A delay insensitive regular expression recognizer

T.S. Anantharaman

January, 1989

CMU-CS-89-109

Dept. of Computer Science
Carnegie-Mellon University
Pittsburgh, PA 15213

A delay insensitive regular expression recognizer

T.S. Anantharaman

Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

This research was partially supported by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 5167, monitored by the Space & Naval Warfare Systems Command under Contract N00039-85-C-0163. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied of the Defense Advanced Research Projects Agency or the U.S. Government.

Table of Contents

List of Figures

Abstract: Several types of synchronous Regular Expression (RE) recognizers have been proposed by several authors. This paper describes a self-timed and delay-independent RE recognizer. The problem is non- trivial because of the ε-closure operation that is implicitly performed every cycle of the recognition process. The design is based on expression-tree recognizers, and has a self-timed cycle time O(h) where h is the high of the parse tree of the RE. Since it is an expression-tree recognizer it has a compact O(N) layout algorithm, where N is the size of the RE. The design also results in an synchronous RE recognizer with the shortest worst case cycle time O(h) reported for recognizers with area O(N).

1. Introduction

Several researchers [4, 5, 3] have independently discovered synchronous regular expression (RE) recognizers based on the expression tree of the regular expression. They have the advantage of having a compact O(N) layout, where N is the length of the RE. In section 2 we first show that these schemes have a worst case cycle time of O(N), with the critical delay path being the DFS tree walk, and also explain why it is difficult to convert this circuit into a self-timed circuit in a straight forward manner. In section 3 a modified scheme is introduced in which the critical delay path is a single path from a leaf to the root and back, resulting in a worst case delay of O(h), where h is the height of the parse tree of the RE. The proof of correctness of this scheme is the key to this paper. The resulting synchronous circuit is useful in itself since it exploits more parallelism than previous designs with O(n) area complexity. In section we show how this synchronous circuit can be made self-timed and delay-independent. The resulting self-timed circuit can be used in a self-timed implementation of Path Expressions [1] and thus can be used to coordinate the behaviour of asynchronously operating systems.

2. Expression-Tree Recognizers

In Expression-Tree recognizers [4, 5, 3] a set of primitive cells is interconnected to form the recognizer. The following description is taken from Foster's Ph.D. Thesis [3]. There is a primitive cell for each of the operators (Union, Concatenation or Kleene-Star) that appear in a RE . In addition there is a separate leaf cell for every character in the regular expression. Figure 2-1 shows the leaf cell, with the clocked register denoted by the Δ box, and the cells for each of the three operators. The interconnection of these cells directly follows the parse tree of the RE. As shown in [4] compact O(N) layouts exist for this tree circuit as long as each of the cells contains a constant amount of circuitry. Each link of the tree has two signals : **ENB** which is directed towards the leaf, and **RES** which is directed towards the root. To start recognitions, all registers in the leaf cells are reset, and the **ENB** signal at the root asserted during the 1^{st} clock cycle. The k^{th} input character is asserted during the k^{th} clock cycle. The circuit will output a 1 on the **RES** signal at the root at the end of clock cycle i (just before clocking in the i^{th} input) *iff* some string in the language of the RE was input on clock cycles 1 through i-1. In particular it will be 1 at the end of the first clock cycle *iff* ε is in the language.

In general any subtree of the circuit forms a recognizer for the corresponding sub-expression of the RE and the following definitions apply to **RES** and **ENB** at the top of that subtree :

Definition 1: For any link in the tree **RES** will be 1 at the end of cycle i *iff* some string in the language of the sub-expression corresponding to the subtree below the link was input on clock cycles n through i-1, where n is any cycle during which **ENB** was 1.

For a proof that the circuit correctly produces these **RES** and **ENB** signals see [3].

In the absence of any Kleene-Star operators, the circuit delay from the output of the leaf cells (which

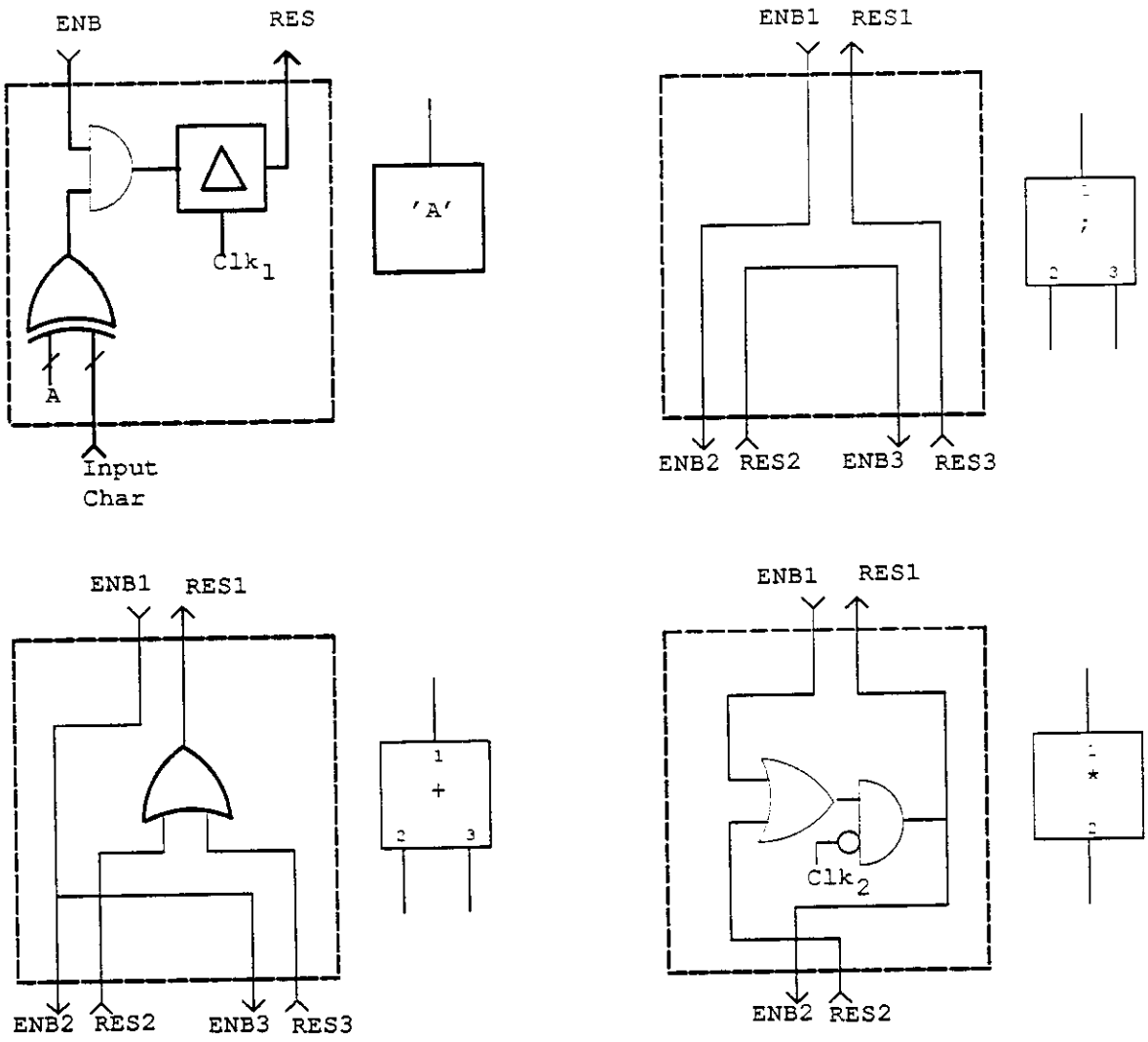


Figure 2-1: Cells of an expression-tree RE recognizer

contain the state register of the circuit) to their input, is proportional to the height of the tree. When the Kleene-Star operator is introduced it is first necessary to reset the **RES** of every Kleene-Star cell after every cycle, to prevent latch ups due to ε-cycles implicit in the circuit. This requires a second phase of the clock with duration proportional to the largest ε-cycle. Worse, since now the **ENB** of the link can depend on the corresponding **RES** as well as conversely, the total logic delay during the main clock phase can be as large as the tree walk of the expression tree i.e. O(N). This is illustrated in Figure 2-2 for the regular expression $(A^*B^*C^*D^*E^*F^*G^*H^*)^*$. In this case no matter what the input sequence is the critical path will be a DFS tree walk from the leaf cell with the last valid input through the root and back to the same leaf cell. This illustrated in the Figure assuming the last input was 'H'.

The possible simultaneous dependence of **ENB** on the corresponding **RES** as well as vise versa also means that it is not easy to make the cells self-timed using any of the conventional signalling protocols. The Kleene-Star cell cannot output a validation signal for its top **RES** until it gets a validation for the **ENB** signal since the **RES** value depends on the **ENB** signal. However the next upper cell might not be able to output the validation signal for **ENB** until the validation signal for **RES** is passed up, if there is another Kleene-Star further up in the tree. The synchronous implementations get around the problem by making

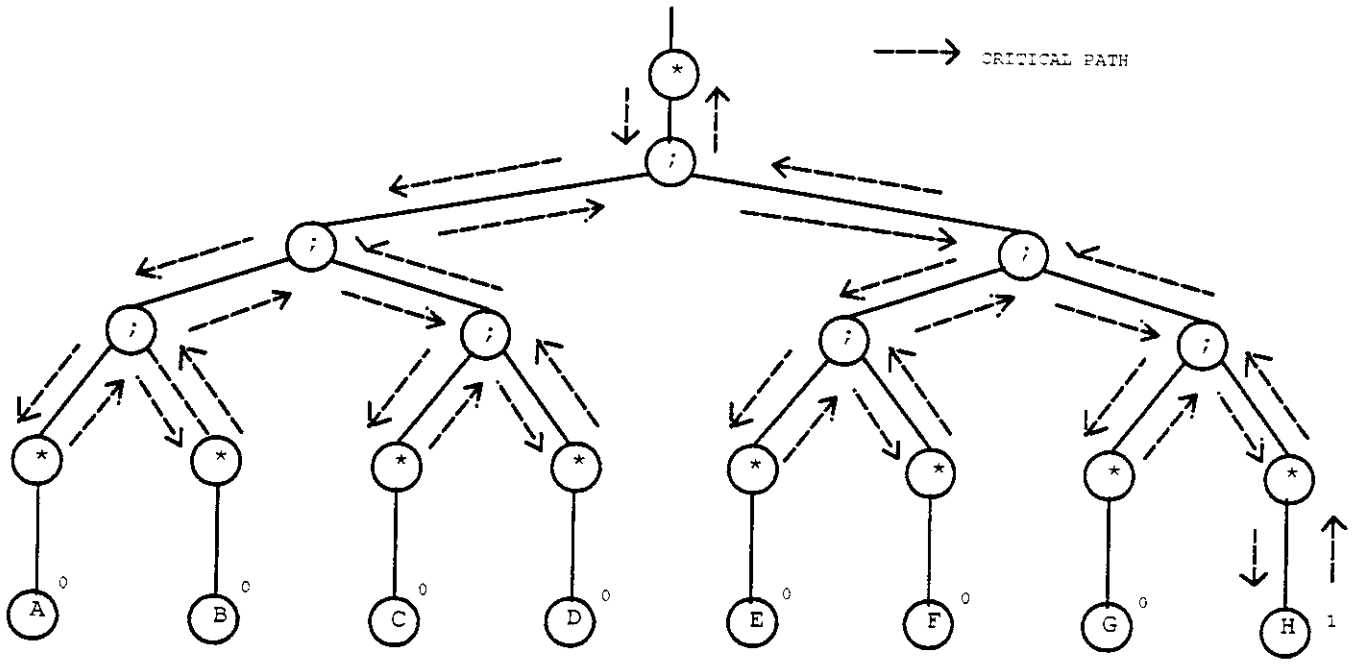


Figure 2-2: Example of a RE recognizer

the clock cycle large enough to allow for the worst case circuit delay, without worrying about where the critical path may be located. Unfortunately this can mean a very slow clock cycle. In the next section we show how the worst case critical path can be reduced without changing the tree structure of the circuit.

3. A new implementation for Expression Tree Recognizers

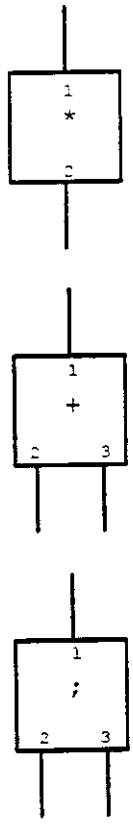
The key modification is to make the **RES** signal tri-valued, with values in the set (0,x,1). As before any subtree of the circuit forms a recognizer for the corresponding sub-expression of the RE. The meaning of **RES** and **ENB** are now defined as follows:

Definition 2: At the end of cycle i **RES** will be :

- 1 *iff* some string in the language of the sub-expression was input on clock cycles n through i-1, where n is any cycle during which **ENB** was 1 during the cycle n, and this result is independent of the value of **ENB** during cycle i (i.e. some string other than ε was recognized).
- x *iff* ε is in the language of the sub-expression and no other string in the language was input on clock cycles n through i-1 where n is any cycle during which **ENB** was 1. (i.e. the ε string was recognized conditional to **ENB** being 1 during cycle i).
- 0 otherwise.

The basic idea is to allow **RES** to be generated without making use of the **ENB** value of the same cycle. The x value encodes all possible cases in which **RES** value could be dependent on the current **ENB**, and should be interpreted as standing for "equal to **ENB**".

The leaf cell remains the same as before and its **RES** signal takes the values 0 or 1 as before. The operator cells however are different and are be described by the equations in Figure 3-1. The Max function is an arithmetic max over the values (0, x, 1) with $0 < x < 1$. It will be noticed that in each case the **RES** signal is only a function of lower (relative to the tree) level **RES** signals, whereas the **ENB**




```
RES1 = MAX ( RES2 , x)
ENB2 = ENB1 OR ( RES2 = 1)

ENB2 = ENB1
ENB3 = ENB1
RES1 = MAX ( RES2 , RES3)

RES1 = IF ( RES3 = x ) THEN RES2 ELSE RES3
ENB2 = ENB1
ENB3 = IF ( RES2 = x ) THEN ENB1 ELSE RES2
```

Figure 3-1: New Cells for an expression-tree RE recognizer

signals are functions of higher level **ENB** signals as well as lower level **RES** signals. It follows, that the all signals will stabilize after one upward propagation of **RES** signals followed by a downward propagation of **ENB** signals. The critical path of the circuit it therefore proportional in length to the tree height. It should be noted that in this case wiring delay may dominate since it is at least $O(\sqrt{N})$. There is also no longer any need for a second clock phase to reset the Kleene-Star cell.

The working of the circuit is illustrated in Figure 3-2 for the same RE used previously. On the right side of each link the **RES** signal values propagating upward are shown, and on the left side the **ENB** signals propagating downward.

It will be noted that intuitively this technique reduces worst case propagation delay by increasing the parallelism in the circuit : the circuit generates **RES** values even before it has the intermediate results (the corresponding **ENB** values) needed to determine their final values (0 or 1).

The proof that the new circuit works correctly, will be by case analysis. For each of the cells, and each possible combination of input **RES** and **ENB** signals in the new circuit, we show that the outputs produced in Figure 3-1 are the same as those produced by the original circuit in Figure 2-1, if each x value of a **RES** signal in the new circuit is interpreted as equal to the corresponding **ENB** value. The leaf cells, which store the state, remain the same. It then follows that the new circuit works correctly, given definition 2 of the **RES** and **ENB** signals in the new circuit.

Theorem 3: The new RE recognizer circuit described above (Fig 3-1) correctly generates the **RES** and **ENB** signals defined by 2, given that the original circuit (Fig 2-1, and definition 1) works

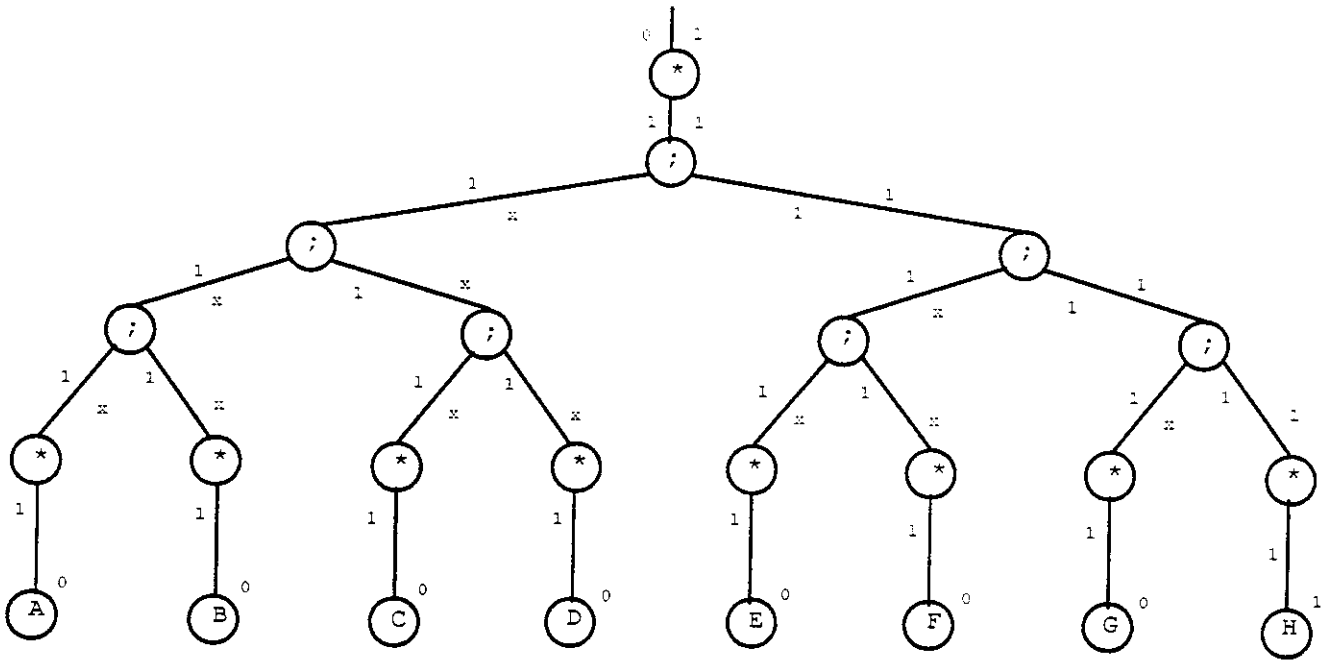


Figure 3-2: Example with the new RE recognizer

correctly.

Proof: : By case analysis (with reference to the new circuit) :

1. Kleene-Star cell ('*'):
   a. **RES_2** is 1 : The new and original circuits both set **RES_1** and **ENB_2** to 1.
   b. **RES_2** is 0 : The new circuit sets **ENB_2** to **ENB_1** and **RES_1** to x which by definition is equal to **ENB_1**. The original circuit does the same.
   c. **RES_2** is x : If **ENB_1** is 0 the new circuit sets **ENB_2** to 0 and **RES_1** to x. This means both the **RES** values are equal to 0 by definition. In this case the original circuit generates the same outputs. If **ENB_1** is 1 the new circuit sets **ENB_2** to 1 and **RES_1** to x, and hence both the **RES** values are equal to 1 by definition. Once again the original circuit generates the same outputs.
2. Union cell ('+') : **ENB_2** and **ENB_3** are set to **ENB_1** just as in the original circuit. It is only necessary to consider output **RES_1** :
   a. Neither **RES_2** nor **RES_3** are x : In this case the logic of the new and original circuits are the same (the max function in the new circuit works like the logical OR circuit in the original circuit).
   b. Both **RES_2** and **RES_3** are x : In this case **RES_1** will be x, and by definition all **RES** values are equal to **ENB_1**, i.e. all **RES** values are 0 or all values are 1. Both cases are equivalent to what the original circuit does (logical OR).
   c. One of **RES_2** and **RES_3** is x and the other is 1 : In this case **RES_1** will be set to 1 which is what the original circuit does. (If one input to a logical OR is 1 the other doesn't matter).
   d. One of **RES_2** and **RES_3** is x and the other is 0 : In this case **RES_1** will be set to x. If **ENB_1** is 0, all the **RES** signals will be equal to 0 by definition, performing the same logical OR as the original circuit. Else **RES_1** and one of the input **RES** signals will be equal to 1, again correctly performing the same logical OR function as the original circuit.

3. Concatenation cell (';') : $\mathbf{ENB}_2$ is equal to $\mathbf{ENB}_1$, always matching the logic of the original circuit. It is only necessary to consider outputs $\mathbf{ENB}_3$ and $\mathbf{RES}_1$.
    a. Neither $\mathbf{RES}_2$ nor $\mathbf{RES}_3$ are x : In this case the logic of the new and original circuits are the same ($\mathbf{ENB}_3$ is connected (set) to $\mathbf{RES}_2$ and $\mathbf{RES}_1$ is connected to $\mathbf{RES}_3$).
    b. Both $\mathbf{RES}_2$ and $\mathbf{RES}_3$ are x : In this case $\mathbf{RES}_1$ is set to x and $\mathbf{ENB}_3$ is set to $\mathbf{ENB}_1$. This means that all **ENB** and **RES** signals are equal to $\mathbf{ENB}_1$, which matches the logic of the original circuit.
    c. $\mathbf{RES}_2$ is x but $\mathbf{RES}_3$ is not : In this case $\mathbf{RES}_1$ is set to $\mathbf{RES}_3$ as in the original circuit. $\mathbf{ENB}_3$ is set to $\mathbf{ENB}_1$. However since $\mathbf{RES}_2$ is x, by definition, it is equal to $\mathbf{ENB}_2$ which is set to $\mathbf{ENB}_1$. Hence $\mathbf{ENB}_3$ is also equal to $\mathbf{RES}_2$ as in the logic of the original circuit.
    d. $\mathbf{RES}_3$ is x but $\mathbf{RES}_2$ is not : In this case $\mathbf{ENB}_3$ is set to $\mathbf{RES}_2$ as in the original circuit. $\mathbf{RES}_1$ is set to $\mathbf{RES}_2$. However since $\mathbf{RES}_3$ is x, by definition, it is equal to $\mathbf{ENB}_3$ which is set to $\mathbf{RES}_2$. Hence $\mathbf{RES}_1$ is also equal to $\mathbf{RES}_3$ as in the logic of the original circuit.

4. The self-timed Recognizer

This section describes how the new synchronous RE recognizer of the previous section can be implemented as a self-timed and delay-independent circuit. The reader is assumed to be familiar with an elementary treatment of self-timed and delay-independent circuits as in [6]. The key to the implementation is the reduction of the synchronous circuit to a Moore machine : all state information is contained in registers, the output is a combinatorial function of the state and the next state is a combinatorial function of the the previous state and the current inputs. This was done in the previous section, without destroying the tree structure of the circuit necessary for the O(N) layout.

In order to obtain a self-timed and delay-independent implementation

1. The clocked register is replaced with a register which has the following behaviour : Whenever it determines that all its inputs are *valid* (according to a suitable code described later) it remembers their value (the next state), asserts an acknowledge wire and sets all its outputs to *invalid*. It then waits until all its inputs are again invalid before resetting the acknowledge wire and setting all the outputs to the next state value.
2. The combinatorial circuit is replaced by a self-timed version that obeys Seitz's Weak conditions [6]. This means that the circuit obeys the following function constraints
    a. Some input becomes *valid* before some output becomes *valid*.
    b. All inputs become *valid* before all outputs become *valid*.
    c. Some input becomes *invalid* before some output becomes *invalid*.
    d. All inputs become *invalid* before all outpus become *invalid*.

    whenever the following domain constraints are satisfied
    a. All outputs become *valid* before some inputs become *invalid*.
    b. All outputs become *invalid* before some input becomes *valid*.

    The important property of these conditions is that when all the inputs are *invalid*, all outputs change monotonically to *invalid*, and when all the inputs become *valid* all outputs change monotonically to *valid* (and correct) values.
3. Inputs and output of the circuit follow a new self-timed signalling convention : Wait for the acknowledge wire to be deasserted, and set all new inputs to *valid* values. Wait for the

acknowledge wire to be asserted, read the *valid* output value, and reset all inputs to invalid.

It is easy to see that under these these conditions the new self-timed circuit will behave the same way as the original synchronous circuit. The clock has been replaced by a 4-cycle signalling convention (see [6] : The other possibility, 2-cycle signalling requires more complex logic to implement and is only faster if wire delay is the limiting factor.). The only care that needs to be taken in implementing the self-timed circuit is that each of the original circuit cells must remain self-contained with a constant amount of circuitry, and each logical link between the cells must use a constant number of wires. This ensures that the O(N) layout remains valid.

Implementing each of the combinatorial cells (other than the leaf cell) as seperate self-timed circuits that obey Seitz's Weak conditions ensures that they can be composed in their original fashion, since Seitz's Weak conditions are invariant under composition if no cycles are introduced [6]. The leaf cell, which contains a bit of the state register, can remain self contained if the self-timed register implementation can be seperated into bit wide component parts, with only a constant amount of communication. Finally chosing a 1 of N code for the 4-phase signalling convention ensures that only a constant number of wires are used for each original link between the cells. In the 1 of N code [6] a seperate wire (rail) is used for each possible value being conveyed. All rails reset represents the *invalid* value, and a single rail set represent one the of N possible *valid* values. The **ENB** signal would be conveyed using 2 wires, and the **RES** signal using 3 wires.

The remainder of this section describes a possible implementation of the above self-timed circuits. There is no known way of ensuring that a non-trivial circuit made up of AND/OR/NOT gates is both gate and wire-delay independent. However if a small number of more complex circuits are allowed as gate primitives, it is possible to build complex circuits that are both gate and wire-delay independent. These complex primitives may be implemented in a way that makes them gate-delay independent, but not independent of wire-delays within them. Since they only need to be designed once, care can be taken to make sure that the actual wiring in them does not violate these wire-delay dependencies. A detailed design of common primitives used in building delay-independent circuits is described in [2].

To implement the registers and transition function two complex gate primitives will be used (Figure 4-1):

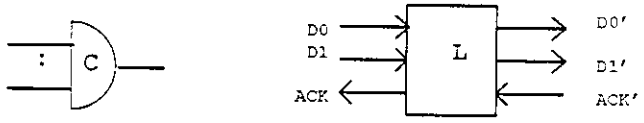


Figure 4-1: Muller 'C' element and 4-phase FIFO element

- The Muller 'C' gate : Its output goes high when all its inputs are high and goes low when all its inputs are low, and retains its previous value otherwise.
- A self-timed 4-phase FIFO element (1 bit wide) : It has two 1 bit ports, one for input and one for output. Each port follows a 4-phase request acknowledge protocol : First one of the two data wires are asserted, then the acknowledge wire is asserted, then the data wires are deasserted and finally the acknowledge wire is deasserted.

For implementations of such elements see [6, 2].

Two of the FIFO elements in series are used to implement each bit of the state register (analogous to the master-slave flip-flops present in synchronous registers). The input acknowledge of all FIFO pairs are combined with a large C-gate to generate the single acknowledge which is also internally connected to all output acknowledges of the FIFO pairs. See the left side of Fig 4-2.

The implementation of the combinatorial logic can be obtained by a mechanical procedure :

1. Obtain the minterm AND/OR expressions of each output rail as a function of input rails without using inverted literals. This is always possible with the 1 of N code since the complement of any input rail can be replaced by the OR of all the other rails of the same logical input.
2. Replace all AND gates by Muller C gates.

It is easy to verify that the resulting circuits satisfy Seitz's *Weak Conditions* for delay-independent circuits using the 4-phase protocol [6]: Each output is determined by exactly one Muller C gate, which in turn is a function of each logical input. Hence the output becomes valid only after all inputs are valid, and the output gets reset only after all inputs have been reset. It also follows immediately that each circuit output implements the correct logic whenever all logical inputs transition from the reset state to a valid state (with 1 of N rails asserted) : each minterm that is true corresponds to a C gate with all its inputs and hence output '1' and each minterm that is false corresponds to a C gate that has at least one input always '0' and hence an output of '0'. The C gate in effect implements the same logical function as the original AND gate but in a delay-independent manner. It should be noted that this method does not always produce the most compact implementation.

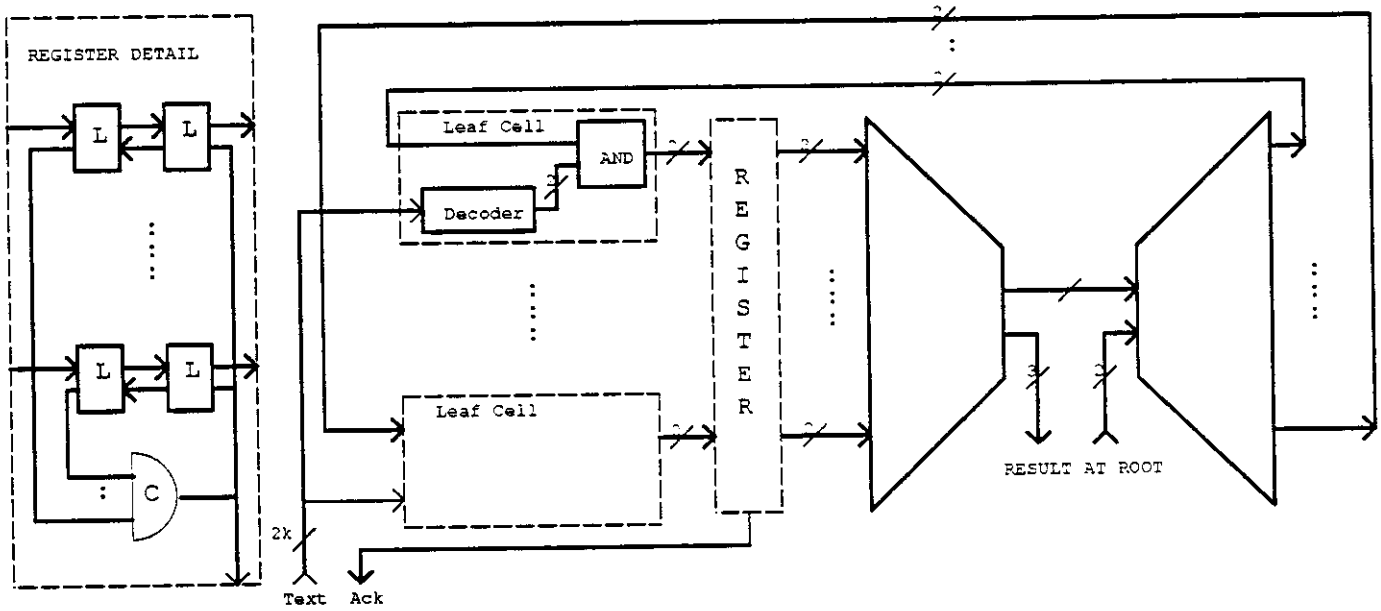


Figure 4-2: Outline of the delay-independent RE recognizer

The result of applying the above design procedure is illustrated in figure 4-2. The tree has been shown unfolded to emphasise that the circuit is of the form of a Moore machine. The character input occurs using a dual rail code using 2*k wires for the data, in effect treating each logical input bit as a seperate input implemented with 2 wires. The output occurs via the link at the root (an **RES** value).

For all logic blocks except the decoder block in the leaf cell the mechanical technique described above

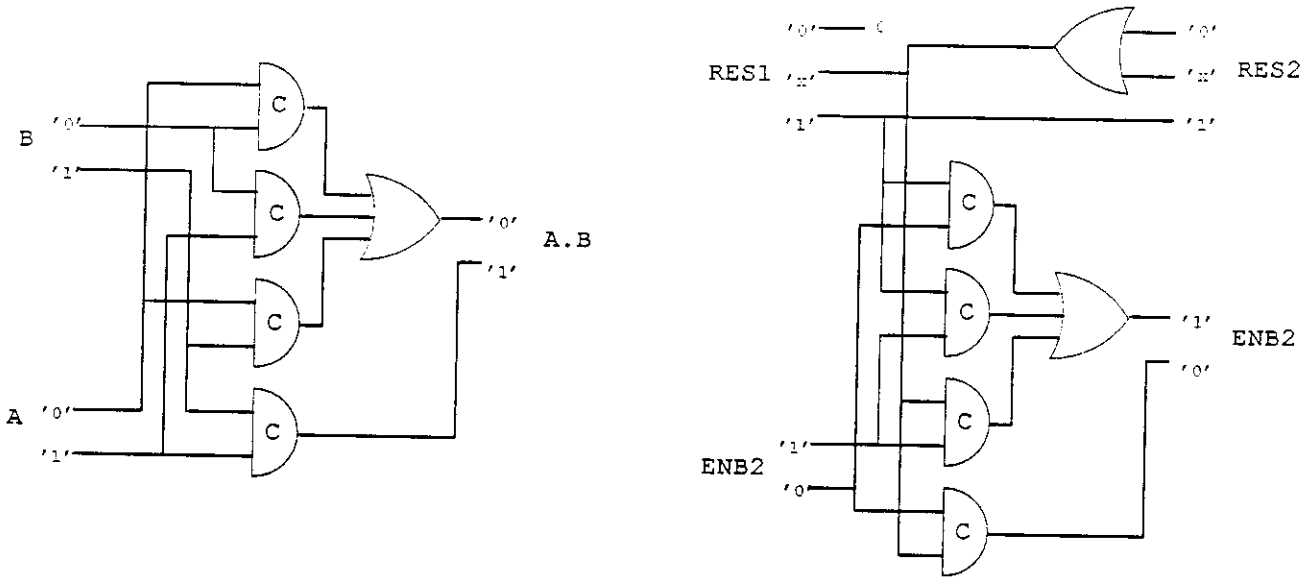


Figure 4-3: Self-timed implementation of AND gate and Kleene-Star cell

can be applied. This is illustrated in the case of the self-timed AND gate and the Kleene-Star cell in Figure 4-3. For the decoder block (in the leaf cell) this technique results in a large circuit. A more efficient implementation is given (without proof) in Figure 4-4 which shows a 4 bit decoder for the code '0110'.

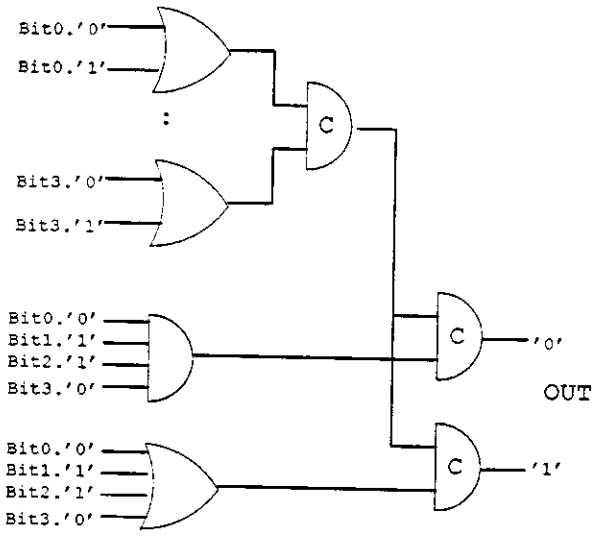


Figure 4-4: Implementation of decoder for code '0110'

5. Acknowledgements

The author wishes to thank Ed Clark, Mike Foster and Bud Mishra for many helpful discussions.

References

1. T.S.Anantharaman, E.M.Clark, M.J.Foster, B.Mishra. "Compiling path expressions into VLSI circuits". *Distributed Computing* (1986).

2. Tam-Anh Chu. Circuit Analysis of Self-Timed Elements For NMOS VLSI systems. MTLM 220, MIT, Cambridge Massachusetts, May, 1982.

3. Foster, M. J. *Specialized Silicon Compilers for Language Recognition.* Ph.D. Th., CMU, July 1984.

4. Floyd, R. W. and Ullman, J. D. "The Compilation of Regular Expressions into Integrated Circuits". *Journal of the Association for Computing Machinery 29*, 3 (July 1982), 603-622.

5. Mukhopadhyay, A. "Hardware Algorithms for Nonnumeric Computation". *IEEE Transactions on Computers C-28*, 6 (June 1979), 384-394.

6. Seitz, C. L. "System Timing". *Introduction to VLSI systems by Mead and Conway Charpter 7* (), 242-262.