the local situation. In this way the micropipeline approach provides a straightforward solution for data flow control.

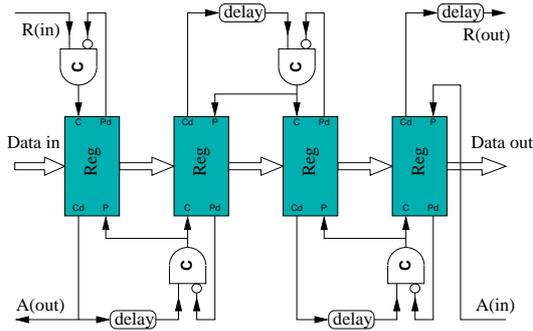


Figure 10. Micropipeline

In the original micropipeline introduced by Sutherland consistency and validity are solved using *delay-elements*. However, it is possible to generate the completion signals by combining the micropipeline with other approaches.

5.4. Huffman Approach

Huffman circuits [9] are intended to be used for asynchronous state machines. As depicted in figure 11 a Huffman circuit has primary inputs, primary outputs and a feedback loop. The current state is “stored” in the feedback path and thus may need delay elements to prevent state changes from occurring too rapidly. The fundamental mode requires that only one input signal changes at a time. Due to the fact that the feedback signals which hold the state information are inputs of the combinatorial logic, it is even required that by passing from one to the next state, only one signal changes. A new input cannot be issued until the whole circuit has reached a stable state. A further requirement of Huffman circuits is, that the combinatorial logic is glitch-free. While validity is solved in the information domain (glitch-free functions) and by the environment (only one bit changes at the input side), consistency is solved by the delay element in the feedback path. The lossless property has to be solved by the environment: It is assumed that a new input vector is issued only when the circuit is already stable.

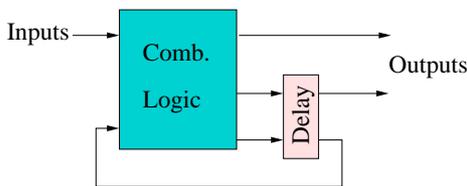


Figure 11. Huffman circuit

There are some enhancements of the Huffman cir-

cuit (nonfundamental mode, burst mode) which allows, that more than one input changes per time [1].

5.5. Graphical Description Approach

There are many possibilities to represent a circuit using a graphical description, Petri Nets, STG, I Nets, e.g. [9]. There are differences between representation, working conditions and requirements, but the basic principle is always the same: In contrast to hardware description languages, graphs yield a pictorial description of the circuit. Petri nets are a well established approach to model concurrent processes and therefore constitute an ideal medium to describe circuits. Basically, a petri net is a graph composed of directed arcs and two types of nodes: transitions and places. Some places can be marked with tokens. The petri model is executed by firing transitions. A transition is enabled to fire if there are all tokens on its input places. This requirement guarantees consistency and provides a synchronization. Further a token is considered to be always valid. The circuit is re-triggered by providing some feedbacks, which ensure that the tokens circles within the graph. The main advantage of such a graphical description is that there exists a well established theoretical background, which allows to analyze the circuit and to prove the correct behavior. However the graphical representation is a high level description of a circuit, the designer/design tool has to ensure that the real implementation of the nodes fulfills validity, consistency on her part.

5.6. NCL Null Convention Logic Approach

Null Convention Logic extends the Boolean logic by a so-called NULL state [2]. In particular an NCL signal can assume a DATA state – which is either a valid HI or a valid LO, in NCL called “TRUE” or “FALSE”, respectively – or a NULL state. For encoding these three states the single-rail approach is obviously not sufficient, and a two-rail signal representation is used instead, with NULL being represented as (0,0), TRUE as (1,0) and FALSE as (0,1). The NULL state does not convey any information, it serves only as neutral state separating two consecutive codewords. Figure 12 illustrates this behavior.

Feedback ensures that a new data (DATA or NULL) can be processed only when the input vector is consistent. To realize this behavior so-called threshold gates are used. These gates change their output only when the complete input vector is either DATA or NULL. This hysteresis provides a synchronization of the wavefronts on the gate level. In other words consistency

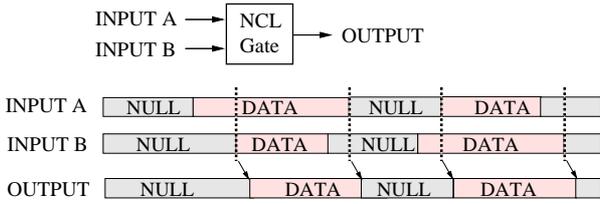


Figure 12. Sequence of DATA and NULL waves

and validity check on signals are implemented at gate level. With the proposed encoding on signal level exactly one rail changes its logic level upon the transition from NULL to DATA and vice versa, regardless of whether DATA is TRUE or FALSE. Due to the mandatory introduction of the NULL waves a neutral state (NULL) is assumed on every signal after every single data word, which enforces the edge required to meet the existence condition. From this neutral state an edge on any one of the two rails leads to the TRUE or FALSE state, which guarantees that the codeword itself is always valid. The NCL approach does not provide any mechanism to ensure lossless.

5.7. Four State Logic Approach

Similar to NCL the four state logic approach [8][7] uses dual rail encoding to represent a signal. In contrast to NCL no neutral state is used. Instead two states are used to represent "TRUE" (P1 and Q1) and two to represent "FALSE" (P0 and Q0). In this way a phase information is added to the pure boolean information - consecutive data wave are coded alternatively in P and Q phase. This makes consistency visible. The coding scheme ensures that only one transition occurs when passing from P to Q phase and vice versa, independently of the information conveyed. Furthermore it is required that function units keep their old output value if the input vector is not consistent. By doing so, the signal are always valid. (see 13) The four phase logic approach does not provide any mechanism to ensure losslessness.

5.8. Transition Signalling Approach

In conventional coding techniques logic states of signals are mapped to levels of physical rails. In contrast to this transition signaling [10] uses *edges* on rails to convey the information. Transition signaling also employs two-rail coding on the signal level. A transition on one rail indicates a HI, a transition on the other rail

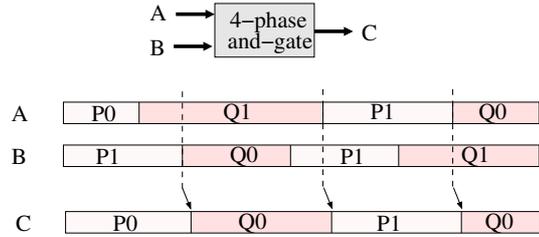


Figure 13. Sequence of data waves in four phase logic

a LO. From a more abstract point of view transition signaling uses a one-hot encoding scheme for HI and LO and therefore fulfils the validity property on code level. The neutral state between consecutive codewords is defined by the absence of transitions on the rails. In contrast to NCL, where the neutral state must be explicitly generated, transition signaling provides this state automatically and hence a new codeword is recognized even if it carries the same information as the previous one. In this sense consistency is integrated directly in the coding style. In [5] it has been shown that the only single output gates that can be used in conjunction with transition signalling circuits are Muller-C-Gate and inverter. This limits the usability of this scheme for real circuits.

5.9. Comparison

In table 1 the presented methods are compared with respect to the domain, where they solve consistency, validity and losslessness. The column *E* (Environment) is used to express that the design technique does not solve the corresponding subproblem, but moves the responsibility to the environment. Column *I* (Information) and *T* (Time) are used to express whether the problem is solved in the information or in the time domain.

	Validity			Consistency			Lossless		
	T	I	E	T	I	E	T	I	E
Sync	x	-	-	x	-	-	x	-	-
Bundled	x	-	-	x	-	-	-	-	x
Microp.	x	-	-	x	-	-	-	x	-
Huffman	-	x	x	x	-	-	-	-	x
Graph. Desc.	-	x	-	-	x	-	-	x	-
NCL	-	x	-	-	x	-	-	-	x
4 Phase Logic	-	x	-	-	x	-	-	-	x
Trans. Sig.	-	x	-	-	x	-	-	-	x

Table 1. Comparison of design methods

6. Summary

The multitude of different theories and approaches concerning asynchronous logic tends to hide the basic problem that all methods try to solve. However, knowledge and understanding of this fundamental problem is of crucial importance for hardware designers. In this paper we figure out this fundamental design problem and its originates. Two basic strategic options, namely solving the problems in the time or in the information domain, are provided and characteristic design methods with respect to the strategy they use are analyzed. This examination is important to get a better understanding of current design techniques, to be able to classify them and to use them according to their particular strengths.

References

- [1] A. Davis and S. M. Nowick. An introduction to asynchronous circuit design. Technical Report UUCS-97-013, University of Utah, Department of Computer Science, 1997.
- [2] K. M. Fant and S. A. Brandt. Null convention logic: A complete and consistent logic for asynchronous digital circuit synthesis. In *Proc. International Conference on Application Specific Systems, Architectures and Processors*, pages 261–273, august 1996.
- [3] S. Hauck. Asynchronous design methodologies: An Overview. In *Proceedings of the IEEE*, volume 83, pages 69 – 93, 1995.
- [4] D. A. Huffman. The synthesis of sequential switching circuits. *Journal of the Franklin Institute*, March/April 1954.
- [5] A. J. Martin. The limitations to delay-insensitivity in asynchronous circuits. In *Proceedings of the sixth MIT conference on Advanced research in VLSI*, pages 263–278. MIT Press, 1990.
- [6] D. Matzke. Will physical scalability sabotage performance gains. *IEEE Computer* Vol 30, num. 9, pp.37-39, September 1997.
- [7] A. McAuley. Four state asynchronous architectures. *IEEE Transactions on Computers*, Volume 41(Issue 2):129 – 142, 1992.
- [8] T. W. M.E. Dean and D. Dill. Efficient self-timing with level-encoded 2-phase dual-rail (ledr). In *Proceedings of the 1991 University of California/Santa Cruz conference on Advanced research in VLSI*, pages 55–70. MIT Press Cambridge, MA, USA, 1991.
- [9] C. J. Myers. *Asynchronous Circuit Design*. John Wiley & Sons, inc., 2001.
- [10] J. Sparso and S. Furber. *Principles of Asynchronous Circuit Design*. Dimes, 2001.
- [11] I. E. Sutherland. Micropipelines. In *Communications of the ACM*, volume 32, pages 720 – 738, 1989.
- [12] T. Verhoeff. Delay-insensitive codes—an overview. *Distributed Computing*, 3(1):1–8, 1988.