STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVI**, Number 1, 2001

RECURSIVE RULES FOR DEMULTIPLEXERS EXPANDING

ANCA VASILESCU

Abstract. This paper introduces a model for the operation of the demultiplexers based on the CCS language. The main result is a set of recursive rules for the one-dimensional expanding of demultiplexers. Starting from the CCS model for 1×2 DMUX and 1×2^2 DMUX we shall infer the CCS model for the general case of a 1×2^n DMUX.

Keywords: recursive rule, demultiplexer, CCS model

1. Introduction

An important part of the internal structure of digital computers consists of digital and logic circuits: combinational or sequential. A demultiplexer (DMUX) [2, 3, 5] is a combinational logic circuit designed for receiving a value from its single input line and transferring that value to one of its multiple output lines. In addition, a DMUX has a set of special input lines for selecting which of the output line will receive the input signal.

Intuitively, the binary combination of selection lines values represents the index of the output line selected to transfer the value received from the input line. So that, there has to be a relation between the number of output lines and the number of selection lines of a DMUX. A *DMUX of type* n has one INPUT line, n SELECTION lines and 2^n OUTPUT lines. Usually, such a DMUX is a 1×2^n DMUX.

The operation of a DMUX is based on the different kinds of activities, such as deciding if the DMUX is or not valid at one moment, loading a binary value on a particular line, interpreting the combination of selection values. All these activities can be regarded as communication between different components of the unit of DMUX.

Considering this, it is proper to model the operation of a DMUX with an algebraic language like CCS, Calculus of Communicating Systems [1, 4]. In such a model, each of the activities of the demultiplexer can be associated with an action of a CCS agent. Besides, each state of the DMUX could be a special agent in the CCS model for the demultiplexer.

2000 *Mathematics Subject Classification.* 68Q85.

1998 *CR Categories and Descriptors.* B.6.1 [**Hardware**]: Logic Design – *Design Styles*.

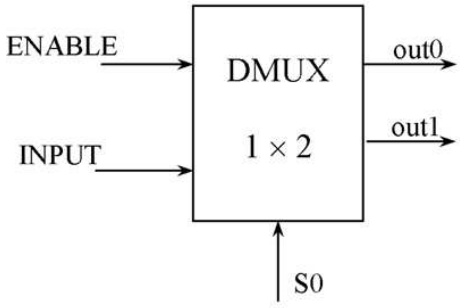

FIGURE 1. Diagrammatic representation of a 1×2 DMUX

In order to increase the power of the systems there is an efficient method to expand more identical systems to obtain a better one. In our case, we shall consider the problem of obtaining a demultiplexer of type n from demultiplexers of type $n-1$.

2. Modeling demultiplexers with CCS

In this section we shall use the algebraic language CCS (Calculus of Communicating Systems) to describe a demultiplexer (DMUX). Moreover, we shall infer a set of relations, some recursive, for expressing the method of obtaining the CCS model for a DMUX of type n, based on the model for DMUX of type $n-1$. Practically, it is useful to expand two or more demultiplexers to a demultiplexer with a large number of outputs.

2.1. **A CCS model for a 1×2 DMUX.** A 1×2 DMUX, or a DMUX of type 1, receives information from its single INPUT data line and directs it to one of the 2 OUTPUT lines, namely out0 and out1. The selection of the particular output data line is determined by a single SELECTION input line, namely S0. The bit value of this SELECTION line determines which output line receives the input value in order to transfer it to output.

For the reason of this paper, we add to this basic model an ENABLE input to control the operation of the unit. When the ENABLE value is 0, the outputs are disabled.

We propose for this unit the CCS model in Figure 3:

For this model, the possible actions are named 0, 1, INPUT, out0 and out1. Action named 0 arises when the unit accepts an input bit 0 from the ENABLE line or from the SELECTION line S0 (see Figure 1). The same for the action named 1. An action INPUT arises when the unit reads the input signal from the INPUT line. Action out0 arises when the unit transfers the input value to the selected

DMUX = DMUX'
DMUX' = 0.DMUX' + 1.D (E)
D = 0.SEL0 + 1.SEL1 (S)
SEL0 = INPUT. .DMUX'
SEL1 = INPUT. .DMUX'

FIGURE 2. CCS model for the 1×2 DMUX

D = 0.D0 + 1.D1
D0 = SEL0
D1 = SEL1

FIGURE 3. Definition of the agent D for a 1×2 DMUX

OUTPUT line out0. Action out1 arises when the unit transfers the input value to the selected OUTPUT line out1.

According to the syntax of CCS [4], the name of output ports, here $\overline{\text{out0}}$ and $\overline{\text{out1}}$, has to have the label overbared.

The agents defined in Code 1 are DMUX, DMUX', D, SEL0, SEL1.

In terms of transition representation, we can describe the operation of the demultiplexer as follows. The unit is initially in state DMUX'. The equation (E) from Figure 2 represents the role of ENABLE input. So that, while the bit value on ENABLE is 0, the unit is constantly in state DMUX'. When the value of ENABLE input is 1, the unit is changing to state D.

In Figure 2 the equation (S) means that the unit reads the bit value from the SELECTION input line. If this value is 0 the unit is changing to state SEL0, otherwise it is changing to state SEL1. The definitions of SEL0 and SEL1 mean that the signal from INPUT is copied on the suitable OUTPUT and the unit returns to the initial state DMUX'.

In order to prepare the recursive rule for generating the higher degree DMUX, it is useful to redefine the expression for the agent D from the Figure 2 as follows.

We have just introduced two new agents, D0 and D1.

2.2. **A CCS model for a 1×2^2 DMUX.** A 1×2^2 DMUX, or a DMUX of type 2, has one INPUT line, 2^2 OUTPUT lines — namely out0, out1, out2 and out3 — and 2 SELECTION lines — namely S0 and S1. In addition, we add the ENABLE input.

Like in the general model, the operation of this DMUX consists in transferring the INPUT value to one of the four OUTPUT lines, which is determined by the bit combinations of SELECTION line values.

For the 2-dimension DMUX, the CCS model could be:

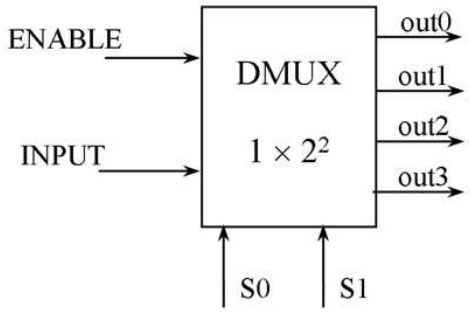

FIGURE 4. Diagrammatic representation of a 1×2^2 DMUX

DMUX = DMUX'
DMUX' = 0.DMUX' + 1.D (E)
D = 0.0.SEL0 + 0.1.SEL1 + 1.0.SEL2 + 1.1.SEL3 (S)

SEL0 = INPUT.$\overline{\text{out0}}$.DMUX'
SEL1 = INPUT.$\overline{\text{out1}}$.DMUX'
SEL2 = INPUT.$\overline{\text{out2}}$.DMUX'
SEL3 = INPUT.$\overline{\text{out3}}$.DMUX'

FIGURE 5. CCS model for the 1×2^2 DMUX

In this model, the meaning of the CCS constants is the same as in Code 1, but it is important to note that there are four bit combinations for the SELECTION input line values: 00, 01, 10, 11. For the demultiplexer from Figure 4, each of these words of 0 and 1 represents a different OUTPUT line selected to receive the INPUT value and, implicitly, a different current agent.

As in the case of 1×2 DMUX (see Figures 2 and 3) we redefine the expression for the agent D from Figure 5 as follows.

Note that for this representation we used much more constants.

2.3. **Recursive rules for DMUX expanding.** The first problem to solve now is to unify the notations used in Code 1and Code 3, namely to make the difference between the agents D from the simulation of 1×2 DMUX and 1×2^2 DMUX (see Figures 5 and 6). So that, we shall use an upper index to represent the type of current DMUX.

For a 1×2 DMUX, combining Figures 2 and 3, we have the next CCS representation:

$$\begin{array}{l} \text{D} = 0.\text{D0} + 1.\text{D1} \\ \\ \text{D0} = 0.\text{D00} + 1.\text{D01} \\ \text{D1} = 0.\text{D10} + 1.\text{D11} \\ \\ \text{D00} = \text{SEL00} \\ \text{D01} = \text{SEL01} \\ \text{D10} = \text{SEL10} \\ \text{D11} = \text{SEL11} \\ \\ \text{SEL00} = \text{SEL0} \\ \text{SEL01} = \text{SEL1} \\ \text{SEL10} = \text{SEL2} \\ \text{SEL11} = \text{SEL3} \end{array}$$

FIGURE 6. Definition of the agent D for a 1×2^2 DMUX

$$\begin{array}{ll} \text{DMUX} = \text{DMUX'} & \\ \text{DMUX'} = 0.\text{DMUX'} + 1.\text{D}^{(1)} & \text{(E)} \\ \\ \text{D}^{(1)} = 0.\text{D0}^{(0)} + 1.\text{D1}^{(0)} & \text{(S)} \\ \text{D0}^{(0)} = \text{SEL0} & \\ \text{D1}^{(0)} = \text{SEL1} & \\ \\ \text{SEL0} = \text{INPUT}.\overline{\text{out0}}.\text{DMUX'} & \\ \text{SEL1} = \text{INPUT}.\overline{\text{out1}}.\text{DMUX'} & \end{array}$$

FIGURE 7. Detailed CCS model for the 1×2 DMUX

For a 1×2^2 DMUX, combining Figures 5 and 6 we have the next CCS representation:

Note as very important that each name of the agent *memories* the sequence of previous actions already done.

We define

$$L_n = \{w \in 0, 1^\star |\ |w| = n\}$$

as the set of n-dimensional words over $\{0, 1\}$ and

$$L = \{w \in \{0,1\}^\star || w| \leq n\} = \bigcup_{k=1}^{0} L_n.$$

As a generalization, we propose the next CCS model for a 1×2^n DMUX:

$$\begin{aligned}
&\text{DMUX} = \text{DMUX'}\\
&\text{DMUX'} = 0.\text{DMUX'} + 1.\text{D}^{(2)} \quad \text{(E)}\\
&\\
&\text{D}^{(2)} = 0.\text{D0}^{(1)} + 1.\text{D1}^{(1)} \quad \text{(S)}\\
&\text{D0}^{(1)} = 0.\text{D00}^{(0)} + 1.\text{D01}^{(0)}\\
&\text{D1}^{(1)} = 0.\text{D10}^{(0)} + 1.\text{D11}^{(0)}\\
&\\
&\text{D00}^{(0)} = \text{SEL0}\\
&\text{D01}^{(0)} = \text{SEL1}\\
&\text{D10}^{(0)} = \text{SEL2}\\
&\text{D11}^{(0)} = \text{SEL3}\\
&\\
&\text{SEL0} = \text{INPUT}.\overline{\text{out0}}.\text{DMUX'}\\
&\text{SEL1} = \text{INPUT}.\overline{\text{out1}}.\text{DMUX'}\\
&\text{SEL2} = \text{INPUT}.\overline{\text{out2}}.\text{DMUX'}\\
&\text{SEL3} = \text{INPUT}.\overline{\text{out3}}.\text{DMUX'}
\end{aligned}$$

FIGURE 8. Detailed CCS model for the 1×2^2 DMUX

$$\begin{aligned}
&\text{DMUX} = \text{DMUX'}\\
&\text{DMUX'} = 0.\text{DMUX'} + 1.\text{D}^{(n)}\\
&\\
&\text{Dw}^{(k)} = 0.\text{Dw0}^{(k1)} + 1.\text{Dw1}^{(k1)}, \text{ for each } k = \overline{n,1} \text{ and } w \in L_{nk} \quad (\star)\\
&\text{Dw}^{(0)} = \text{SELi, for each } w \in L_n, \text{ where } i_{(10)} = w_{(2)} \quad (\star\star)\\
&\text{SELi} = \text{INPUT}.\overline{\text{outi}}.\text{DMUX'},\ i = \overline{0,\ 2^n - 1}
\end{aligned}$$

FIGURE 9. CCS model for a 1×2^n DMUX

In $(\star\star)$ we have to see the word w over $\{0, 1\}$ as a binary representation of the decimal value i.

The most important problem for a correct simulation of a 1×2^n DMUX is to assure that all the bit combinations are generated on the SELECTION lines. For our model this means to demonstrate that the $(\star)$ and $(\star\star)$ relations build all the elements of L_n. Hence, because every word from L_n is bijectively the representation of a decimal number from 0 to 2^n1, we can say that the model defines all the 2^n agents of the type SEL.

Theorem. The relation

$$Dw^{(k)} = 0.Dw0^{(k1)} + 1.Dw1^{(k1)}, fork = \overline{\text{n,1}} \text{ and } w \in L_{nk},\ (T)$$

generates the Dw agents for all the words w from the language L.

Proof. For $k = n$ we have $D^{(n)} = 0.D0^{(n1)} + 1.D1^{(n1)}$, because $|w| = nk = 0$ means $w = \lambda$, the empty word. This level defines the agent D and generates for this the agents $D0$ and $D1$.

The agent $D^{(n)}$ is completely defined if $D0^{(n1)}$ and $D1^{(n1)}$ are defined. That is done by applying the relation (T) for $k = n1$ and $w \in L_1 = \{0, 1\}$.

$$D0^{(n1)} = 0.D00^{(n2)} + 1.D01^{(n2)}$$

$$D1^{(n1)} = 0.D10^{(n2)} + 1.D11^{(n2)}$$

Until now, we have defined the agents D, $D0$, and $D1$ and we have generated the subset $\{\lambda, 0, 1, 00, 01, 10, 11\}$ of L, that means the words of length zero, one and two.

We prove by structural induction that for a fixed value of k, the relation (T) has already generated all the agents Dw with $w \in \{w \in \{0, 1\}^\star || w| \leq n - k + 1\}$. The value of k decreases from n to 1.

The inductive hypothesis is already verified by the previous relations written for $k = n$ and $k = n - 1$. We suppose that for a fixed value of k the relation (T) has already generated all the agents Dw with $w \in \{w \in \{0, 1\}^\star || w| \leq n - k + 1\}$ and we argue that for $k - 1$ the relation (T) generates all the agents Dw with $w \in \{w \in \{0, 1\}^\star || w| \leq n - k + 2\}$.

The induction step consists in proving that the next value of k adds all the words w which have the length with one unit greater than the words already generated. This result is obvious because the relation (T) for $k - 1$, i.e.

$$Dw^{(k-1)} = 0.Dw0^{(k2)} + 1.Dw1^{(k2)}, w \in L_{nk+1}$$

defines the agent Dw for $|w| = n - k + 1$ and generates the new agents $Dw0$ and $Dw1$. These two agents represent the binary words $w0$ and $w1$ obtained from w by adding the suffix 0 or 1. Hence, the new words $w0$ and $w1$ have the length equal with the length of w plus one.

Based on this induction we can say now that the relation (T) for $k = 1$ generates all the words $w \in \{w \in \{0, 1\}^\star || w| \leq n\} = L$. □

Corollary. The relation

$$Dw^{(0)} = SELi, \text{ for each } w \in L_n, \text{ where } i_{(10)} = w_{(2)}$$

generates 2^n agents of the type SEL.

Proof. This issue is obvious considering the next two elementary observations. Firstly, the set L_n has 2^n elements that represent all the n- dimensional bit combinations, from 0^n to 1^n. Secondly, the transformation of numbers from one base to another is a one-to-one function. Hence, the relation ($\star\star$) defines all the decimal numbers from 0 to $2^n - 1$ which become the indices for the agents SEL. □

3. Conclusions

Practically, the problem of expanding the dimension of a logic circuit like demultiplexer is a very important one because it is useful to enclose more circuits within a single integrated circuit package. Many well-known works [2, 5] refer to this subject in specific terms for logic circuits (gates, integrated circuits, blocks of diagrams and so on). Based on the support of the CCS language [4] we have inferred a set of recursive rules for describing the operation of a 1×2^n DMUX.

The relations from Figure 9 are important not only as a theoretical result, but they connect two different approaches: the diagram representation (Figures 1 and 4) and the algebraic representation (Figures 7 and 8).

Moreover, the use of languages L_n and L in the above considerations suggests working further on a (dia)grammatic representation based on the operation of suitable automata for this model. Because of the importance of the length of the words w, it is certain that this representation involves a Turing machine, perhaps a particular model of it.

References

[1] Bruns G., Distributed Systems Analysis with CCS, International Series in Computer Science, C.A.R. Hoare Series Editor, Prentice Hall, London, 1997

[2] Hennessy J.L., Patterson D.A., Computer Organization and Design — the hardware/ software interface, Morgan Kaufmann Publishers, San Francisco, CA, 1998

[3] Mano M.M., Computer System Architecture, Prentice Hall, Englewood Cliffs, NJ, 1993

[4] Milner R., Communication and Concurrency, International Series in Computer Science, C.A.R. Hoare Series Editor, Prentice Hall, London, 1989

[5] Tanenbaum A.S., Goodman J.R., Structured Computer Organization, Prentice Hall, Englewood Cliffs, NJ, 1999

Department of Computer Science, Transilvania University of Braşov
E-mail address: vasilex@info.unitbv.ro