Design, Analysis and Implementation of a Self-timed Duplex Communication System

Alex Yakovlev[a], Steve Furber[b] and Rene Krenz[c]
(a) University of Newcastle, U.K.,
(b) University of Manchester, U.K.,
(c) Royal Institute of Technology, Stockholm, Sweden

Abstract

The design of an asynchronous communication system using partially automated techniques is described in this paper. The protocol is formally specified as a protocol state machine and verified with respect to deadlock-freedom and delay-insensitivity using Petri net based model-checking tools. A protocol controller has been synthesized by direct mapping of the Petri net model derived from the protocol specification. The logic implementation was analysed using the Cadence toolkit. While most of the controller's logic is robust to arbitrary gate delay variations, a number of speed-up strategies based on relative timing have been considered. The results of SPICE simulation show the advantages of the direct mapping method compared to logic synthesis. Overall, the design process suggested here offers a generic way to constructing asynchronous communication systems, for both on-chip and off-chip interconnects.

1 Introduction

Asynchronous or self-timed circuits offer a number of advantages for system design. The most attractive properties of these circuits are low power consumption, electromagnetic compatibility, greater modularity and operational robustness. One particular area of digital IC design where circuits with global asynchrony are seen more as an inevitable technological reality rather than an optional design discipline is interfacing. Development of formally sound methodologies and tools to support design of communication protocols and interfaces for systems-on-chip and multi-chip systems is a difficult problem. The best way to acquire experience in solving such a problem in its generic form would be to tackle a specific interface design example maximally using formal techniques. This way is also motivated by the fact that most of future SOC design will be communication-driven [1].

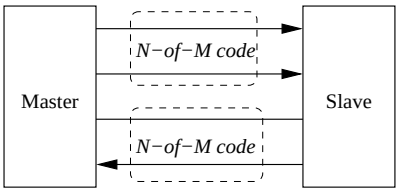

Figure 1. Overall communication system

An inter-chip communication system that uses delay-insensitive (DI) encoding and in this way optimises both power- and pin-efficiency has been proposed in [2]. The overall view of the system is shown in Figure 1. The construction of the communication controller for this system presents an interesting case study for asynchronous design techniques. This process is described in this paper. First, we construct a formal description of the protocol using the concept of a protocol state machine. Second, this protocol is verified, with respect to its requirement to provide delay-insensitive communication, using a Petri net analysis tool based on unfoldings. Third, we derive a Petri net specification for control logic in both entities of the protocol, master and slave. Forth, we translate the Petri net model of each controller to a circuit implementation using the so-called direct translation into David cells. Finally, we compose the main controllers with additional transmit and receive interfaces and study the performance of the overall system by SPICE simulation using the Cadence toolkit.

The use of direct translation from a Petri net specification is a distinct feature of our design, which produces results that compare favourably against circuits obtained by logic synthesis from Signal Tranistion Graphs (STGs) and the Petrify tool [3]. This design example can be seen as an important benchmark for a future automatic control synthesis tools employing direct translation techniques [4]. The overall methodology based on formal specification of a protocol, its refinement as a controller specification and finally, direct mapping to logic, guarantees correctness of the de-

sign described in this paper.

2 Asynchronous Communication System and its Protocol

This part briefly describes the asynchronous communication system proposed in [2] and presents a formal capture of the protocol.

2.1 Proposed bidirectional communication scheme

The classic non-return-to-zero (NRZ) dual-rail approach with acknowledgement requires at least six wires in order to provide a bidirectional communication channel. Each direction uses two wires to transmit one bit of data (by making a transition on wire 0 when transmitting a logical zero and a transition on wire 1 when transmitting a logical one) and one wire to transmit an acknowledgement. Using other types of delay-insensitive codes, such as for example N-of-M codes, is also possible; this may increase the pin-efficiency compared to the dual-rail code but at the cost of having some additional logic to convert the N-of-M code to normal one and vice versa.

The scheme proposed in [2] optimised the classic dual-rail mechanism by using only four wires, two in each direction, and exploiting the data wires in one direction to carry acknowledges for communication in the other direction. Namely:

- Four wires are used for bidirectional communication. A0 and A1 carry data symbols in one direction; B0 and B1 carry data symbols in the other direction.
- During true bidirectional communication a transition on A0 or A1 is acknowldeged by a transition on B0 or B1, and vice versa.
- During unidirectional communication the same protocol applies, but the retured data is void.
- The start of valid data is indicated by preceding it with a 'Start' symbol. For example, a void response could be zero and a Start symbol a one. A predefined number of bits following a Start symbol represent valid data.

For a low-power communication system it is desirable for us to minimise the number of transitions on wires used to send a given data value. In particular, we want to avoid sending transitions when there is no data to send. The above bidirectional discipline has therefore been enhanced with a mechanism in which either end of the channel may initiate communication at any time, including the possibility of both ends to initiate it at the same time (to within some tolerance). The latter condition is called collision. Initiation of communication requires the generation of a token and to prevent the generation of two tokens in the system, the following is ensured in the protocol:

- Both ends of the channel 'know' there is a collision in the system as both will issue a Start symbol and receive a Start symbol instead of an Ack symbol.
- One end of the channel must 'defer' to the other. The end which defers is called the Slave, and the other the Master.
- The Slave defers by retracting its Start symbol and replacing it by an Ack.

In a true DI system true retraction is not possible since, once the sender has made a transition on a wire, it cannot make another transition on the same wire until it has had confirmation (in the form of some sort of acknowledgement) that the first transition has been received at the other end. Instead of such a single wire retraction, which is truly delay-dependent, a special symbol, called SlaveAck, is used. SlaveAck subsumes the Start symbol. In the case of dual-rail encoding a SlaveAck symbol is superposition (i.e. union) of Start and Ack symbols. Thus, the wire that makes a transition in the Start symbol also does so in the SlaveAck symbol, and the additional wire which does not make a transition in the Start symbol makes a transition within the SlaveAck symbol to indicate a retraction of the Start symbol. That second wire is also used, on its own, to represent an Ack symbol.

To sum up, in a dual-rail DI system, 01 can be used for Start, 10 for Ack and 11 for SlaveAck.

2.2 The Protocol

The full protocol for the above communication scheme is represented by a state-transition graph in Figure 2. This graph specifies an imaginary (protocol) state machine, placed between the Master (M) and Slave (S), which defines permissible signal sequences that can be seen on the wires connecting the Master and Slave. Serving the purposes of the protocol definition only, this machine does not show any behaviour that is internal to the Master and Slave. The six major states of the protocol are Idle, Slave transmit (Ts), Master transmit (Tm), Retract (Ret) and two duplex states (TsTm and TmTs). These states together with minor (transient) states, shown by small circles, are assigned to four main modes that are Initialisation (I), two simplex modes Master transmit (II) and Slave transmit (III) and one duplex mode (IV).

For every state in the protocol certain transmission events are allowed to take place. For example, when the

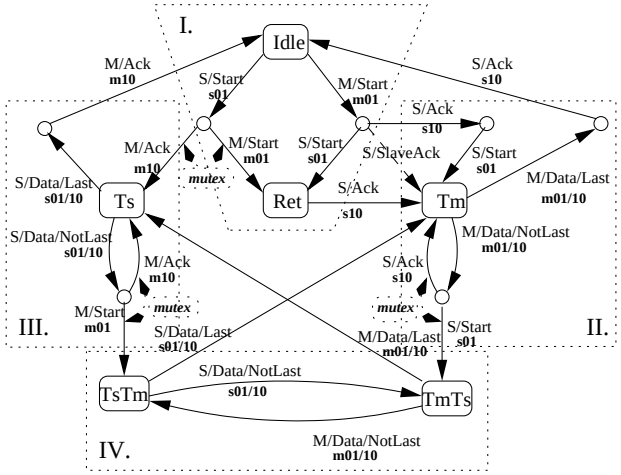

Figure 2. Protocol State Diagram

protocol is in state Idle, there may either be received a Start symbol from the Slave (S/Start) or a Start symbol from the Master (M/Start). Likewise, when in the state Ts, either 'S/Data/Last' or 'S/Data/NotLast' symbol may be received from the Slave. The notation 'S/Data/Last' indicates that the Slave sends its last data value.

The labelling of these arcs, e.g. m01 or s10, is associated with the source of the signal ('m' for Master and 's' for Slave) and the dual-rail encoding of the symbols (Start, Ack and bit-data values), and its interpretation, whether it is a special symbol or a bit value, depends on the current mode and the state. Thus, s10 (s01) indicates that a transition is made on wire 0 (1) that comes from the Slave. Similarly, m10 (m01) stands for a transition on wire 0 (1) from the Master. When occurring (coming out or leading into) the initialisation mode these labels correspond to symbols; in simplex transmission modes they are used for a data value from one side and a symbol from the other side; finally, in the duplex mode, these labels always correspond to data values only.

The protocol implicitly has two types of choice in its states. One such type, non-arbitrating choice, is made in states Ts, Tm, TsTm and TmTs. It depends on the mutually exclusive conditions that are assumed to be satisfied within Master and Slave statically, whether the transmitted value is either the last data value or not last. We will also assume, for simplicity of logic in the controllers, that both the receiver and transmitter have the inner (higher level) facilities for counting the number of data values transmitted and received in such a way that they always agree on their decisions Last/notLast between them.

Three states, which happen to be transient, involve arbitration or dynamic 'mutexing' (note the 'mutex' labels), where the decision which state must be next is made non-deterministically within the Master or the Slave. For example, let the S/Start signal arrive first in the initialisation mode but the Master has just been requested by its client to start data transmission. The decision between sending M/Ack and M/Start is made through an arbitration process in the Master. Similarly, when the protocol is in the Master transmit mode (II) and the Slave receives a data value that is not last just at the time when its client issues a request for data transmission, the Slave must arbitrate to decide whether to send S/Ack or S/Start. The issue of the implementation of arbitration will be discussed later.

We have put this distinction between non-arbitrating and arbitrating choice deliberately here, in order to create intuition for analysis of the protocol although, in principle, the protcol state machine carefully hides this distinction away because any choice is assumed to be made outside this imaginary machine.

Note that the transmission of symbol SlaveAck, defined in the previous section, is illustrated in the form of a confluent (diamond) structure of state transitions representing the transmission of both S/Start and S/Ack, involving both wires s10 and s01. Such a diamond structure shows an interesting property of this protocol, namely that it theoretically does not need another arbitration in the Slave, which would have been symmetric to the one in the Master when exiting the initialisation mode. This is because whether the Slave sends an acknowldegement to M/Start or retracts it always produces both symbols S/Start and S/Ack. The latter, transmitted by the Slave either in sequence or in parallel, due to the delay-insensitivity of the channel, are assumed to arrive in the protocol machine (and the Master) in either order.

3 Protocol Verification

The correctness of the protocol defined in the previous section is seen in terms of the following two main properties that it must satisfy: absence of deadlocks and delay-insensitivity. Both these properties can be verified by constructing a formal model of the communication system using our protocol definition. The model of the system consists of the models of the Master and Slave and the dual-rail communication channel. In order to adequately capture the concurrent behaviour of the Master and Slave, which may perform some of their actions independently, we use the language of Petri nets to represent these models. This idea is illustrated schematically in Figure 3. We constructed two Petri net parts for the Master and Slave following the basic protocol in Figure 2 and inserted two pairs of places, (m01, m10) and (s01,s10), between those parts to represent the two-wire channels. Having a place for each wire in the channel allows modelling a delay between the source of a particular signal wire event and its destination. The former

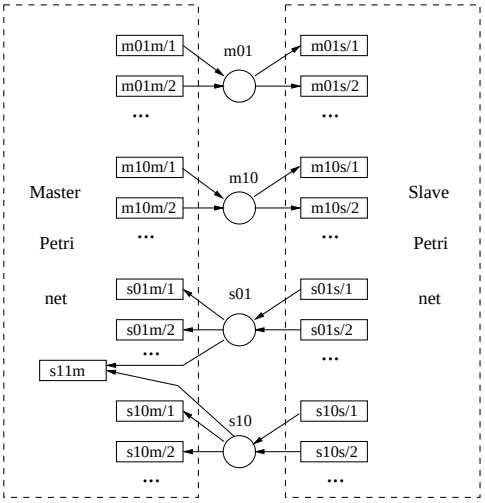

Figure 3. Petri net modelling of the communication system

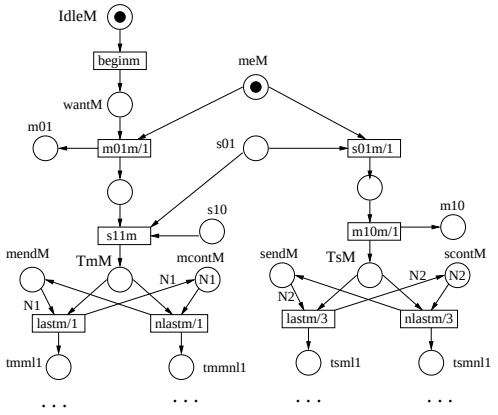

Figure 4. Fragment of Master subnet

is modelled by a transition that acts as a producer of tokens, e.g. ‘m01m’ indicates sending a signal event on wire m01 by the Master. The latter is represented by a consumer transition, e.g. ‘m01s’ stands for receiving a signal event on wire m01 by the Slave. The fact that there can be several instances of the same event, activated in different states of the protocol is shown by the index of the transition label, cf. ‘m01m/1’, ‘m01m/2’ etc. Additionally, we may use the combined consumer transitions, e.g. ‘s11m’ to indicate the fact that the Master must receive both an event on ‘s01’ and ‘s10’ (SlaveAck symbol).

The Petri net models of Master and Slave parts are built by tracing the protocol in Figure 2 except that we should adequately model concurrency inside the Master and Slave due to interaction with their respective clients. A fragment of the Master model is shown in Figure 4. It illustrates the arrival of a request to transmit from the client by the transition labelled ‘beginm’, which takes the Master from state ‘IdleM’ to ‘wantM’. This transition may fire independently of the arrival of a token in place ’s01’, which models the arrival of a S/Start signal from the slave. The fact that there is an arbitration in the Master which must decide whether to send M/Start (i.e. fire transition ‘m01m/1) or send M/Ack (i.e. fire transition ‘m10m/1’) is represented by the mutual exclusion construct in the Petri net with a single token in place ‘meM’. Note the places representing the wires with corresponding producing and consuming transitions, shown previously in Figure 3.

To model synchronism in counting the length of messages by both Master and Slave, we use net fragments consisting of pairs of places, (‘mendM’, ‘mcontM’) for the counter which counts bit values transmitted by the Master, and (‘sendM’,‘scontM’) for counting bit values received by the Master. The lengths of transmitted and received messages is ‘programmed’ by the values N1 and N2, respectively, which makes the lengths of such messages N1+1 and N2+1. N1 and N2 also indicate the number of tokens that are initially placed into places ‘mcontM’ and ‘scontM’, as well as the weights of the corresponding arcs. For example, in the case of N1, this arrangement of tokens and weights allows transitions ‘nlastm’ (standing for the NotLast data bit value case), which decrement the counter initially set to N1, to fire N1 times before a transition ‘lastm’ (standing for the Last data bit value case) may fire, which also resets the counter back to N1.

The full Master subnet (produced by the graph drawing tool from a symbolic description) is illustrated in Figure 5. In this net we used a simple message length count model, in which N1=N2=1, i.e. each message consists only of two bit values. This way we could exercise both Last and NotLast branches of each choice concerned with data transmission and reception. The reason for that was to ensure 1-safeness of the initial Petri net model to avoid problems with unfolding k-safe nets (where complexity grows exponentially). This restriction could in principle be removed by using a 1-safe net model of a modulo-k counter to some k>2, the idea for which is shown in Figures 6 and 7. Here, the first figure shows a 3-safe Petri net model for a modulo-4 counter while the second figure illustrates a refinement using a 1-safe net. Note also that these models show how the same counter can be accessed for performing Last and NotLast actions from different access points (this is crucial because in our model, either Master or Slave, we have two such access points for each of the two counters, one in simplex and the other in duplex transmission modes).

We analysed the overall Petri net model of the communi-

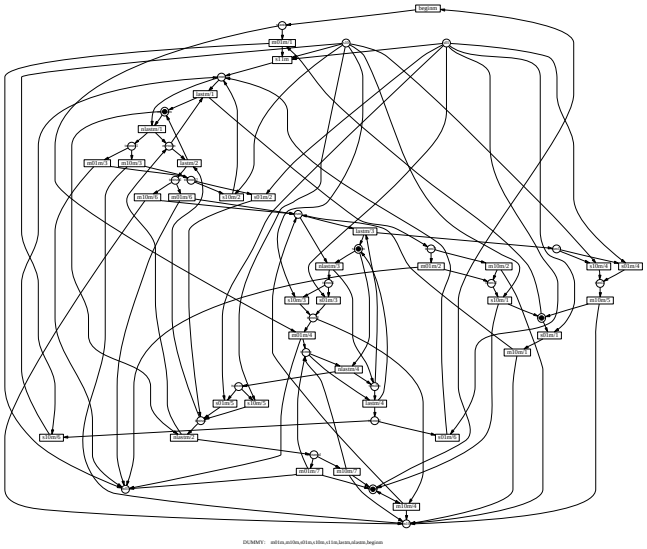

Figure 5. Master part verification

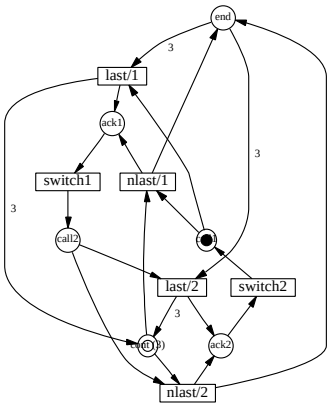

Figure 6. Petri net for modulo-4 counter

cation system using the partial oder technique based on the construction of a finite prefix of the unfolding implemented in the PUNT tool [5]. The tool proved that the net was free from deadlocks. We verified the delay-insensitivity in the following way. If the system was not delay-insesnitive with respect to delays in the wires, any violation would have manifested itself in a so-called communication interference [6]. The latter is a transmission event on a channel wire that is not acknowledged by a receiving side, and as a result another event may occur on the same wire. This condition is easily detected in the Petri net model by means of a check for 1-safeness with respect to the places corresponding to wires m01, m10, s01 and s10. Indeed, if any such place was not 1-safe that would have been equivalent to the occurrence of two producing actions on such a place (wire) without at least one consuming action. In its turn check-

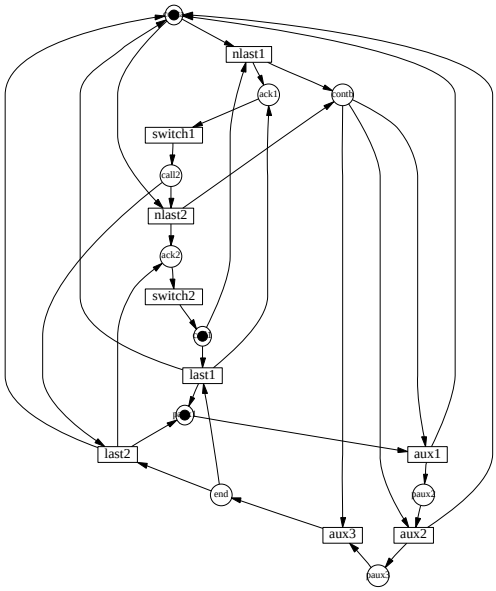

Figure 7. 1-safe refinement for modulo-4 counter

ing whether a given place is not 1-safe is a trivial test in the unfolding prefix; it amounts to finding a pair of mutually concurrent instances of this place. The PUNT tool has proved that the net is 1-safe and thus the protocol is delay-insensitive. We illustrate the convenience of this tool by presenting the finite prefix of the communication system's net in Figure 8, which contains 84 transition instances. The verification results were corraborated by reachability analysis using Petrify, which generated a state graph with 166 states for this net. Showing such a state graph, produced by a graph drawing tool, would not make much sense due to its topological complexity whereas the unfolding prefix is a relatively compact capture of the semantics of the protcol. Such a prefix can usually be traced without difficulty.

4 The Controller System

The controller system is divided into three parts, as shown in Figure 9:

- main controller
- send interface
- receive interface

The main controller manages the interaction with the send and receive interfaces according to the protocol described above. It activates and acknowledges the transmission and

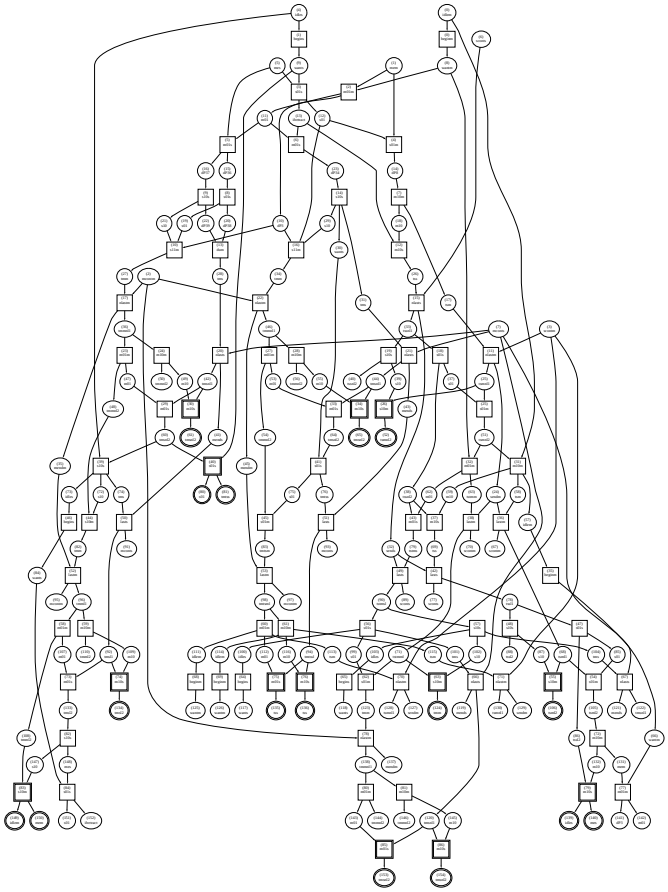

Figure 8. Unfolding prefix built by PUNT for verification

reception of data and control symbols, and controls forwarding data from the source system, Sender, to the output channel, and from the input channel to the destination system, Receiver. The controller has no direct connection to the data path; this is done entirely via the send and receive interfaces.

The send interface recognises a request to send data from the source system and informs of that the main controller, which starts the initialisation procedure. The send interface carries out the appropriate commands of the main controller concerned with issuing data and control symbols. In doing so, it performs phase conversion of data bits from the Sender so they appear in two-phase NRZ dual rail form in the output channel; it also indicates to the Sender its readiness to transmit the next bit of data. A counter recording the number of data bits transmitted is also incorporated in the send interface (however, alternative ways could be explored, such as the provision of an explicit signal from Sender to indicate the last data bit).

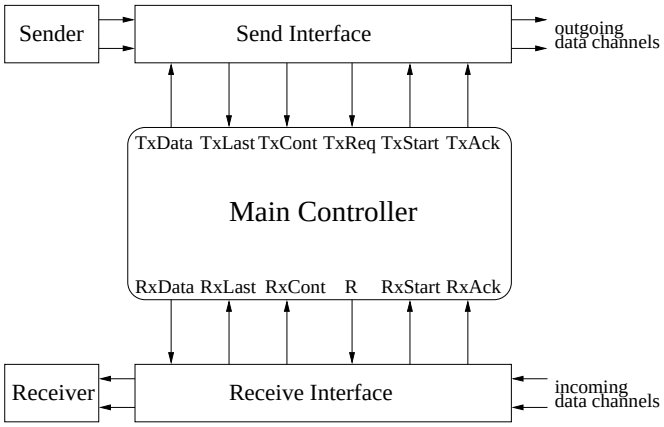

Figure 9. Overall structure of controller system

The receive interface recognises the arrival of data from the input channel, carries out appropriate commands of the main controller concerned with receiving data and control symbols. In doing so, it performs phase conversion of data bits from two-phase to four-phase RZ form for the Receiver. It also maintains a counter to register the number of bits received.

4.1 The Main Controller

As mentioned above, the main controller follows the protocol. It does not deal with the actual encoding of data and symbols in the input or output data channels - these are the functions of the send and receive interfaces. Such a distribution of functions allows the re-use of the main controller logic in designs where different delay-insensitive coding is used (cf. M-of-N codes [2]). There are two versions of the controller, one for Master and the other for Slave. The differences lie in the initialisation part of the protocol. The LPN of the Master controller which is the base of the logic implementation process is shown in Figure 10. Some transitions are labelled as 'dummy'; these events are internal for the controller - they do not activate any actions in the interfaces. They are inserted in order to satisfy the requirement of direct mapping of LPNs into circuits, where every cycle in the LPN must have at least three transitions to avoid deadlocks in the circuit [4].

The non-dummy transitions are labelled as follows. For instance, the $TxStart$ label activates a two-wire push handshake on the send interface side — it means the command to send the M/Start symbol. The $R \rightarrow RxStart$ label corresponds to the activation of a two-wire pull handshake on the receive interface side meaning the command to pull the expected S/Start symbol. The $R \rightarrow RxStartANDRxAck$

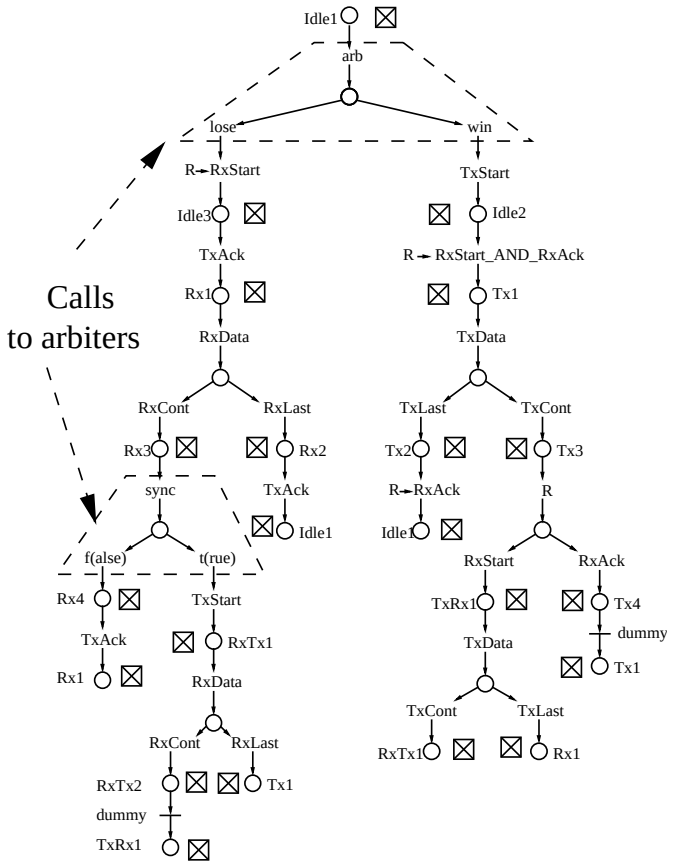

Figure 10. LPN of master controller

refers to a three-wire pull handshake which pulls the expected SlaveAck symbol. A number of handshakes involve three transitions, such as for example $RxData$, $RxCont$ and $RxLast$, which corresponds to pulling the value of the receive data counter. This is modelled by nondeterministic choice, with the decision made outside the controller. Finally there are two groups of three transitions corresponding to the three-wire handshakes with arbitration blocks, which are part of the Main Controller logic but they are implemented outside the logic mapped from the LPN in Figure 10.

4.2 The Send Interface

The functionality of the send and receive interfaces must correspond to the specification of the main controller, and to the links with the Sender and with the output channel, for the send interface, and with Receiver and input channel, for the receive interface. The send interface consists of a counter, a 4-2 phase converter and a Tx-adapter.

The counter controls the input signals RxCont and RxLast of the main controller. It is triggered by the TxData signal of the main controller. If the currently transmitted data bit is not the last one, the counter responds with TxCont, otherwise with TxLast. The counter of the send interface of the source-system and the counter of the receive interface of the destination system have to be initialised with the appropriate number of data bits in the message. The counter works independently of the other components of the send interface. The structure of the counter will be discussed in the implementation section.

The 4-2 phase converter turns a RZ-signal into a NRZ-signal. That means that every complete pulse on the input is turned into an edge on the output. The converter produces an acknowledgement to the Tx-adapter for every data bit.

The Tx-adapter is the central component of the send interface. Here, the data signals from the sender are received and acknowledged and the TxReq signal is produced. It also synchronises control signals from the main controller and data signals from the source system to generate signals to the data channels. It provides an additional handshake between the 4-2 phase converter and the control signals of the main controller.

4.3 The Receive Interface

As in the send interface the specification of the receive interface is derived from the definition of the interfaces between data channels, the main controller and the destination system. It consists of a counter, a Rx-adapter and a 2-4 phase converter.

During the initialisation state any signal which is received and converted into a four-phase signal is forwarded either to the RxStart input or to the RxAck input of the main controller. This process is also valid for the unidirectional transmission mode. In the case of receiving data either in the bidirectional or unidirectional reception mode the received data bits are forwarded to the destination system and the counter gets incremented. The reception of data is acknowledged by the Rx-adapter.

The counter of the receive interface is triggered by the Rx-adapter. It responds to the main controller with RxLast signal and the RxCont signal, depending on whether the currently received data bit is the last one or not.

The 2-4 phase converter generates a full pulse on the same rail on the output for every edge received on the corresponding input rail. This means that the incoming NRZ-signal will be turned into an RZ-signal. The pulse generation is synchronised with the handshake with the Rx-adapter. The details of the structure of the 2-4 phase converter will be discussed more thoroughly in section 5.3.

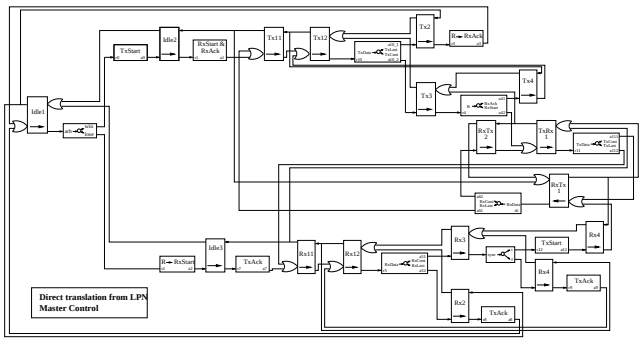

Figure 11. Block Diagram of Master Control

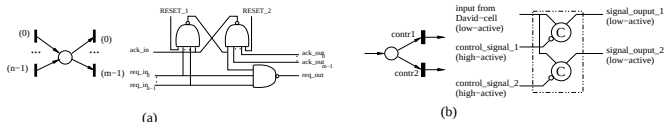

Figure 12. Basic David Cells (a) and branching structure logic (b)

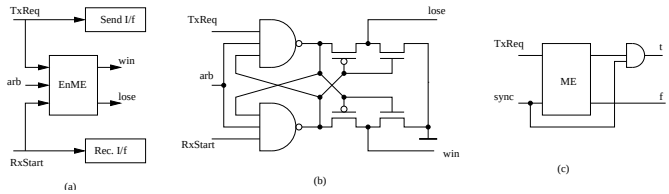

Figure 13. Arbitration components: initial action arbiter (a,b) and simplex-to-duplex arbiter (c)

5 Implementation of the Communication System

This section describes the actual implementation of the communication controller and the send and receive interfaces. Two versions of the controller implementation were considered and compared. One was based on the direct mapping of the LPN into logic (it is called *place-to-latch* implementation). The other was based on the logic synthesis from the Signal Transition Graph refinement of the LPN model, using Petrify [3] (it is called *minimisation* version). The system designs were entered, at the gate level, into the Cadence toolkit using the AMS $0.6\mu m$ technology.

5.1 Implementation using the Place-to-Latch Version of the Main Controller

The approach to circuit implementation according to the place-to-latch approach is described in [4]. The current status of the automation of this method can be found in [7]. The block diagram of the master controller is depicted in Figure 11. It should be possible to recognise in this diagram the overall structure of the original LPN in Figure 10, which confirms that the direct mapping method preserves the topology of the behavioural model in its circuit implementation. According to this method, most places (those shown with crossed boxes in Figure 10) are turned into David cells and every transition is implemented as a logic block, which works according to the appropriate logical operation. In the block diagram David Cells have a control flow direction pointer in their symbol.

The David cells in this implementation are equipped with a reset-mechanism, which affects the placing of the token into these cells or into a token-free state during the reset phase. The structure of the cells used is depicted in Figure 12(a), which also shows how to implement David cells with n predecessors and m successors. Note that the use of NAND gates in David cells implies that the interface with David cells is low-active, where the normal stable state is a logical 1. The three-wire handshakes with externally made choice are implemented in our David cell framework as shown in Figure 12(b), using C-elements. If, for example, an RxData request is generated by the controller, then the reception of an appropriate signal, either RxCont or RxLast, will cause firing of the relevant C-element and the token will be passed to corresponding successor David cell.

The arbitration elements activated by the control logic obtained in the above way are designed as shown in Figure 13. There are two points in the LPN specification where calls to arbiters are required. One is the initial action arbitration, where the controller decides whether to enter the Tx or Rx mode from the Idle state. Its design is shown in Figure 13 (a) and (b). Here, the so called Mutex with Enabling (third input arb) is used. The other arbiter, actived by signal $sync$ implements the decision that controller should make every time it receives new data, whether it can remain in the simplex (Rx) mode or should switch to the duplex (Tx) mode, because a transmission (TxReq) request has been generated.

5.2 Implementation using the Minimisation Version of the Main Controller

The design of the second version of the main controller was based on logic minimisation. The LPN was refined to an STG which was then used by Petrify for deriving logic equations for the circuit. This method of synthesis is described in detail in [3]. The communication of the different sub-circuits inside the main controller is high-active. The implementation of the logical equation of every non-

input signal is included in an individual sub-circuit. In complex STG specifications Complete-State-Coding (CSC) conflicts may occur, due to identically encoded states in different positions of the STG. Petrify detects CSC conflicts and attempts resolving them by inserting extra state signals. Unfortunately, in models like ours, with most handshake signals having multiple occurrences in the STG, the automatic solution of CSC conflicts, which were numerous in the model, was very poor. We prefered using an interactive way of resolving CSC conflicts, where the designer made decisions where to insert new signals in the STG. The position of CSC signals was very important in producing an efficient circuit. A strategy for their insertion was as follows. Firstly, the areas in the STG which formed subsets of states without CSC-conflicts were determined. CSC conflicts were between those areas, called sub-modes of the STG. The inserted signals thus had to act as indicators of the currently active sub-mode. That meant that if the control left the current sub-mode, the state of the indicator had to be changed. Thus the actual position of the control in the system was clearly defined at all times by the indicator. The insertion process was performed on the STG-level, interactively with Petrify which acted as a CSC checker.

5.3 Implementation of the Send and Receive Interfaces

In this section the structure of all components of the send and receive interfaces will be discussed. The main tasks of both interface blocks in relation to the other components of the system were described in Section 4.

5.3.1 Send Interface Implementation

The implementation of the send interface is depicted in Figure 14. The central component of this interface is the Tx-adapter shown in Figure 15. As mentioned earlier, one task of this component is to control the input signals for the main controller. Every input signal, which is specified as a request for a control signal to the main controller, will be set if the appropriate signal is received. The deactivation of this signal after the completed operation is caused by an acknowledgment signal. This adapter provides solely for the forwarding of signals to the 4-2-phase-converter. That means, that the real source of the acknowledgment signal can only be the 4-2-phase-converter. The acknowledgment signal is distributed to all control blocks for the output signals, but it affects only the reset of the currently-set signal. A Set/Reset-block consists of a C-element with one high-active input (set-input) and a low-active input (unset-input). The circuit is depicted in Figure 16.

The TxReq signal, which triggers the transmission process in the main controller, is activated by receiving data on

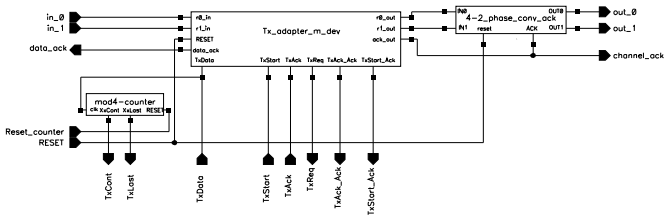

Figure 14. Send Interface Logic

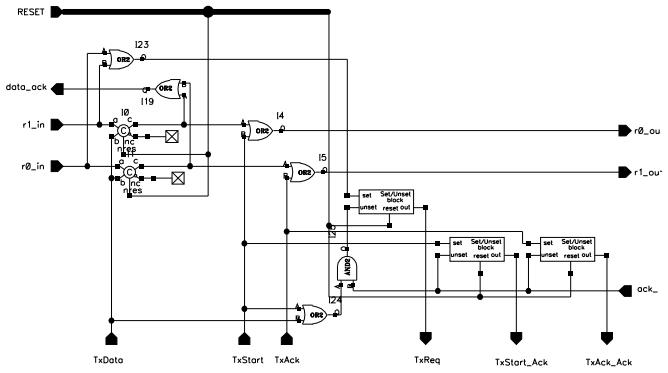

Figure 15. Tx-Adapter

one of the data inputs, $r0\_in$ and $r1\_in$. Because of the possibility of the arrival of a request for sending data from the source (Sender) client at the same time as the Tx-adapter performs a transmission of control-data, the set TxReq signal could be deactivated. That can happen, due to the time delay between recognizing the activation of the TxReq signal and the reaction of the main controller. To avoid conflict, the deactivation of the TxReq is enabled by the TxData signal or by the TxStart signal. Both signals acknowledge the reception of the TxReq signal in the controller. TxStart confirms the receiving of the first data item in the data block. The TxData signal indicates the sending of the other items of the data block. The TxData signal latches the data from the data input channels. Immediately after that, the Tx-adapter sends an acknowledgement signal to the Sender client. The data is removed from the channel after that. No extra handshake functionality is implemented between the counter and the Tx-adapter, which helps those parts of logic to work safely in parallel. This decision can be justified by sufficient timing assumptions. The request of the other controller will arrive much later than the signal from the counter. Thus the controller will be ready to receive the next data bit from the Sender.

The described method is also used for the control of the handshake signals for the TxAck signal and the TxStart signal. The appropriate acknowledgement signal, TxAck_Ack and TxStart_Ack, will be activated after the reception of the control signal and will be deactivated after the reception of

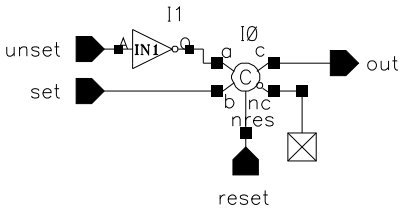

Figure 16. Set-Reset block

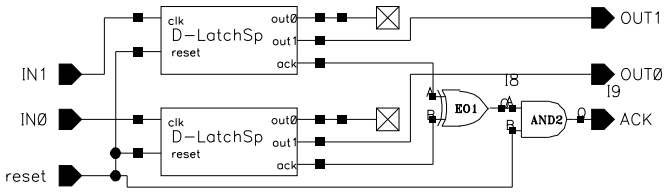

Figure 17. 4to2 phase converter

the acknowledgment signal of the 4-2-phase-converter.

The counter is implemented using ordinary David cells which are arranged as a circle. The number of used David cells is equal to the modulus of the counter. The position of the currently activated David cell inside the circle indicates the actual state of the counter. Between every pair of David cells a C-element is implemented which stops the token flow until the next trigger pulse arrives. The appropriate output signal (TxCont or TxLast) will be activated. This confirmation causes a reset of the trigger signal (TxData). The low-active signals of a set of David cells are connected using a NAND-gate. This affects a binary state '1' of the output, if the activated David cell belongs to the set of connected cells. As mentioned before every component of the interface logic exhibits 4-phase behaviour required for the main controller. This behaviour is obtained by a logical AND-combination of the described NAND-gates and the trigger signal.

The 4-2-phase-converter is depicted in Figure 17. It consists of two D-Latch elements with acknowledgment function, which are assigned to the individual channels. They are working as frequency dividers with base 2. For the generation of the acknowledgment signal, the acknowledgment signals are combined using an XOR-gate. Because during every transmission of data only one channel is activated, the XOR-gate produces an acknowledgment signal for every transmitted data.

5.3.2 Receive Interface Implementation

The structure of the receive interface is shown in Figure 18. The generation of complete pulses for every incoming edge on the individual channel is done using a combination of a D-Latch and an XOR-gate in the 2-4-phase-converter (see Figure 19). The change of the state of the channel will be di-

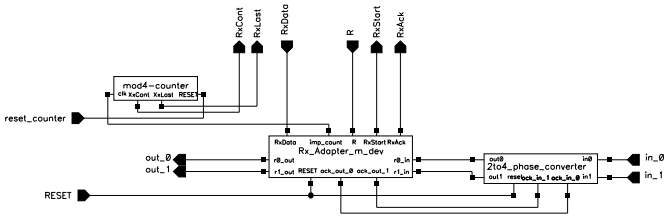

Figure 18. Receiving interface in place-to-latch version

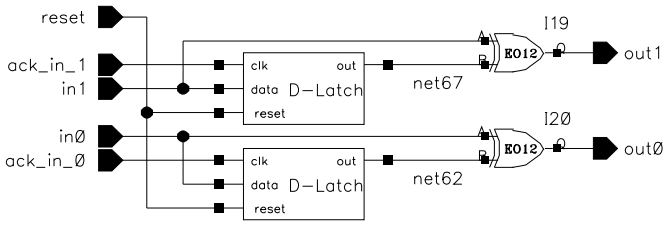

Figure 19. 2to4 phase converter

rectly forwarded to one input of the XOR-gate. This causes a '1' state at the output of the XOR-gate. The acknowledgment signal from the Rx-adapter triggers the adoption of the state of the channel by the D-Latch circuit. The output of the D-Latch is connected to the second input of the XOR-Gate. Thus the acknowledgment of the Rx-adapter resets the binary state '1' on this output channel.

The Rx-adapter is shown in Figure 20. It has to direct the incoming signals to the appropriate input channels and has to produce the control signals RxStart and RxAck for the main controller. The direction of the signals is controlled by the output signals of the main controller. The described functionality is achieved using C-elements. There are two groups of C-elements for both signal channels. One set of C-elements is enabled if the controller expects control signals like RxStart or RxAck. The other set of C-elements is enabled, if the received signals are interpreted as data.

In the case of the expectation of control signals, the R signal of the main controller is activated until the reception of one of these signals. If data signals are expected the RxData signal of the main controller is activated. The data will be forwarded to the destination system. In parallel with this forwarding the counter of the receive interface is incremented. The output signals of the counter are connected either to the RxCont input or to the RxLast input of the main controller. The activation of one of these signals indicates the reception of data and the deactivation of the RxData signal.

The counter as a sub-circuit works totally in the same way like the counter of the send interface.

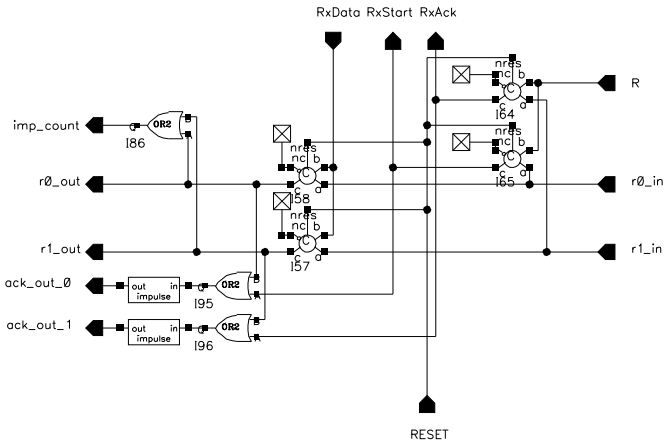

Figure 20. Rx-Adapter of the place-to-latch version

6 Speed-up Strategies

This section discusses the speed-up strategies and improvments achieved during the redesign phase.

6.1 Place splitting and dummy insertion

Decomposition of places avoids the occurrence of large fan-in-trees. Considering the most important paths of the token during the data transmission, it was realised that David cell structures require a precise analysis to explore the possibilities of the optimisation of the system structure. In this context, the cells Rx1 and Tx1 in both Master and Slave were looked at. It is easy to observe that the token fan-in of the places Rx1 and Tx1 in the LPN of Figure 10 is quite large because it involves merging the token flow from three different modes: intialisation, unidirectional and bidirectional transfers. The David cells Rx1 and Tx1 therefore have very complex logic gates, due to the number of inputs and outputs. A speed-up is expected if we split each of these places into two places with a lower complexity of the gates in the circuit. The decomposition on the LPN level is shown in Figure 21, where new merge places Tx12 and Rx12, and corresponding dummy events are introduced. The reader can compare these LPNs with the LPNs in Figure 10.

Another performance issue is concerned with the positioning of dummy transitions. Remember that some dummies are inserted into LPN cycles to avoid loops of less than three David Cells because it may otherwise lead to a deadlock in the control logic [4]. On the other hand it is clear that the presence of a dummy transition between two places implies extra delay between the preceding and the following actions. For example, the presence of a dummy between Tx4 and Tx1 in Figure 10 means that there is a delay between the processing of the pull (R $\rightarrow$ RxAck) handshake (before place Tx4) and the push handshake started by TxData (after place Tx1). This delay is on the critical path of the protocol and thus slows down the overall communication in the Tx mode. It would be possible to avoid this slow down by changing the position of the dummy, e.g. by inserting it into a split of place Tx3, which is between the push and pull handshakes. This would effectively mean inserting a delay in parallel with the operation of the Slave. Similar transformation can be done with the dummy between RxTx2 and TxRx1 in Figure 10, which slows down the duplex communication. This dummy is shifted into the split of the RxTx1 place.

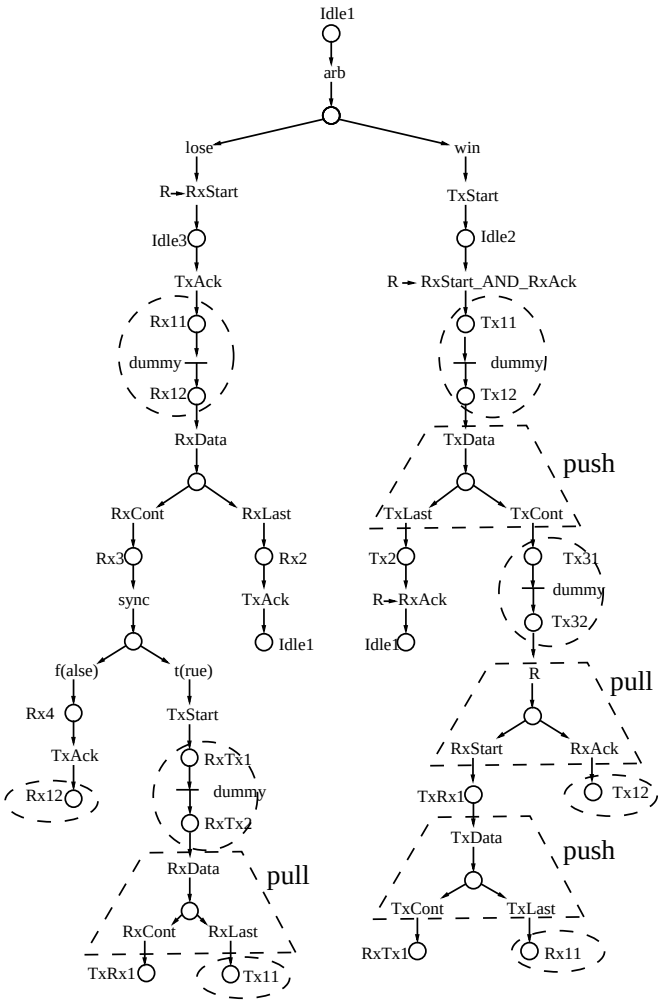

Figure 21. LPN of master controller after optimisation

6.2 David Cell Protocol

A number of usual David cells were replaced by special David cells, with pre-enabling of the consequent transitions.

The pre-enabling of the next operation is possible, because of a handshake specified by the controller protocol on a higher abstraction layer. Usual David cells wait until the previous operation is finished, before they start to transmit the token to the next block of the data path. In our case, a handshake on the higher level is defined, which ensures that another data transmission cannot be started until we get a response (RxAck, RxStart or data) from the communication partner. This response can only be triggered by the completed transmitting operation of the last cycle. It is not possible to receive a response signal without a completed transmission of data or signals, and vice versa. This behaviour is depicted in Figure 22. The STG of the David cell with pre-enabling strategy includes an additional arc depicted as a dashed line. This arc stands for a set of processes in the system, which have strongly defined series of executions. These processes are not part of the circuit itself. It is possible to predict the behaviour of the circuit in conjunction with signals from the environment, according to the specification of the whole system. This arc has to be considered as a black box which could involve a number of different firing sequences. The token placed on this transition has to be interpreted as a token of the protocol covering the whole system. The place-to-latch compilation approach was linked with special firing conditions in the system as defined by the protocol. The timing diagrams in Figure 23 clarify the differences between the protocols. To ensure the sufficient work of the cell, a further condition has to be considered. It is necessary to execute the req_out- transition before the request-input of the David cell is deactivated. The designer has to ensure that the delay time for reseting the request-input is long enough to provide the sufficient switching process of the C-element. The condition is shown as the dashed line in the STG named with "timing assumption".

The application of our speed-up strategies is only part of redesign, after a sufficient check of the individual functionalities of the subsystems. Another point is that a decision whether or not to use this form of David cells must be made by considering the different areas inside a subsystem. Elements of the data path only deal with internal signals of the system. That means, if they do not require a defined handshake of the environment between David cells, then these components cannot be connected by such David cells.

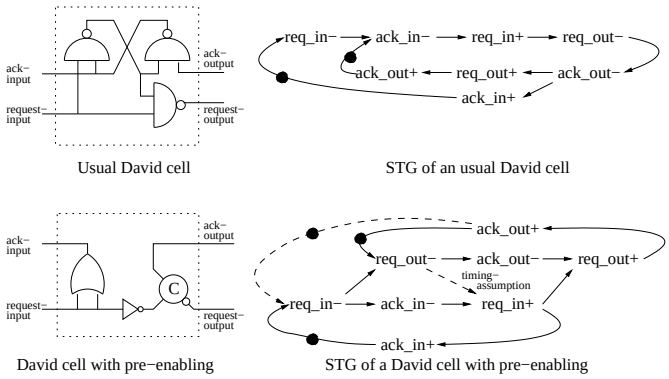

Figure 22. Comparison between usual David cells and David cells with pre-enabling strategy

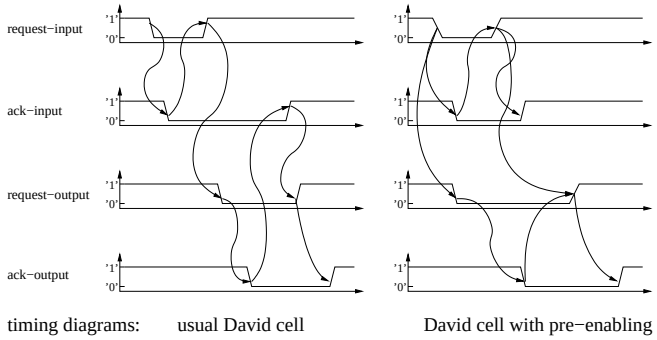

Figure 23. Timing diagrams of an usual David cell and a David cell with pre-enabling

6.3 Analysis of Trigger Signals in Digital Circuits

The following strategy was applied during the implementation of the minimisation version of the controller. Petrify produces an state graph (SG) of the entire STG. It is then possible to obtain a projection of the SG on a subset of signals of the controller circuit. The projection includes only the relevant input signals and the output signal of a individual sub-circuit, asscoiated with the output signal. This reduced SG preserves behavioural equivalence to the original SG with respect to that subset of signals. The formal justification for this STG decomposition technique can be found in [8].

The optimization of the implementation can be divided into two sections. The first section is the identification of the so-called trigger signals of the sub-circuit. The second section is the technology mapping, where special features of the given target system are used. This form of optimisation has to be done carefully, because it is easy to violate the equivalence of the logic behaviour of the original circuit in comparison to the optimized version.

Trigger signals are signals whose transitions are immediate predecessors to the transitions of the output signal in the

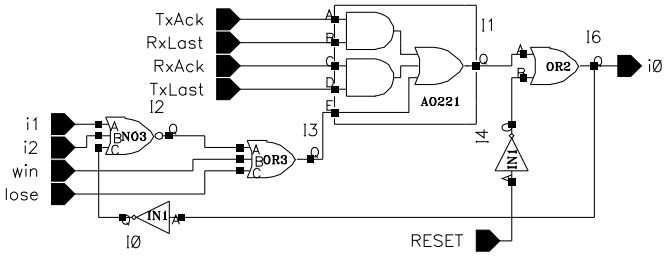

Figure 24. Implementation of the i0 sub-circuit of the slave controller in minimisation version

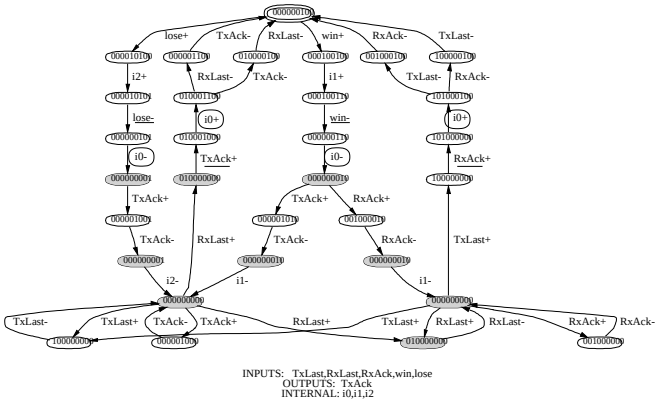

Figure 25. SG of the i0 signal of the slave controller synthsised using the method based on logic minimisation

STG. The individually produced SG simplifies the search for these signals. During the implementation of the sub-circuit the designer should use a very small number of gates between these signals and the output signal. The other input signals can be implemented with a lower priority.

The following example will explain the applied method. The logical equation

```
[i0] = i0 i2' i1' + TxAck RxLast +
 RxAck TxLast + win + lose;
```

is the description of the behaviour of the signal i0 of the slave controller synthesised by using the minimsation method. Figure 25 shows the SG which describes the logical behaviour of i0. The underlined signals are the trigger signals of i0. Figure 24 depicts the implemented circuit. The designer has to insert a very small number of gates between the trigger signals and the output signal, and he/she has to keep the fan-in-tree of the individual gates as thin as possible.

This method is based on the prediction that all signals have similar speed. If it is assumed that very fast series of input signals in contrast to very slow signals, this method might cause a speed-down effect. That means the definition of trigger signals has to be done depending on the position of the transition of the signals inside the SG and the ratio of the expected time delays. The analysis of the mentioned parameters is outside the scope of this paper.

Furthermore the designer has to consider the possibility of producing hazards. Direct logic combinations of concurrent signals should be avoided. This is a requirement especially for concurrent trigger signals.

7 Analysis of the Communication System

In this section simulation tests and their results are described.

7.1 Simulation Test Cases

To prove the whole functionality of the produced circuits test cases have to be applied to check all possible data exchange modes, the crossing from one mode to another and different sequences of starting the communication process. All these cases cannot be checked during one test run. A sufficient check of the system requires a number of test runs, which cover all necessary test cases.

Furthermore the test has to be adapted to the desired purpose of the analysis. In the considered case the priority of the tests was directed to a performance analysis. That means the relevant value was the actual speed of the controller system during the transfer modes. The system was not tested with respect to reliability of the data transfer, speed of changing between the transfer modes, or fairness during the initialisation process. Such tests would require different set of test cases and another class of accuracy of test values.

7.2 Results

The controller in the place-to-latch version was tested at different design stages to produce a meaningful comparison in connection with the discussed speed-up strategies. The results of all possible transfer modes at the beginning of the design process and after the application of the speed-up strategies were compared. But the main point of this analysis was the comparison between the performance achieved by the controller developed using the minimisation and that of the controller built using the place-to-latch method.

Because the optimization process of the minimisation version of the controller was done at the beginning of the design process, there is only one set of results of a perfor-

	unidir mode	bidir mode
place-to-latch no speed-up	9.99ns	12.35ns
place-to-latch after speed-up	7.67ns	8.32ns
logic min.	12.7ns	16.5ns

Table 1. Results of the performance tests of both controller versions

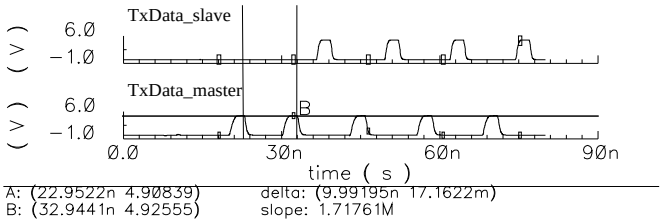

Figure 26. Unidirectional transfer mode before speed-up. place-to-latch version

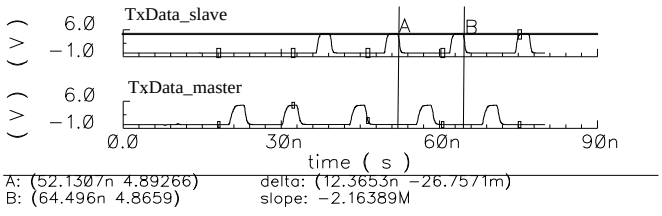

Figure 27. Bidirectional transfer mode before speed-up. place-to-latch version

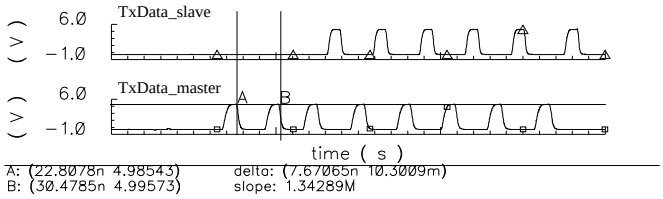

Figure 28. Unidirectional transfer mode after speed-up. place-to-latch version

mance test. The time for the unidirectional modes using the minimisation version of the controller is different for the transfer of data from the master or from the slave. That is caused by the differences in the complexity of the sub-circuits, which are involved in the individual work-flow. In contrast to the minimisation version of the controller the place-to-latch version uses exactly the same sub-circuits in these transfer modes, therefore the delay is independent of the current destination system and the current source system.

Figures 26 to 32 illustrate waveforms from the simulation tests for verious data exchange modes. The results of these tests are summarised in Table 1.

8 Conclusions

As mentioned above, the comparison between two circuit implementation approaches was the focus of this analysis. In the past circuits designed by using the place-to-latch method were slower than circuits produced by using minimisation methods. The delay of the additional logic which was required by the place-to-latch method was too high to compensate for the advantages of the direct mapping approach. The latter normally come from the simplicity of individual elements of the control logic, which is 'thinly' distrubuted between several David cells. If the aspects of performance are considered, the direct mapping method should be efficient in case of designing complex controllers, particularly where the use of minimisation method would require solving a large number of Complete State Coding conflicts and where the same control signals occur in the Petri net specification in many places.

The speed-up phase of the controller in place-to-latch version consists of two parts:

- Place fanin reduction and dummy positioning
- Optimization of the David cell structure

The *decomposition* of a number of places in the controller had the intention of reducing the size of the gates used inside the David cells. The simulations have shown, that the achieved speed up was not appreciable. Therefore the results of the simulations of this design stage are not discussed. On the other hand this form of optimization should not be ignored by the designer. The use of less complex David cells will have advantages during parts of the design like the technology mapping and the routing of the circuit.

After these results it was clear that it was necessary to reduce the actual number of transitions in the system performed in series. The *optimization of the David cell structure* caused a crucial increase in the performance of the controller. As described in section 6.2 this form of optimization can be applied during the redesign phase and under special conditions. These conditions are met by systems, where the interaction of two sub-systems is defined as serial. This behaviour is common to communication controller systems. These systems will be the preferred area for the application

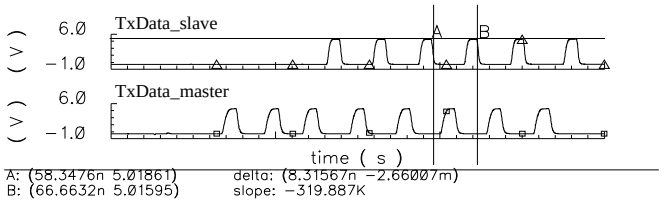

Figure 29. Bidirectional transfer mode after speed-up. place-to-latch version

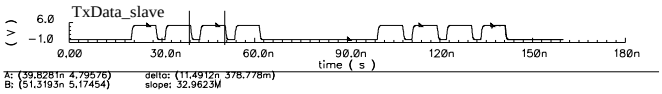

Figure 30. Unidirectional transfer mode from slave, minimisation version

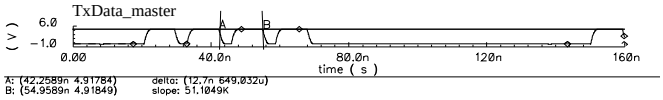

Figure 31. Unidirectional transfer mode from master, minimisation version

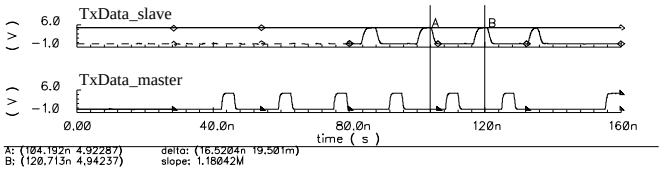

Figure 32. Bidirectional transfer mode, minimisation version

of this form of optimisation. Theoretical analysis will generate a much more precise description of the possible application area. Some techniques enhancing the performance of David cell structures by applying relative timing at the transistor level have been reported in [9].

The speed-up effect of the applied analysis of trigger-signals of the sub-circuits was not verified, because of the above mentioned reasons.

References

[1] L. Benini and G. De Micheli, Networks on Chips: a new SoC Paradigm, Computer, vol. 35, No. 1, Jan. 2002, pp. 70-78.

[2] S.B. Furber, A. Efthymiou and Montek Singh, A power-efficient duplex communication system, In: A. Yakovlev, R. Nouta (Eds.) Proceedings of Int. Workshop on Asynchronous Interfaces: Tools, techniques, and implementations (AINT'2000), TU Delft, The Netherlands, July 2000, ISBN 90-5326-037-4.

[3] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno and A. Yakovlev. Logic Synthesis of Asynchronous Controllers and Interfaces. Springer, 2002.

[4] A. V. Yakovlev and A. M. Koelmans. Petri Nets and Digital Hardware Design. Lectures on Petri Nets II: Applications. Advances in Petri Nets, Lecture Notes in Computer Science, vol. 1492, Springer-Verlag, 1998, pp. 154-236.

[5] A. Semenov, Verification and Synthesis of Asynchronous Control Circuits Using Petri Net Unfoldings, PhD Thesis, University of Newcastle upon Tyne, July 1997.

[6] J.-T.Udding, Classification and Composition of Delay-Insensitive Circuits, PhD Thesis, Eindhoven University of Technology, 1984.

[7] D. Shang, F. Xia and A. Yakovlev, Asynchronous circuit synthesis via direct translation, accepted for ISCAS 2002.

[8] W. Vogler and R. Wollowski, Decomposition in asynchronous circuit design, TR, University of Augsburg, 2002.

[9] A. Bystrov and A. Yakovlev, Asynchronous circuit synthesis by direct mapping: interfacing to environment, Proc. of Async'02, 2002, IEEE CS Press, pp. 127-136.