An Accumulator Chip

PAUL LUCAS

Abstract—A large-scale integrated-circuit chip has been designed under the sponsorship of the Air Force Avionics Laboratory. The chip was developed as part of a logic synthesizer effort of the Design Automation group at Litton Systems, Inc., Guidance and Control Systems Division. It was designed as a type or category to be multiply used with one other gating type chip whose format is prescribed by the logic synthesizer. A major goal of this effort will be to design a complete digital system by design-automated techniques using only these two types of chips in a large-scale array.

This paper provides a logical description of the chip followed by an illustrative example of its employment in a small array.

Index Terms—Accumulator chip as a control element, accumulator chip synthesizer link, logic synthesizer, LSI chip, multiple use of the accumulator chip.

An Accumulator Chip

The development of integrated-circuit chips and the imminent approach of large-scale circuit arrays has fostered a study to find a common “all-purpose” logical chip or a small group of such chips. Research was conducted in this area under the sponsorship of the Systems Engineering Group (RTD) of Wright-Patterson Air Force Base as part of a larger effort to develop a computerized logic design system. Some of the supplementary ground rules included the following.

1) The complexity of the chip must be of a nontrivial nature, i.e., individual gates and/or flip-flops that must be custom-wired do not satisfy the intent.
2) The chip must be of a practical size consistent, for example, with imminent MOS chip technology.
3) The total number of leads emanating from the chip must not exceed a quantity consistent with the dimensions of the chip and good wiring practices.
4) The logic should employ parallel processing techniques, i.e., DDA or serial pulse processing techniques are not admissible.

Toward this end, a chip containing a 4-stage accumulator element was developed. The 4-stage chip is compatible with binary multiples. The size of the chip in the area of 700 transistors. The number of leads used is 22.

Fig. 1 is a functional logic diagram of the chip. The accumulator chip was designed with a length of four bits since this was consistent with rules 1) through 3) above. Fifteen control functions were deemed to be necessary. These functions were binary-coded into four actual lines. The control functions are listed with their codes in Table 1. An X indicates that the bit may be either 0 or 1.

In addition to power, a total of 20 control, data, and clock lines are used. There are four control and fourteen data lines. The data lines are as follows:

1) Four input lines.
2) Four output lines.
3) One carry in line.
4) One carry out line.
5) One shift-right in line.
6) One shift-left in line.
7) One multiply (LSB) line (this is the input from the “least significant bit” of the multiplier).
8) Set carry line.

A two-phase clock makes up the remaining two lines:
1) Clock 1.
2) Clock 2.

Logical Structure of the Chip

The accumulator chip consists of three 4-bit data registers and a fourth 4-bit register used to hold the control logic for the second rank of the upper register which operates at clock 2 time. As shown in Fig. 1, the data is input via the addend register. Ranks 1 and 2 are the double-rank accumulator register. A two-phase clock is used to trigger the chip. Rank 1 and the addend register are triggered by clock 1. Rank 2 is triggered by clock 2. Two clear controls are provided to the chip, one is used to clear the rank 2 register, the other clears rank 2 and the addend register. Rank 1 is cleared at the following clock 1 time by a jam transfer from rank 2 unless a restore command is employed.

Addition occurs between the addend register and rank 1. The addend register holds the addend. Rank 1 holds the augend. The resultant sum is formed in rank 2. Summing is implemented through the sum gates to the toggle input of the rank 2 flip-flops. The sum gates for the least significant bit are represented by gates 1 through 9 in Fig. 1. Their outputs control the input to rank 2. The gating is represented by the equation \((Y+\bar{C})(\bar{Y}C)\). The truth table for generating the add function is shown in Table II. Y is the addend register, X is the accumulator register, or rank 1, and C is the input carry. T is the toggle input to rank 2.

Manuscript received December 11, 1967.

The author is with Litton Systems, Inc., Guidance and Control Systems Division, Woodland Hills, Calif.
Fig. 1. Accumulated chip logic diagram.
Fig. 1. (cont'd).
TABLE I

<table>
<thead>
<tr>
<th>Control Function</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) No Op</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>2) Clear Rank 2</td>
<td>0 0 1 0</td>
</tr>
<tr>
<td>3) Transfer In and Set</td>
<td>0 0 1 0</td>
</tr>
<tr>
<td>4) Clear</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>5) Shift Left</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>6) Shift Right</td>
<td>1 0 1 0</td>
</tr>
<tr>
<td>7) Xfr and OR</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>8) Restore</td>
<td>0 1 1 1</td>
</tr>
<tr>
<td>9) Xfr and AND</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td>10) Clear Chip</td>
<td>1 0 0 1</td>
</tr>
<tr>
<td>11) Transfer In</td>
<td>1 0 1 0</td>
</tr>
<tr>
<td>12) Xfr and exclusive-OR</td>
<td>1 0 1 1</td>
</tr>
<tr>
<td>13) Complement</td>
<td>1 1 0 0</td>
</tr>
<tr>
<td>14) Add and Shift Right MULT</td>
<td>1 1 0 1</td>
</tr>
<tr>
<td>15) Transfer and Add</td>
<td>1 1 1 0</td>
</tr>
<tr>
<td>16) Add</td>
<td>1 1 1 1</td>
</tr>
</tbody>
</table>

TABLE II

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>C</th>
<th>T</th>
<th>X Next</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

THE CARRY LOGIC

The carry logic, for the least significant bit on Fig. 1, is represented by gates 10 through 13. The carry equation \((XY + XC + YC)\) is implemented in these gates. The last stage of carry output of the chip receives the direct carry input from the previous chip and all the flip-flop inputs to generate a parallel carry to the next chip with a delay of only two levels of propagation time. By this means the carry propagation for an arithmetic register can be reduced to two levels per chip plus two levels per gate in the last (most significant) chip.

THE ADD CONTROL

Addition is accomplished in the following manner. Initially, the registers are cleared, following which the augend data is transferred into the addend register at clock 1 time. At clock 2, the augend data is added to the cleared rank 2. At the following clock 1, the addend data is transferred into the addend register, and simultaneously the augend data is transferred down to rank 1. On the next clock 2 time, the addend register and rank 1 are added together and the sum placed in rank 2. If a subsequent addition is to be implemented, the next addend data would be transferred into the addend register and the rest of the procedure would be repeated. If the augend data is in rank 2, repeated additions may be implemented at a rate of one for each two-phase clock period.

IEEE TRANSACTIONS ON COMPUTERS, FEBRUARY 1969

THE SHIFT COMMAND

Shifting is implemented as follows. The shift-left input at the upper left corner of Fig. 1 is introduced into AND gates 14 and 15. The EP4 signal represents output of the previous bit. The FC4 input represents the bit to be operated on. At clock 2 time, the information of the preceding, less significant bit is transferred. At the following clock 1 time, the information is gated into the rank 1 flip-flop of the bit in question. By this means, after every two-phase clock period, the bit information is shifted one position to the left. The shift-right input shown at the center of the left side of the page is introduced into gates 16 and 17 and functions in a manner similar to the left shift. Only one set of gating is depicted as the sum input to the toggles, and one set of gating is shown as the carry output. The identical sets of gating are repeated for every one of the three remaining bits on the chip.

MISCELLANEOUS CONTROLS

The complement input (COMP) enables the toggle input to all the flip-flops in rank 2 generating a polarity switch at clock 2 time. The restore logic (RESTOR) disables clock 1 from triggering rank 1, preventing the transfer of the data from rank 2. The clock 2 trigger then transfers the old data stored in rank 1 back to rank 2 through the transfer gates enabled by the restore controls. The set control (SET) transfers data from the addend register to rank 2 of the augend register.

The gates on the right edge of Fig. 1 are the control decoding gates for the four binary-coded control lines. EI1 through EI4 are the four input data lines to the addend register. They feed directly into the set side and are inverted into the reset side of the flip-flops to accomplish the parallel input transfer. The 1 sides of flip-flops FC1 through FC4 at the top of the page represent the set side output lines. The complement or 0 side is not outputted as only single-ended input signals are required.

A Veitch-type diagram of the control signal organization is displayed in Fig. 2. Its arrangement is designed to execute certain combinations of controls within a single two-phase clock time. For example, a transfer and add is implemented on a single two-phase clock by coding \(A B C D\). Transfer and set may be coded \(ABCD\). Table I was derived by decoding the controls from Fig. 2.

MULTIPLICATION

Multiplication takes place under the control of AND gate 18. The MULTI input controls the operation of multiplication. The LSB, or least significant bit control, is the output of the least significant rank 2 flip-flop of the multiplier. When multiplying, the multiplier is placed right justified into rank 2. The multiplicand is set left justified into the addend register. The least significant bit of rank 2 is fed back into the least significant bit control position of gate 18. Whenever a logic
1 appears in the least significant bit position, the multiplicand is added to rank 1, forming the product in rank 2. If a logic 0 appears at LSB, the addition is omitted. Following each operation, a right shift is implemented. After completing as many shifts as adds as there are bits in the multiplier, the multiplication is terminated. Multiplication requires external chips to control the add shift operation, as well as to count the number of sequences required to complete the process.

Consideration was given to the merit of extending the length of the rank registers to double length to implement multiplication-type problems. This has the disadvantage of doubling the number of output data lines—that is, one for each flip-flop in rank 2. Complication is also added by increasing the length of the addend register when operating on multiplication problems with multiplicands longer than four bits. In addition, the last four bits would be wasted in all cases of add-, subtract-, or control-type operations where an operation other than multiplication or division is being implemented. Because of the general nature of this particular chip, the longer configuration was thought to be unwise.

The Accumulator Chip as a Computer Element

A computer system generally contains an input–output buffering section, the arithmetic unit, and a control unit. The buffering requires input–output transfer capability and possibly shifting. The arithmetic transfer and shifting properties may be implemented by control lines 3, 5, 6, 10, 11, and 15 of Table I. The arithmetic section of a computer requires the four basic arithmetic operations plus the ability to shift right and left and to make decisions. The add, subtract, and multiply may be accomplished by standard techniques within the chip framework. Division may be accomplished by various methods. One such method will be demonstrated later in this paper in an example using successive trial subtractions and shifts. These manipulations are all achievable within the basic functions of this device excluding the comparison or decision gate.

To prevent the accumulator chip from becoming overly cluttered with gates, a second type of chip has been designed but will not be detailed in this paper. It essentially consists of a general-purpose gating chip that is matched in fan in size to the accumulator chips output. The accumulator chip and the gate chip will represent the basic building blocks. The computer control section is the most difficult to design, particularly under the limitations of only two chip types. It may employ any or all of the accumulator chip controls, plus the decoding properties of the gate chip.

A line called the “set carry” line is included in the carry output or gate to produce a logic 1 when the line is set. This, in combination with the forcing function created by introducing the set carry into the control lines to generate a transfer and set, produces an initial data set into rank 2. This function is most useful for counter-type applications, where a prescribed number of counts may be implemented by counting from a fixed number to the full chip count rather than from zero to a particular count. As an example, take the case of counting 113 steps. Seven binary bits are required to implement this count. Since two chips are necessary (8 bits), the 113 is subtracted from $2^8 = 256$. Then, $256 - 113 = 143$. The binary number 10001111 (143) must be incremented to 255 and then reset to 143. At the full binary count (255) a carry out is generated from the most significant chip. The carry output is tied to the data input of all the bits that contain a 1 for the initial count of 143 (see Fig. 3).

It is assumed that all inputs are 0 unless set to 1. Normal counting takes place by tying the least significant bit carry in to “1” and tying the control inputs $A$, $B$, $C$, and $D$ to 1, 1, 1, and 0, respectively, which is a transfer and add function. By this means the counter will be sequentially incremented until the full count is reached, at which time 143 will be added to the counter to reset the process. The “set carry” line is used for initial system reset, which instead of adding the 143, sets it into the upper register. The count, or no count, is controlled by setting the $A$, $B$, and $C$ inputs to 1 to count or all 0's (No Op condition) when stopped.

As previously mentioned, the accumulator and gating chips were developed as part of a logic synthesis system.

The “state diagram” techniques that are the basis for the accumulator chip synthesizer link are predicated on the design of a control system from a “flow diagram.” With this flow diagram, the designer expresses a sequential specification of a given system. Included in this diagram are the operations of various dependent subsystem blocks as they operate within each interval of time. Decision logic, governing the transition from one state to another, is present as well.

By further partitioning the flow diagram, so that each step represents a physically realizable step for a digital system, the flow diagram can be converted to a state diagram. The logic for each state may then create control signals as a function of the current state. From the current state, and its associated external signals, output signals are generated to communicate with other modules or systems. Gating of these signals form decisions that direct the control counter through a progres-
Fig. 3. Counter diagram.

Fig. 4. Divider state diagram.

Fig. 5. Divider block diagram.
sive count or force it to jump to any state dependent upon the occurrences in the preceding state. The basis of these techniques is further explained in references [1] and [2].

The logic design system discussed here is an outgrowth of this method. The detailed configuration of a state diagram provides an interface to the synthesizer for the logic derivations.

The synthesizer is used to reduce and format the interaccumulator chip gating. To illustrate the operational use of the chip, the following example problem will be used.

**The Divider**

Fig. 4 is a state diagram of the divider. Fig. 5 is a block diagram of the configuration of chips necessary to implement a divider. The divider is illustrative in content and not intended to be complete. For example, the signs of the divisor and dividend have not been taken into account but are assumed to be separately considered on an absolute basis. The intention is rather to demonstrate how a system can be linked together to generate a quotient. In this example, the dividend is assumed to be eleven bits plus sign. The divisor is seven bits. The dividend is entered into rank 1 of three linked accumulator chips. The two's complement of the divisor is placed in the left justified corresponding addend register. For simplicity it has been assumed that the divisor was complemented prior to its transfer into this register. The division in this example is implemented by performing trial subtraction by addition of the two's complement of the divisor to the dividend. When the divisor is larger than the dividend, a 1 appears in the sign bit and a 0 is placed in the least significant end of the quotient register. The dividend and quotient are then left-shifted one bit. If the divisor is smaller than the dividend, a 0 appears in the sign bit and a 1 is placed in the quotient register. The divisor is then subtracted from the remainder with the difference in rank 2. If the sign bit is 1, the transfer of the difference from rank 2 to rank 1 is prevented and the old remainder is restored to rank 2 from rank 1 and left-shifted one bit. After the predetermined number of operations has taken place, the division is completed and the remainder is placed in the remainder register, which also serves as the quotient counter.

Table III is a state table of the control configuration to each register and counter during each state. The control logic is reduced from the flow diagram. For example, in state 1, the control counter must transfer and add 1 (increment) to state 2. The code for the transfer and add operation is 1110. The quotient register has no function in this state and so is assigned to 0000 No Op. The quotient counter increments, which is coded 1110. The accumulator, in state 1, is to perform an input transfer and add of the dividend which is also 1110.

### Table III: Divider State Table

<table>
<thead>
<tr>
<th>Present State</th>
<th>Control</th>
<th>Quot Reg</th>
<th>Quot Ctr</th>
<th>Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>54 (Reset Sw)</td>
<td>1 0 0 1</td>
<td>1 0 0 1</td>
<td>1 0 0 1</td>
<td>1 0 0 1</td>
</tr>
<tr>
<td>50</td>
<td>1 1 1 0</td>
<td>1 0 0 1</td>
<td>1 0 0 1</td>
<td>1 0 0 1</td>
</tr>
<tr>
<td>51</td>
<td>1 1 1 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>52</td>
<td>1 1 1 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>*53 (Acc MSB=0)</td>
<td>1 1 1 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>53 (Acc MSB=1)</td>
<td>1 1 1 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>54 (Quot Ctr=5)</td>
<td>1 1 1 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>54 (Quot Ctr&lt;5)</td>
<td>0 1 1 0</td>
<td>1 1 1 0</td>
<td>0 0 0 0</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>55 (Quot Ctr=5)</td>
<td>1 1 1 0</td>
<td>1 1 1 0</td>
<td>0 0 0 0</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>55 (Quot Ctr&lt;5)</td>
<td>1 1 1 0</td>
<td>1 1 1 0</td>
<td>0 0 0 0</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>56</td>
<td>1 1 1 0</td>
<td>0 0 0 0</td>
<td>1 1 1 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>57 (Continue Sw=1)</td>
<td>0 0 0 0</td>
<td>1 1 1 0</td>
<td>0 0 0 0</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>57 (Continue Sw=1)</td>
<td>0 0 0 0</td>
<td>1 1 1 0</td>
<td>0 0 0 0</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>58</td>
<td>1 1 1 0</td>
<td>1 0 0 0</td>
<td>0 1 0 0</td>
<td>0 1 0 0</td>
</tr>
</tbody>
</table>

* It may be noted that no apparent differences occur between the two S3 and S5 states in this table. This is only the control configuration. The data inputs, as detailed in the text, account for the difference.

### Chip Options

Prior to solving the gating of the programmable segment of the chip, a group of five options are stated as the initial conditions for each block of chips in accordance with the following rules.

IA. If the block is labeled "counter": the carry input of each bit is tied to the output of the next less significant bit with the least significant bit equal to 0.

IB. If the block is labeled "register": all carry inputs are 0.

IIA. If right shift command is used: all right shift outputs are tied to the next less significant chip inputs. The end bit is connected to 0.

IIB. If the left shift is used: all left shift outputs are tied to the next more significant chip input. The end bit is made equal to 0.

IIIA. If a block is labeled "control": the data input numbers must be included with the control input functions.

IIIB. If a block is labeled "not control": The data inputs are, bit for bit, the same as the outputs of another chip and are listed as such.

IV. Block length, i.e., 8 bits or 12 bits, etc., must be designated.

### Applying the Binary Input Format

The right-most column of Table I represents the compressed binary format of the controls depicted. When the state diagram of Fig. 4 is reduced to equation form, the controls are listed that are required to generate the sequential operations of the block. The inputs to a non-control type counter are listed prior to the listing of the state equations. The equations of the control network outputs that feed this type counter is then listed in the following manner:
The equation for a control type counter is as follows:

<table>
<thead>
<tr>
<th>first chip controls</th>
<th>second chip controls</th>
<th>external inputs to control block</th>
<th>control inputs</th>
<th>first chip data inputs</th>
<th>second chip data inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1X_2X_3X_4$</td>
<td>$X_5X_6X_7X_8$</td>
<td>$Y_1Y_2Y_3Y_4$</td>
<td>$Z_1Z_2Z_3Z_4$</td>
<td>$P_1P_2P_3P_4$</td>
<td>$P_5P_6P_7P_8$</td>
</tr>
</tbody>
</table>

The external inputs listed above are inputs to the gating block that, in turn, presents incoming conditions to the sequencing of the control block. These inputs gated with the present state data of the control block control the stepping. The external inputs are listed in a predetermined order and the binary format configuration is accordingly assigned to them. The data inputs are in the right column below the control inputs. They are the four $EIX$ inputs to the addend register shown in Fig. 4. They represent the binary configuration required to set a certain number into the control.

**The Divider Accumulator Block in the Synthesizer**

Synthesis of the accumulator portion of the block begins by initializing the options. The options would appear as follows.

I. Counter (the interchip carries must be tied together).
IIA. Not Right Shift.
IIB. Left Shift.
III. Not Control.
IV. Twelve-Bit Length.

The accumulator logic would appear as follows.

**Accumulator Logic:**

1) $S_0$ = Clear Chip  
   0000  1001
2) $S_1$ = $Xfr$ and Add  
   0001  1110
3) $S_2$ = $Xfr$  
   0010  1010
4) $S_5$ = Restore  
   0101  0111
5) $S_4$ = No Op  
   0100  0000
6) $S_3$ = Add (2’s Comp)  
   0011  1111
7) $S_6$ = No Op  
   0111  0000
8) $S_7$ = No Op  
   0111  0000
9) $S_8$ = Shift left  
   1000  0100

In 2) the $Xfr$ and Add 1 require a 1 on the $E14$ data input at $S1$ time. This can be wired in by inspection because of the simplicity of this situation. The following section, however, is more complex and will be handled differently.

**The Quotient Counter in Binary Format**

The options for the quotient counter are as follows.

I. Register (the quotient counter counts by shifting, except for 0 to 1 count, rather than incrementing; therefore, the carry is not needed).
IIA. Not Right Shift.
IIB. Left Shift (to count).
III. Not Control.
IV. Eight-Bit Length.

The quotient counter logic is as follows.

**Quotient Counter Logic:**

1. $S_0$ = Clear Chip  
   0000  1001
2. $S_1$ = $Xfr$ and Add 1  
   0001  1110
3. $S_7$ = $Xfr$ and Add (remainder)  
   0111  1110
4. $S_6$ = Clear Chip  
   0110  1001
5. $S_8$ = Shift Left  
   1000  0100
6. $S_2+S_3+S_4+S_5$ = No Op  
   0010  0000
   0011  0100
   0100  0101
   0101

In 2) the $Xfr$ and Add 1 require a 1 on the $E14$ data input at $S1$ time. This can be wired in by inspection because of the simplicity of this situation. The following section, however, is more complex and will be handled differently.

**The Control Counter in Binary Format**

The options for the control counter are as follows.

I. Counter.
IIA. Not Right Shift (the right shift command is not used in the execution of the control counter).
IIB. Left Shift.
III. Control.
IV. Four-Bit Length.

The control counter differs from the preceding block in option III, i.e., Control. Because of this, the external
inputs and variable data inputs must now be processed. The inputs to the block must be preorganized into a pattern from which the order into the logic synthesizer may be evaluated, as previously explained. For this case the inputs and their order shall be as follows.

**External Inputs:**

Order 4—Accumulator MSB—\(e\).
Order 3—Quotient Ctr (Bit 5)—\(f\).
Order 2—Continue Switch—\(g\).
Order 1—Reset Switch—\(h\).

The Control Counter logic is as follows.

**Control Counter Logic:**

1) \(SA \cdot \) Reset Switch = Clear Chip

<table>
<thead>
<tr>
<th>State Output</th>
<th>Control Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>(XXXY)</td>
<td>(1)</td>
</tr>
<tr>
<td>(XXX)</td>
<td>(1)</td>
</tr>
<tr>
<td>(XXX)</td>
<td>(1)</td>
</tr>
<tr>
<td>(XX)</td>
<td>(1)</td>
</tr>
<tr>
<td>(X)</td>
<td>(1)</td>
</tr>
<tr>
<td>()</td>
<td>(1)</td>
</tr>
</tbody>
</table>

2) \(S0\) = Increment

\(0000XX0\) \(\rightarrow 11100001\)

3) \(S1\) = Increment

\(0001XX0\) \(\rightarrow 11100001\)

4) \(S2\) = Increment

\(0010XX0\) \(\rightarrow 11100001\)

5) \(S3\) (Acc MSB=0) = Increment

\(0011XX0\) \(\rightarrow 11100001\)

6) \(S4\) (Quot Ctr=5) = \(Xfr\) and Add 2

\(0100X1X0\) \(\rightarrow 11100010\)

7) \(S5\) (Acc MSB=1) = \(Xfr\) and Add 2

\(00111XX0\) \(\rightarrow 11100010\)

8) \(S4\) (Quot Ctr<5) = Shift Left

\(0100X0X0\) \(\rightarrow 0100XXX\)

9) \(S5\) (Quot Ctr=5) = Increment

\(0101X1X0\) \(\rightarrow 11100011\)

10) \(S6\) (Quot Ctr>5) = \(Xfr\) and Add 3

\(0101X0X0\) \(\rightarrow 11100011\)

11) \(S6\) = Increment

\(0110XX0\) \(\rightarrow 11100001\)

12) \(S7\) (Continue Switch=1) = Reset

\(0111XX1\) \(\rightarrow 1001XX\)

13) \(S7\) (Continue Switch<1) = No Op

\(0111X0X0\) \(\rightarrow 0000XX\)

14) \(S8\) = Subtract 5 (\(Xfr\) and Add 11)

\(1000XX0\) \(\rightarrow 11101011\)

In the control counter logic, case 1) uses \(SA\) (any state) and reset switch to reset the control counter. In the left half of the left column of the binary listing, the \(XX\) represents all states. The \(XX1\) in the right half of the left column is obtained by observing the order and condition of the external inputs above. The condition of the accumulator, quotient counter, and continue switch is irrelevant and, therefore, is expressed as \(X\)'s. The reset switch is a 1. The condition is then \(XXX1\), which describes this state.

Case 12) prescribes \(S7\cdot\) Continue Switch = Reset. It may be noted that the "continue switch" could be replaced by the "reset switch." The continue switch was used, basically, to add an extra external input to increase the level of complication slightly to better illustrate this example.

The left-most four binary bits on the left side of 14), i.e., 1000 represent the number 8. The right-most four bits are the conditions of the external inputs. Observing the order of these inputs, from the table above, the accumulator, quotient counter, and continue switch conditions are irrelevant (\(X\)). The reset switch must be a 0 because it overrides all other inputs. The total input is then \(XXX0\).

The left-most four bits on the right side are 1110, which are the control numbers for "transfer and add" in Table 1. The right-most four bits on the right are 1011, which is the two's complement of the number 5 that must appear on the input to the addend register. This implements the transfer in of the number 11 as prescribed in the flow diagram and resulting table.

The rest of the table was similarly developed by inserting the proper digits in the selected positions. The fourteen equations are placed into the logic synthesizer. The binary equations for each of the blocks are now reduced by the computer. This is accomplished by listing all equation numbers in the left column of the state table into groups that perform a similar function for input to the right equations. For example, in the case of the quotient counter, considering the isolated case numbers 2 and 4 of the quotient counter logic table, we have 0001 and 0110. To clarify the situation, convert to Boolean letters. These are equivalent to \(a'b'c'd\) and \(a'bd'\). Considering only these two equations as a separate problem, these equations would be used to perform a 1110 and a 1001 to the individual control inputs. The control inputs are different in \(B, C,\) and \(D\) columns with only \(A\) being a 1 in both cases. This makes \(A\) equal to \(a'b'c'd+a'bd'\), which reduces to \(b'c'd+bd'\) because no states over \(S8\) are used.

The control counter logic is then reduced by the synthesizer to the following equations.

\[
A = \overline{c}d + \overline{c}d + \overline{d}f + dg + \overline{b} + h \\
B = bh + \overline{c}h + \overline{d}h \\
C = \overline{c}dh + \overline{c}dh + dfh + bh \\
D = bcdg + h.
\]
Data Inputs:

\[ EI1 = a \]
\[ EI2 = 0 \]
\[ EI3 = bc\bar{d} + cde + b\bar{d}\bar{f} + a \]
\[ EI4 = b\bar{d} + \bar{c}d + c\bar{d} + d\bar{e} \]

To verify the chip logic, a program of logic simulation was undertaken. The single chip as well as the accumulator portion of the divider was successfully computer simulated.

A second chip which may be considered a candidate for a universal gating chip has been designed for service with the accumulator chip. Its composition and format is programmed into the synthesizer so that the reduced equations represented above would have been customized to the gates.

In designing a general system with this type of chip, many functions will be easy and efficient to design.

Others, particularly in the area of control, will be cumbersome and in many cases considerably short of optimal. This is unfortunately the case with any universal chip. Compromises become necessary and the ingenuity of the designer is required. In the future, the Network Synthesizer will become increasingly useful in this area. The general application of the chip should have economic advantages, particularly in the MOS technology. In addition, the number of spares in a system with only two types of chips would be relatively small.

References


Unate Cellular Logic

AMAR MUKHOPADHYAY, MEMBER, IEEE

Abstract—A fundamental problem in "cellular logic" is to be able to "synthesize" "cellular arrays" for arbitrary switching functions that minimize a set of design parameters like size of the arrays and complexity of the individual cells and of the interconnection pattern on the cells. This paper is an attempt to develop "minimization algorithms for cellular arrays."

The particular cellular array considered in this paper is a unate cellular array which is a one- or two-dimensional arrangement of two-input one-output combinational cells, each of which is capable of being set to give any one of the fourteen unate functions of its two inputs, that is, excluding EXCLUSIVE-OR and EQUIVALENCE. The main results obtained are as follows.

1) A theorem is proved which says that the output of a unate cascade is a unate function.

2) A simple test procedure is suggested for checking whether or not a given switching function is unate cascade realizable. The procedure needs a maximum of \( n \) iterations for testing an \( n \)-variable switching function. The procedure being constructive, an algorithm is obtained for getting all the unate cascades which realize a given function.

3) An arbitrary single-output or multiple-output switching function can be realized by using one or more unate cascades arranged in a two-dimensional array. An algorithm is presented for obtaining the minimal cascade realizations of a given function or a set of functions.

Index Terms—Cellular logic, minimization algorithms for cellular arrays, synthesis of cellular arrays, unate cellular arrays.

Manuscript received December 11, 1967; revised March 22, 1968, and October 3, 1968. This work was supported by Air Force Cambridge Research Laboratory under Contract F19628-67-C-0293, and in part by National Science Foundation Grant GJ-158.

The author is with the Department of Electrical Engineering, Montana State University, Bozeman, Montana.

Introduction

A CELLULAR logic array is a one-, two-, or many-dimensional iterative arrangement of identical combinational or sequential cells with a regular interconnection pattern on the cells. With the great strides that are now being achieved in the field of large scale integration (LSI) technology, experts feel it pretty certain that most of the future generations of digital computers will employ cellular logic arrays as parts of their completely integrated, highly parallel and sophisticated subsystems. The various advantages of cellular arrays in LSI are well known: high packing density, high reliability, high manufacturing yield, flexibility in performance, ease of error diagnosis, etc. The importance of cellular logic arrays in digital technology is also reflected in the phenomenal growth of literature1 in the recent years, devoted to engineering and logical aspects of these arrays. This paper is primarily concerned with the logical aspects of cellular arrays.