An Algorithm for Synthesis of System-Level Interface Circuits*

Ki-Seok Chung Rajesh K. Gupta C. L. Liu

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801

Abstract

We describe an algorithm for the synthesis and optimization of interface circuits for embedded system components such as microprocessors, memory ASIC, and network subsystems with fixed interfaces. The algorithm accepts the timing characteristics of two system components as input, and generates a combinational interface (glue logic) circuit. The algorithm consists of two parts. In the first part, we determine the direct pin-to-pin connections in the interface circuit employing a 0/1 ILP formulation to minimize wiring area and dynamic power consumption. In the second part, we determine logic subcircuits in the interface circuit, utilizing the timing diagrams of the system components. The proposed algorithm has been implemented in a software package SYNTERFACE. Experimental results are presented to demonstrate the effectiveness of the algorithm.

1 Introduction

To reduce the complexity of system integration, predesigned components are used more and more often. Use of predesigned off-the-shelf components, such as microprocessor, memory, and macro circuit block leads to reduction in cost and design time in complex systems. However, these components often have different interface characteristics and require interface circuit to connect them together. Depending on their interface timing characteristics, these components are connected together either directly, or through combinational, or sequential glue logic circuits. In this paper we address the problem of automatic generation of an interface circuit between two predesigned components.

The generation of control signals with the minimum number of additional gates is a difficult problem. Consider the following example.

Example 1.1. Figure 1 shows an example on the design of an interface circuit between a microprocessor and a SRAM chip. Each pin on a chip has a fixed electrical characteristic. It is either an *input* or an *output* pin depending on whether it is driven by, or drives, an electrical signal, respectively. There are several ways to connect the two chips in Figure 1-(a). We show two different ways in Figure 1-(b) and 1-(c). The costs of the interface circuits, however, are different in the two cases. One way to quantify the cost is to examine the number of signal transitions as it is directly related to power consumption in current technology.

Also in Figure 1-(c), we can see that the $AS*$ signal is used to drive three input pins. Such sharing of output signal by as many input pins as possible typically results in area minimization in the interface circuit, reducing the number of wires from one chip to the other. Therefore, another good design objective is to maximize pin sharing within the fanout limitation while reducing the amount of glue logic. □

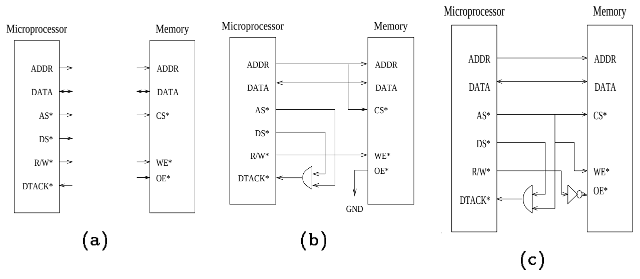

Figure 1: Pin layout of components to be interfaced and two different designs of the interface circuit

The problem of interface design has been studied earlier in the context of timing analysis and verification. In [2], methods to synthesize interface circuit based on interface templates were presented. In [3], methods to synthesize interface blocks that consist of logic circuits and/or software routines as straight-line codes were presented. The design of interface circuit from a high-level specification, such as interface protocols, was studied in [4, 5, 7, 8]. A set of interface co-synthesis techniques for the synthesis of hardware/software interfaces was presented in [6].

Our work differs from previous works in two respects: we present a procedure to synthesize (a) an optimal direct pin-to-pin interconnect networks and (b) an optimal combinational glue logic circuit.

This paper is organized as follows. In Section 2, we present our problem formulation. In Section 3, we present our synthesis algorithm. In Section 4, we present

*The work was partially supported by the National Science Foundation under grant MIP-9222408 and NSF Career Award MIP95-01615.

experimental results. Section 5 concludes the paper.

2 Problem Statement

A signal on a pin may have four possible **actions**: (a) transition from high to low, (b) transition from low to high, (c) becoming valid, and (d) becoming invalid. The last two actions are relevant in the case of multi-bit (or data) pins. A signal action on a pin x is denoted $x+$ or $x-$. $x+$ represents *becoming valid* or *transition from low to high* and $x-$ represents *becoming invalid* or *transition from high to low.* An **event** refers to an instance of an action with a time stamp. Let $T(e)$ denote the time at which event e takes place. A **protocol** is a partial ordering relation over a set of events. A protocol is often described by a protocol flowchart. A **communication operation**(which will also be referred to as "operation" for brevity), such as a "memory read," is defined by a protocol consisting of a sequence of events.

For our synthesis problem, the inputs are the data sheets for the components(or chips) to be connected together and the protocol flowcharts for the operations between them. From the data sheet of a chip, we can obtain information on (i) the pin description which specifies the operations in which each pin is involved and whether the pin is an input pin or an output pin for each of the operations, and (ii) timing diagram (with a parameter table) which describes the timing relationships between events in terms of order and max/min separation. Also from the timing diagram, we can determine, for each pin, the total number of transitions for a given operation. The minimum timing separation between two events e_1 and e_2 is denoted $\text{min\_sep}(e_1, e_2)$, which means $T(e_2) - T(e_1) \geq \text{min\_sep}(e_1, e_2)$. The maximum timing separation between the two events is denoted $\text{max\_sep}(e_1, e_2)$ which means $T(e_2) - T(e_1) \leq \text{max\_sep}(e_1, e_2)$. We assume that $\text{max\_sep}(e_1, e_2)$ is always nonnegative which implies that event e_1 occurs no later than event e_2.

There are two types of timing relationships in a protocol for each operation (such as data or control transfer).

- *Specified timing*: It refers to the min/max timing separation between two events where the timing relation will always be met when the operation is performed.
- *Required timing*: It refers to the min/max timing separation (between two events) which is required to happen in order to perform the operation correctly. Such timing relation is a required timing.

We address the problem of interface circuit design by considering two components at a time. We can extend our work to the design of interface circuits in a multichip environment by carefully deciding the order in which two of the several components are considered. Therefore, the problem of interface synthesis is stated as follows: **Given an input description consisting of the timing diagram and protocol flowchart of two chips, synthesize an interconnect circuit between the two chips such that the interface cost is minimized.**

The input information for our synthesis algorithm is represented by signal transition graphs [1]. The vertices in an *STG* represent actions on the pins. The edges in an STG represent timing constraints. There are two types of edges corresponding to minimum and maximum timing constraint between actions. Figure 2 shows an example of a timing diagram and its corresponding STG. The dotted arrows represent *maximum timing constraints*, and the solid arrows represent *minimum timing constraints.* The *required* and *specified* timing replationships between events are expressed in two separate STGs, namely the **requirement STG** and the **specification STG**. The causality relations obtained from the protocol flowchart is implicitly represented in the STGs as minimum timing constraints.

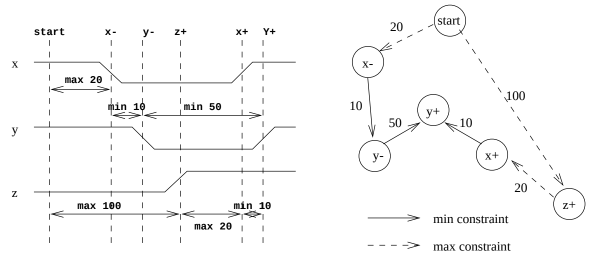

Figure 2: A timing diagram and an equivalent signal transition graph

3 Modeling and Synthesis of System Interface

A combinational interface consists of three types of connections:

- *Type 1* Connection: A pin is connected to the power supply(VDD) or ground(GND).
- *Type 2* Connection: A direct connection between a driving pin and a driven pin.
- *Type 3* Connection: A connection which requires logic gates between driving signal(s) and a driven signal.

We assume that a *Type 1* connection is least expensive and a *Type 3* connection is most expensive. To minimize the interface cost we shall try to maximize the usage of *Type 1* and *Type 2* connections. The overall design flow is shown in Figure 3.

Generation of interface circuit is carried by examining the *compatibility* of pins. We define the notion of *compatibility* between a driving and a driven pin as follows.

Definition 3.1 **(Compatibility)**: For an operation, a driving(output) pin is **compatible** with a driven(input) pin if the driving pin can trigger the driven pin while satisfying all required timing constraints.

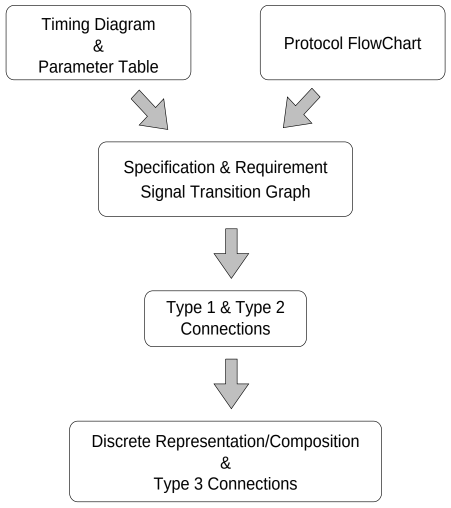
Timing Diagram & Parameter Table
Protocol FlowChart
Specification & Requirement Signal Transition Graph
Type 1 & Type 2 Connections
Discrete Representation/Composition & Type 3 Connections

Figure 3: Overview of the interface synthesis algorithm.

3.1 Determination of Type 1 and Type 2 Connections

To minimize the wiring area and dynamic power consumption, our algorithm attempts to maximize the sharing of output pins. If there are more than one compatible output pins, we select the one with the least number of transitions to reduce dynamic power consumption. The algorithm for the determination of *Type 1* and *Type 2 connections* is based on two different STGs.(specification STG and requirement STG) A specification STG is constructed for each operation on each chip. One requirement STG is defined for one operation. The determination of the *Type 1* and *Type 2* connections is done by the following three steps.

STEP 1: Pin Compatibility Checking by Merging STGs

For a given operation, we inspect the timing diagram of each driving pin to check whether there are any driven pins that can be triggered by it.

For each operation, we merge the two specification STGs of the two chips as follows. The vertices corresponding to an input pin in one STG are merged with the vertices corresponding to an output pin in the other STG. The merging is done with respect to every pair of input pin and output pin in the two specification STGs to test the compatible connections. After merging of an input and an output, the input pin will take on the same timing characteristic as the output. Hence, the merged graph of the two specification STGs will represent the timing characteristics of the two corresponding components when the input pin is connected to the output pin. A merged graph is not feasible, i.e. cannot be implemented by any circuit, if it contains a positive cycle.

We check whether each of the required timing constraints is satisfied using the merged STG and the requirement STG. A minimum separation requirement $A \xrightarrow{r} B$ is guaranteed to be satisfied only when the length of the longest path between A and B in the merged specification STG is greater than or equal to r. A maximum separation requirement can be handled similarly since $A \overset{l}{\dashrightarrow} B$ is equivalent to $B \xrightarrow{-l} A$. To test whether a required timing can be satisfied, we compute the length of the longest path for all pairs of vertices in the merged specification STG using the Floyd-Warshall algorithm. Since a connection between an input pin and an output pin should guarantee none of the required timings is violated, we test one requirement at a time and repeat the step until all the edges in the requirement STG have been tested.

STEP 2: Operation Compatibility Using Interface Graphs

For each operation, we build an interface graph. **Interface Graph** $G_B(V_1 \cup V_2, E)$ is a bipartite graph where V_1 is a set of vertices corresponding to the driving pins, and V_2 is a set of vertices for corresponding driven pins. There is an edge (u, v) in E if v is a driven pin which can be triggered by u. An edge (u, v) is given a weight which reflects the effectiveness when v is triggered by u. In the current implementation, we determine the weight of each edge based on the total number of transitions in the corresponding driving signal for all the operations.

After building the interface graph for each operation, we construct a graph which is the intersection of all interface graphs. We call this the **intersection graph**. The intersection of two graphs is a graph that contains only those edges that are in both graphs. The intersection graph is used to determine an interface circuit that would support all operations.

STEP 3: Signal Selection

The selection of a driving pin is made according to two criteria. The first is to maximize the sharing of a driving signal. Maximizing the sharing of a driving signal under fanout constraints will most likely lead to a minimization of the wiring area. The second is to minimize the dynamic power consumption by selecting a driving pin so that the pair of driving and driven pins have the least total number of transitions. We formulate this as a 0/1 ILP problem.

0/1 ILP Formulation

Let V_{out} be a set of vertices each of which represents a driving(output) pin in the intersection graph, and let V_{in} be a set of vertices each of which represents a driven(input) pin which has at least one incident edge in the intersection graph. Let s_i be a decision variable corresponding to $v_i \in V_{out}$. Let c_{ij} be a decision variable corresponding to each edge between $v_i \in V_{out}$ and $v_j \in V_{in}$ in the intersection graph. Let W be a set of weights each of which is defined for each $v_i \in V_{out}$. In a solution to 0/1 ILP, $s_i = 1$ means that v_i will be used to drive input pins in the interface. Otherwise, it will be 0. The $c_{ij} = 1$ means that v_i will drive v_j. Otherwise, it will be 0. Thus, $s_i = 1$ only if there exists at least one $c_{ij} = 1$ for some j. Let $w(v_i)$ denote the weight for $v_i \in V_{out}$. Let E denote the set of edges in the intersection graph. Let N_i denote the number of edges incident with the output vertex v_i in the intersection graph. The objective function consists of two weighted sums. The first sum is a measure of the wiring area weighted by the number of transitions in the driving signal and the second sum is used to ensure that a connection is unique while considering the additional switching activity due to multiple fanout. Thus, the objective function to be minimized is:

$$Cost \ = \alpha \cdot \sum_{v_i \in V_{out}} w(v_i) \cdot s_i \ + \beta \cdot \sum_{(v_i, v_j) \in E} w(v_i) \cdot c_{ij}$$

where α and β are user defined weighting factors.

The constraints are:

$$\forall v_j \in V_{in}, \qquad \sum_{v_i \in V_{out}, (v_i, v_j) \in E} c_{ij} = 1$$

$$\forall v_i \in V_{out}, \quad s_i - \frac{1}{N_i} \cdot \sum_{v_j \in V_{in}, (v_i, v_j) \in E} c_{ij} \geq 0$$

3.2 Synthesis of *Type 3* Connections

After deciding all *Type 1* and *Type 2* connections, each unconnected pin is considered for a *Type 3* connection.

3.2.1 Discrete Representation of Signals

A discrete representation for a signal is obtained by discretizing its waveform. A clock cycle is sub-divided into micro-timesteps. We assume that the size of a micro-timestep is small enough to enable us to capture all the transitions of a waveform in a clock cycle, but large enough so that the waveform will not be changed by the possible additional delay due to the introduction of gates. With the min/max constraints, a driving signal can be represented by a sequence of 0's, 1's, and u's. u stands for "unknown" which means that a transition may occur in the micro-timestep. A required signal is represented by a sequence of 0's, 1's, and d's.(d stands for "don't care".) Let D_S denote the discrete representation of signal S. Figure 4 shows an example of discrete representations of driving signals. The upper time limit of the representation is the maximum number of micro-timesteps within which all the relevant signal actions have occurred for a given set of communication operations. The upper time limit of the timestep is usually available from the datasheet. To synthesize a driving signal for a *Type 3* connection from available driving signals, we need to generate the 0's and 1's properly in the required signal, and need not pay attention to the d's.

Suppose $DTACK$ is a pin for which we want to find a *Type 3* connection and there are four available signals,

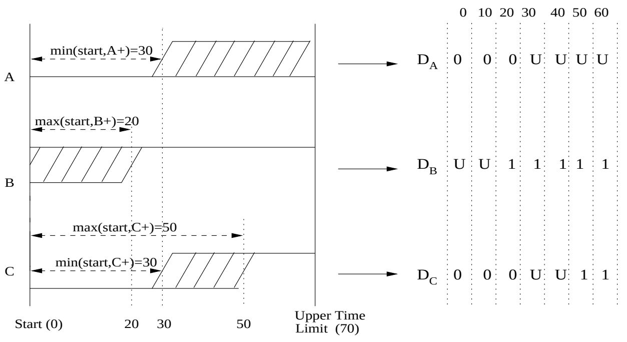

Figure 4: Discrete representations of driving signals

AV(Address Valid), AS, DS, and $R/\bar{W}$ as in Example 1.1. For the READ and the WRITE operations, the discrete representations of signals are shown in Table 1.

READ (time scale: 10, micro-timestep size: 3)

TIME(ns)	0	3	6	9	12	15	18	21	24	27	30
D_{AV}	u	u	u	1	1	1	1	1	1	u	u
D_{AS}	1	1	u	u	0	0	u	u	u	u	1
D_{DS}	1	1	u	u	u	u	0	0	u	u	1
$D_{R/\bar{W}}$	u	1	1	1	1	1	1	1	1	1	1
D_{DTACK}	1	1	d	d	d	0	0	d	d	d	1

(a)

WRITE (time scale: 10, micro-timestep size: 3)

TIME(ns)	0	3	6	9	12	15	18	21	24	27	30
D_{AV}	u	u	1	1	1	1	1	1	u	u	u
D_{AS}	1	u	u	u	0	0	0	u	u	u	1
D_{DS}	1	1	1	1	1	u	u	0	0	u	1
$D_{R/\bar{W}}$	1	u	u	u	0	0	0	0	0	0	u
D_{DTACK}	1	d	d	d	d	d	d	0	0	d	1

(b)

Table 1: Signals for synthesizing $DTACK$ for (a) READ and (b) WRITE

3.2.2 Signal Composition

Available driving signals can be combined to yield "composite" driving signals. We are looking for a composite driving which matches the required signal in all the 0's and 1's.(The d's are ignored.) A driving signal that contains some u's cannot be used by itself to synthesize the driving signal since we do not know the exact value of the signal at the micro-timesteps containing the u's.

Thus we try to eliminate the u's as much as possible in order to maximize the number of available signals. We present an algorithm based on signal compositions. The following composition rules are used for the composition of driving signals containing u's.

- Conjunctive(AND) composition with u:
  $$\mathrm{u} \cdot 0 = 0, \quad \mathrm{u} \cdot 1 = \mathrm{u}, \quad \mathrm{u} \cdot \mathrm{u} = \mathrm{u}$$
- Disjunctive(OR) composition with u:
  $$\mathrm{u} + 1 = 1, \quad \mathrm{u} + 0 = \mathrm{u}, \quad \mathrm{u} + \mathrm{u} = \mathrm{u}$$
- Complementive(NOT) composition of u:
  $$\neg \mathrm{u} = \mathrm{u}$$

We call a (composite) signal **valid** if it does not contain any u. And we call a composition **valid** if the resulting signal of the composition is valid.

3.2.3 A Heuristic Algorithm for Determination of Valid Composite Signal Set

An exhaustive search of all possible compositions of a pair of signals to find all possible valid composite signals leads to an exponentially large number of composite signals. Therefore, we suggest a heuristic algorithm for finding a set of valid signals from a given signal set. First we define the following concept.

Definition 3.2 **(Null-Representation)**: A representation D_A that contains only u and 1, or only u and 0, is called a *null-representation.*

Using the concept, we present a heuristic algorithm as shown below. Let D be the set of original representations of all available signals. The algorithm takes D as the input, and returns D' which is the new set of valid signals which are obtained from the D through NOT, AND and OR compositions.

```
Procedure Generate_Valid_Signals($D$)
  Initialize $D'$ to $D$
  While $\exists$ a new signal to consider
    For each pair ($D_A, D_B$) which hasn't been considered yet
      Generate_Composite_Signals($D_A, D_B$)
      Discard all null representations
      Remove all duplicated signals
      Check the equivalence to all the existing signals
      Update $D'$ by adding/deleting representations
    endFor
  endWhile
  Eliminate all the invalid representations from $D'$
  return $D'$
endProcedure
```

Procedure `Genrerate_Composite_Signals(`D_A, D_B`)` micro-timestepwise inspects occurring patterns between two representations and generates only the minimum set of useful compositions.

3.2.4 Finding a Common Expression

Since there will be, in general, different driving signals in the same driving pin for different operations, we need to find a composition of signals that is common to all operations. For example, from Table 1-(a), the set of valid signals for READ is $\{D_{AS} \cdot D_{DS},\ D_{AV} \cdot \neg(D_{AS} \cdot D_{DS}),\ D_{R/\bar{W}} \cdot \neg(D_{AS} \cdot D_{DS})\}$. From Table 1-(b), the set of valid signals for WRITE is $\{D_{DS},\ D_{AS} \cdot D_{DS},\ D_{R/\bar{W}} + D_{DS}\}$. The intersection of the two sets is $\{D_{AS} \cdot D_{DS}\}$. Thus, we can build a truth table for both operations as shown in Table 2. And we can determine whether there is a common expression or not. In this case, $D_{AS} \cdot D_{DS}$ is the expression for D_{DTACK}.

3.3 Interface Algorithm Implementation

Our algorithm for interface synthesis is shown by procedure `SYNTERFACE`. The algorithm takes as input S^A, S^B and R which are the set of specification STGs of chip A, the set of specification STGs of chip B, and the set of all the requirement STGs, respectively. Let S_i^A be the specification STG of chip A for the ith operation(S_i^B is defined similarly.), and let R_i be the requirement STG for the ith operation. Also the interface graph for ith operation is denoted I_i.

Truth Table for D_{DTACK}

$D_{AS} \cdot D_{DS}$	D_{DTACK} for READ	D_{DTACK} for WRITE
0	0	0
1	1	1

Table 2: Finding a common expression for a signal $DTACK$

```
Procedure SYNTERFACE($S^A$,$S^B$,R)
  /* PART 1 */
  For each operation $i$
    For each driving/driven pairs of $S_i^A$ and $S_i^B$
      M ← merge_Graph for a pair of input and output pins
      For each edge of $R_i$
        Test $M$ satisfies the requirement edge
      endFor
      If $M$ satisfies all the requirements
        Draw an edge on $I_i$ between the pins considered
      endIf
    endFor
  endFor          /* At every iteration, construct $I_i$ */
  Obtain $I$
  Formulate and Solve a 0/1 ILP to determine connections
    for Type 1 & 2
  /* PART 2 */
  Determine the size of the microtime step $\Delta$
  For each pin for Type 3 connection
    For each operation
      Encoding signals and obtain composite signals
      Finding valid boolean expressions
    endFor
    Determine a Type 3 connection by finding a common
      expression
  endFor
endProcedure
```

4 Experimental Results

The interface synthesis algorithm was implemented in approximately 3000 lines of C++ code. Experiments were performed on a Sun Sparc10. We use MPOS[1] to solve the 0/1 ILP problem. We tested our algorithm on several commercial chips. We present here some examples and Table 3 summarizes the results.

1. MC68000 and MC6850 ACIA

The MC6850 [11] Asynchronous Communication Interface Adaptor (ACIA) is a general purpose interface device which provides the data formatting and control functions to interface serial asynchronous communication between a microprocessor and an external system. In this example, we only consider the interface circuit between the microprocessor MC68000 and MC6850 ACIA. As mentioned previously, our design algorithm focuses

[1]Multi-Purpose Optimization System(MPOS) is a program from Northwestern University for solution of linear, integer and quadratic programs.

Experiment	Com. Op.	Spec. Con.	Req. Con.	T1	T2	T3	NC
MC68000/ MC6850 (8MHz)	RD	19	17	0	5	1	0
	WR	25	19				
	AVI	11	3				
8085AH-2/ 8231A(4MHz)	S/R	16	10	1	4	1	0
	S/W	15	7				
80186/ 82530 (4MHz)	RD	42	14	2	3	2	2
	WR	36	15				
	INT	35	9				

Table 3: Summary of the experimental results

on the control glue logic for proper communication between two chips, and some necessary decoding logic or priority encoding chips are assumed to have been provided.

2. Intel 8085AH-2 and Intel 8231 Arithmetic Processing Unit

In this experiment an 8 bit microprocessor, Intel 8085AH-2 [9] is interfaced to an Arithmetic Processing Unit, Intel 8231A [10]. The APU provides high performance floating-point computation.

3. Intel 80186 and Intel 82530 Serial Communications Controller

The Intel 82530 Serial Communications Controller [12] is a dual-channel, multiprotocol data communications peripheral. In this experiment, we consider the interface between an 8 Mhz 80186 and a 6 Mhz 82530 when they are clocked at 4 Mhz for READ, WRITE, and INTERRUPT operations.

SYNTERFACE has automatically generated combinational interface circuits for the above experiments. Table 3 and Figure 5 summarize the input informations and the result for the above experiments. `Com.Op.` lists operations that the resulting interface circuit supports. `Spec.Con.` and `Req.Con.` are the number of specified timings and required timings, respectively. `T1`, `T2`, and `T3` show the number of *Type 1*, *Type 2*, and *Type 3* connections in the resulting interface circuits. `NC` is the number of input pins for which SYNTERFACE could not synthesize a proper driving signal. In Experiment 3, we found that the two pins, $INTA*$ and $WR*$(on 68530) could not be connected properly. We would have to introduce delay elements.

5 Conclusion and Future Work

We present in this paper an algorithm to synthesize a combinational interface circuit between two components. The algorithm consists of two parts. In the first part, direct pin-to-pin connections are determined using interface graphs and a 0/1 ILP formulation to yield an optimal interface circuit. The resultant interface is optimal in terms of wiring area and dynamic power consumption.

For interface circuits that require logic gates they

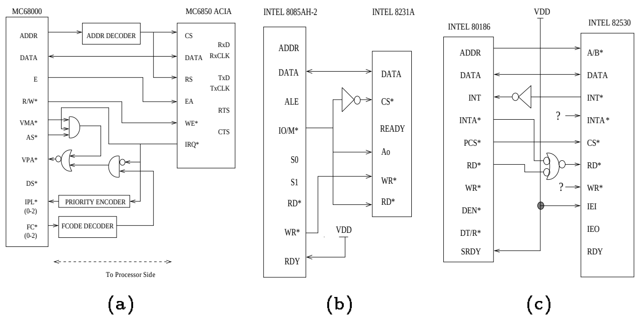

Figure 5: Interfaces generated by SYNTERFACE for (a) MC6850 ACIA & MC68000 (b) INTEL 8085AH-2 & INTEL 8231 (c) INTEL 80186 & INTEL 82530.

are synthesized using the discrete representation. This method relies on representation of a timing waveform using 0/1/u discrete values such that group of driving signals can be considered together to generate a desired waveform.

Our algorithm can be extended to consider an interface among multiple chips. However, this technique is limited to synthesize combinational glue logic circuits. Therefore, our future plan is to address the problem of interface synthesis that requires the use of sequential delay elements such as buffers and flip-flops.

References

[1] Teresa Meng, "Synchronization Design for Digital Systems", Kluwer Academic Publishers, 1991.

[2] Gaetano Borriello, "A New Interface Specification Methodology and its Application to Transducer Synthesis", PhD thesis, University of California, May 1988. Report No. UCB/CSD 88/430.

[3] P. Chou, R. Ortega and G. Borriello, "Synthesis of the Hardware/Software Interface in Microcontroller-Based Systems", In *International Conference on Computer-Aided Design,* pages 488-495, 1992.

[4] S. Narayan and D. Gajski, "Interfacing Incompatible Protocols using Interface Process Generation", In *Design Automation Conference*, pages 468-473, 1995.

[5] E. Walkup and G. Borriello, "Interface Timing Verification with Application to Synthesis", In *Design Automation Conference*, pages 106-112, 1994.

[6] P. Chou, R. Ortega and G. Borriello, "Interface Co-Synthesis Techniques for Embedded Systems", In *International Conference on Computer-Aided Design,* pages 280-287, 1995.

[7] J. Sun and R.W. Brodersen. "Design of System Interface Modules", In *International Conference on Computer-Aided Design,* pages 478-481, 1992.

[8] B. Lin and S. Vercauteren, "Synthesis of Concurrent System Interface Modules with Automatic Protocol Conversion Generation", In *International Conference on Computer-Aided Design*, pages 101-108, 1994.

[9] *Embedded Microcontrollers and Processors Vol. I & II*, Intel,1993

[10] *Peripheral Components*, Intel, 1993.

[11] *M68000 Family Reference*, Motorola, 1990.

[12] *Connectivity*, Intel, 1993