Proceedings

SPIE Volume 855

IECON '87: Small Computer Applications— Hardware and Software

1987 International Conference on Industrial Electronics, Control, and Instrumentation

Phillip Gold
Chair/Editor

Sponsored by
IEEE Industrial Electronics Society

in association with
Society of Instrument and Control Engineers of Japan
and
SPIE-The International Society for Optical Engineering

3 November 1987
Cambridge, Massachusetts

IECON '87: SMALL COMPUTER APPLICATIONS-HARDWARE AND SOFTWARE

SPIE Volume 855

Contents

Sequential Control Synthesis Using State Tables

Robert M. Laurie and Paul H. Lewis

Michigan Technological University, Department of Electrical Engineering
Houghton, Michigan 49931

ABSTRACT

State transition techniques offer many advantages over relay ladder logic as programming and design tools for implementing sequential control algorithms on programmable controllers. Presented in this paper are the state transition diagram, the Petri net, and state table representations for a parallel asynchronous sequential control process. The attributes and limitations of each technique are discussed, and a state table format is presented with the capability of representing parallel asynchronous sequential processes.

1. INTRODUCTION

Sequential control is characterized by current events being dependent on past events. Control is determined not only by the value of control signals but also by their order of occurrence. Conceptually, sequential control is viewed as a discrete (noncontinuous) process. Sensor and actuator signals are discrete in nature, with a finite number of possible values. Generally, the control algorithm can be expressed as a sequence of actions and conditions, rather than algebraic differential equations as in continuous control. Sequential control is usually characterized by complex cyclical behavior, while continuous control tends to act in response to continuous input signals.

The container transfer system, shown in Figures la and lb, is an example of a sequential control application. The controller implements the programmed control algorithm by transmitting discrete actuator-signals to the plant dependent upon discrete sensor-signals received from the plant. Controller functions such as timers and counters can also be included in the control algorithm.

2. RELAY LADDER LOGIC

Initially, relay logic was the primary means of implementing sequential control. Today, programmable controllers have replaced electromechanical relays in most applications to reduce the cost of implementation and improve system reliability. Programmable controllers commonly use a graphical language called Relay Ladder Logic to program a sequential control algorithm using relay schematic symbology. Although this language resulted in quick acceptance by industry, it did not improve design methods for sequential control,

The primary disadvantage of relay ladder logic is that it is a combinational design method and does not describe the sequential progression of the operation. If one symbol is changed, it may affect any other part of the program. The current state of the system is not explicitly given, which makes designing and debugging very difficult for complex systems.

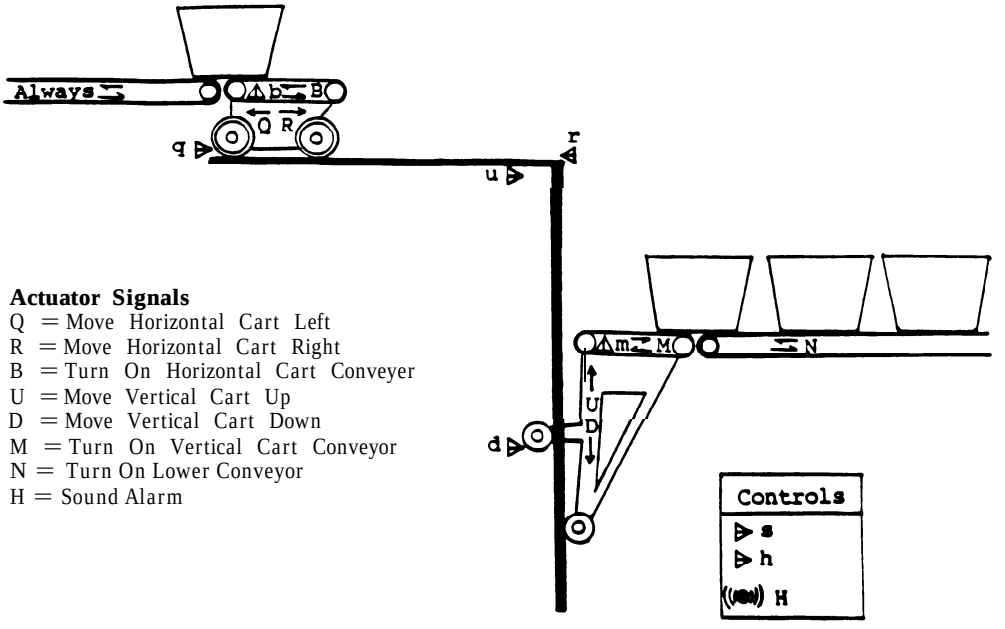
Sequential control designers are demanding much more of programmable controllers than what they were originally designed for. Applications have become more complex, and so has the need for improved design and implementation methods. Relay ladder logic has no formal way of implementing such important design techniques as algorithm partitioning, distributed control, fault detection, fault diagnostics, or redundancy [1,3].

3. SEQUENTIAL CONTROL EXAMPLE

The container transfer system illustrated in Figures la and 1 b will be used as an example to compare various synthesis techniques. It depicts a material handling system which could be used in mining operations. Sensors are represented by triangles and labeled with a lower case letter, while actuators are depicted by an upper case letter with an arrow pointing to the direction of actuated motion. The process begins with the operator engaging the start switch (s). The horizontal cart will move left (Q) until sensor (q) is activated, and the vertical cart will move down (D) until sensor (d) is activated. As shown in Figure la, the horizontal cart will then unload its container by activating its conveyor (B) until sensor (b) is de-activated. The vertical cart is loaded by activating both its conveyor (M) and the mine conveyor (N) until sensor (m) is activated. The horizontal cart will then move to the right (R) until sensor (r) is activated, and the vertical cart will move up (U) until sensor (u) is activated. The carts are now in position to transfer the container as shown in Figure lb. The conveyors of both carts are turned on (M and B) until sensor (m) is de-activated

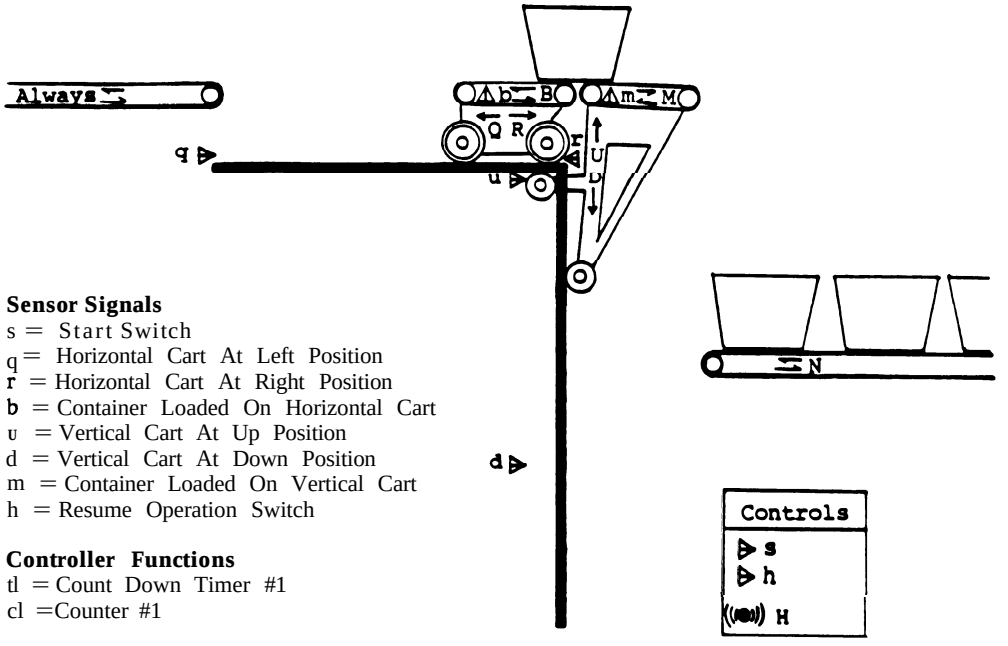
CH2484-4/87/0000/0535$01.00 © 1987 IEEE

Figure 1a: Container Transfer Between Conveyors And Carts



Actuator Signals
Q = Move Horizontal Cart Left
R = Move Horizontal Cart Right
B = Turn On Horizontal Cart Conveyer
U = Move Vertical Cart Up
D = Move Vertical Cart Down
M = Turn On Vertical Cart Conveyor
N = Turn On Lower Conveyor
H = Sound Alarm

Figure lb: Container Transfer Between Carts



Sensor Signals
s = Start Switch
q = Horizontal Cart At Left Position
r = Horizontal Cart At Right Position
b = Container Loaded On Horizontal Cart
u = Vertical Cart At Up Position
d = Vertical Cart At Down Position
m = Container Loaded On Vertical Cart
h = Resume Operation Switch

Controller Functions
tl = Count Down Timer #1
cl = Counter #1

and sensor (b) is activated. If the start button is engaged, this cycle will repeat itself with the horizontal cart moving to the left and the vertical cart moving down.

The common fault mode for this system would be the vertical cart jamming in the move up mode. Fault detection can be included in the control algorithm by having a timer send an alarm to the operator if sensor (u) is not activated within a reasonable time after the vertical cart begins moving up. Once the problem has cleared, the operator should then push a resume operation switch and the operation will continue. A counter can also be included as part of the control algorithm to count the number of containers taken from the mine.

4. STATE TRANSITION TECHNIQUES

State transition techniques have been widely used as a design and analysis method for digital computer systems and have recently been adapted for use with sequential control. Unlike relay ladder logic diagrams which use combinational methods to implement sequential control, state transition techniques show sequential behavior explicitly, All information about past behavior, which is required to determine future behavior, is defined by the currently active state. For all state transition techniques, the current state of the system is always known. Therefore, state transition techniques are a natural and highly structured synthesis method for implementing sequential control.

Discussed in this section are two graphical techniques called the state transition diagram and the Petri net. Also presented is the state table, which is a tabular representation of the sequential control algorithm. The state transition rules for each of these three techniques are uniform. Therefore, each may differ from the more classical state transition representations used for digital design.

4.1. The State Transition Diagram

The state transition diagram for the example is shown in Figure 2. Each state is represented by a circle enclosing a state number and action list. Directed lines between circles represent transitions which occur if the conditions, given in the condition list next to the directed lines, are met. When a state is first entered, the actuator specified in the action list is either turned on as indicated by "B" or turned off as indicated by "$\bar{B}$". The actuator remains on or off unless it is toggled in a succeeding state. Only one state may be active at any time.

This state transition diagram representation differs from the classical Mealy or Moore description. However, it is the 'opinion of the authors that this representation contains all necessary information without redundant symbology.

The state transition diagram is a very explicit method of describing sequential control. The desired sequence is easily designed and debugged. The graphical approach makes it easily interpreted, especially for cases with several transitional paths between states. Since only one active state exists at any time, the progression of control is easily understood. Only the conditions that are specified for leaving the active state need to be polled by the sequential controller.

A disadvantage of state transition diagrams is that algorithms requiring the crossing of transitional paths can become confusing. No formal way of representing parallel asynchronous processes exists. It is not well suited for describing global control.

4.2. The Petri Net

The Petri net is a graphical method in which the states of the system are defined by the position of tokens on places in the net. The Petri net for the example is shown in Figure 3. A place is represented by a circle with a number label, and the initial place is symbolized by two concentric circles. A right angle bracket to the right of each circle contains the action list which is executed when the place is first entered. Directed links indicate the flow of control between places, which is from top to bottom unless otherwise indicated by an arrow. Transitions through a link are represented by a short horizontal line with the associated condition list to the right. The placement of a token on a place signifies that the step is active. The double horizontal line represents an increase or decrease in the total number of tokens. This representation is similar to that used by the GRAFCET sequential programming language [7].

To understand how the Petri net operates consider the example, illustrated in Figure 3. Beginning at place 1 with a single token, actuators (M) and (B) are turned off. No further action occurs until sensor (s) is activated. At this time, the token count is increased to two, the tokens are advanced to places 2 and 6, and their action lists are executed. The two tokens progress independently through each of their place sequences until both places 5 and 9 are active. When this occurs, the token count is reduced to one and a single token advances to place 11.

The advantage of using Petri nets is that parallel asynchronous processes can be represented directly. The number of active places can be increased or decreased at the double horizontal lines to follow each of the parallel processes independently. As shown in Figures 2 and 3, the placement of tokens on the eleven different places of the Petri net can result in a simplified algorithm over the twenty-two states

Figure 2: State Transition Diagram For Container Transfer System

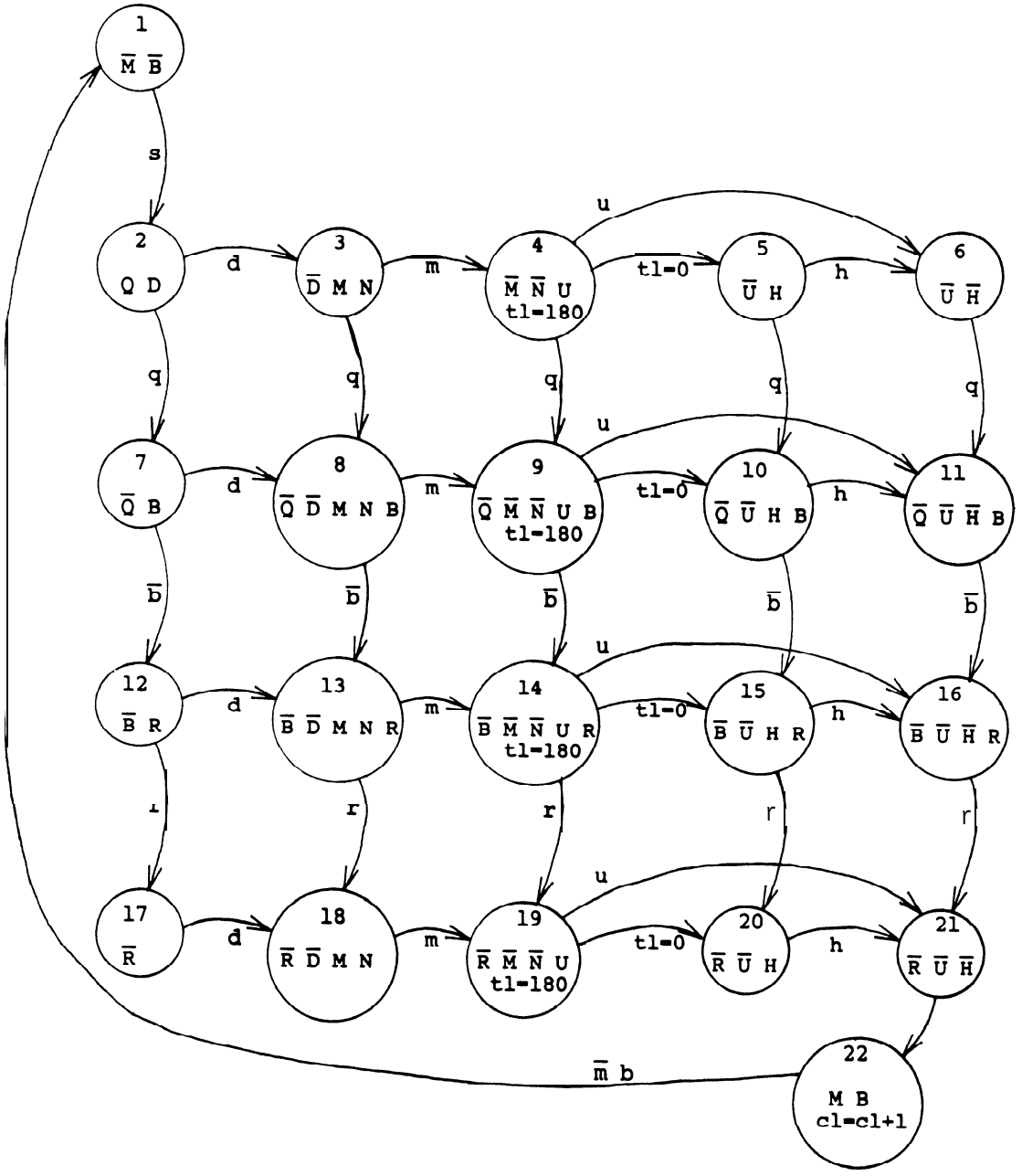
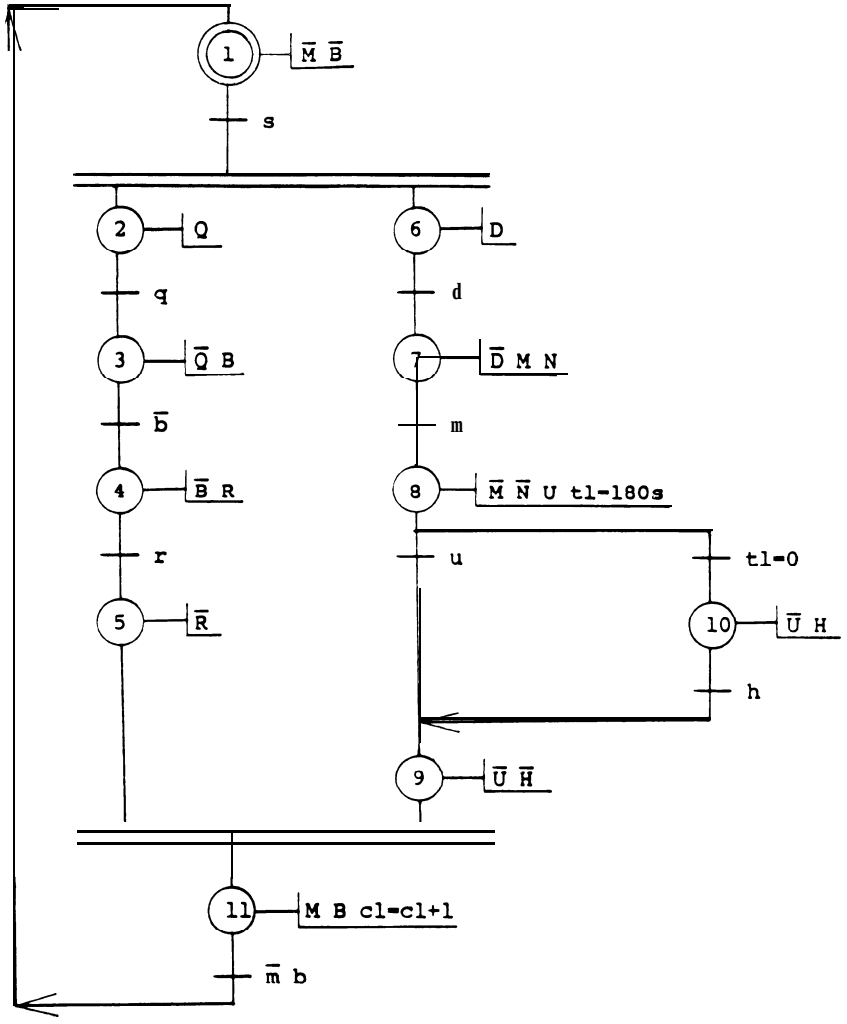


Figure 3: Petri Net For Container Transfer System



required of the state transition diagram. This is because the state transition diagram must have a separate state for all possible combinations of token placements on the Petri net. This reduction in complexity makes the Petri net an excellent choice for the design of global controller algorithms.

A disadvantage of Petri nets is that nets become confusing when several transitional paths exist between places. Petri nets have no advantage over state transition diagrams for local control (single active place) applications.

4.3. The State Table

The state table is a tabular representation of the control algorithm. Its use has been suggested for complex digital logic designs [5]. The state table representation for the example is shown in Figure 4. Four columns exist in the state table: the state label, which includes the state number; the action list, which is executed when the state is entered; the condition list, which defines conditions for a state transition to occur; and the new state, which is entered if a state transition occurs.

Figure 4: State Transition Table For Container Transfer System

STATE LABEL	ACTIONS LIST	CONDITIONS LIST	NEW STATE
1: Ready	$\overline{M}\ \overline{B}$	s	2, 6
2: Horz. Cart Left	Q	q	3
3: Unload Horz. Cart	$\overline{Q}B$	$\overline{b}$	4
4: Horz. Cart Right	$\overline{B}R$	r	5
5: Wait for Vert. Cart	$\overline{R}$	State 9	11
6: Vert. Cart Down	D	d	7
7: Load Vert. Cart	$\overline{D}MN$	m	8
8: Vert. Cart Up	$\overline{M}\overline{N}U$ tl=180s	u	9
		tl=O	10
9: Wait for Horz. Cart	$\overline{U}\overline{H}$		
10: Sound Alarm	$\overline{U}H$	h	9
11: Transfer Container	M B cl=cl+l	$\overline{m}b$	1

Global control can be implemented on a state table if the capability exists for increasing or decreasing the number of active states. An increase in the number of active states can be represented simply by putting multiple state numbers in the new state column as shown in Figure 4, state number 1. A decrease in the number of active state is represented in Figure 4 beginning with the state 5 row. One state is chosen as the decision state (State 5) and all other parallel terminating states are called wait states (State 9). Wait states mark the end of a parallel process and have no condition list. The decision state contains wait state numbers in the condition list, and when all wait states are activated the transition to the new state occurs. The new state is called the reduction state (State 11), because the total number of active states is reduced by the number of wait states specified in the decision state's condition list.

Multiple transitional paths are represented by multiple rows for the condition list and new state columns. This is illustrated in Figure 4 by the state 8 row. The state table is always readable, no matter how many transitional paths exist, since all transitional paths are easily represented by condition-list/new-state rows.

The state table of Figure 4 represents the same sequential control algorithm as described by the Petri net of Figure 3. This is possible because the state transition rules for the state table are the same as the token firing rules for the Petri net. This may allow Petri net verification and reduction techniques to be applied to state tables [4,6]. The state transition diagram control algorithm could also be used to generate the state table, since the state transition

rules are identical. The resulting table would have twenty-two rows of states and would not use the multiple active state functions described in the preceding paragraph.

4.4. Selecting The Optimal Technique

All three state transition techniques can be used to synthesize sequential control algorithms; however, the best choice depends on the requirements of a particular application. For global control, either the Petri net or state table should be chosen. For local control, the state transition diagram is often less confusing than the Petri net. Graphical methods are preferred for simple applications, since the transfer of control is more obvious. However, as applications become more complex, graphical methods become cumbersome and state tables are the preferred approach. Consider the system with sixty states and thirty transitional paths between each of the states. For this case, either graphical approach would be confusing and the state table should be used.

Several programming languages have been proposed for state transition techniques [1,3,7]. A state table based programming language would maximize capabilities and versatility. The state transition diagram or Petri net can best be utilized in the conceptualization phase. As the control algorithm becomes more complex, graphical languages tend to grow in two dimensions while a tabular language will grow in only one dimension. Therefore, a tabular language would be more readable when using conventional CRT or printer output to list the control algorithm. Either graphical method can easily be converted to a state table; however, to base a programming language on graphical methods would limit its capabilities.

A state table language can reside in either a translator or compiler environment. The authors have constructed a compiler, using a two pass assembler [1]. The compiler converts a very readable state table description into application code, which is executable by the state machine processor [1, 2]. The resulting application code is very compact.

5. CONCLUSION

State transition techniques are very useful in synthesizing sequential control algorithms. They are well adapted for sequential control purposes and show the sequential behavior of the system explicitly.

The state table is an excellent choice as a sequential programming language. It is the opinion of the authors that a table formatted language will produce the most structured and readable code.

6. REFERENCES

[1] Laurie, Robert M., "An Optimized Sequential Controller For Implementing State Transition Techniques", M. S. Thesis, Department of Electrical Engineering, Michigan Technological University, Houghton, MI., February 1986.

[2] Laurie, Robert M. and Paul H. Lewis, "A State Machine Architecture Optimized For Sequential Control", IEEE Industrial Electronics Society Conference, Cambridge, MA., November 1987.

[3] Hopkins, Albert L., Jr., "Software Issues in Redundant Squential Control", IEEE Transaction on Industrial Electronics, Vol IE-29, No 4, pp 273-278, November 1982.

[4] Johnsonbaugh, Richard and Tadao Murata, "Petri Nets and Marked Graphs - Mathematical Models of Concurrent Computations", The American Mathematical Monthly, Vol 89, No 8, pp 552-556, October 1982.

[5] McPhillips, A. Scott, "Designing Logic with Software", Proceedings IEEE 1978 MICRO-DELCON, The Delaware Bay Microcomputer Conference, pp 7-11, March 1978.

[6] Esteban, P., R. Valette, and M. Couvoisier, "Simplified Algorithms For Petri Net Analysis", IEEE Industrial Electronics Society Conference, Milwaukee, WI., October 1986.

[7] Lloyd, Mike, "Graphical Function Chart Programming for Programmable Controllers", Control Engineering, Vol 32, No 10, pp 73-76, October 1985.

A State Machine Architecture Optimized For Sequential Control

Robert M. Laurie and Paul H. Lewis

Michigan Technological University, Department of Electrical Engineering
Houghton, Michigan 49931

ABSTRACT

State transition techniques have shown the greatest potential as a synthesis technique for sequential control; however, to maximize the efficiency of these techniques, a new programmable controller software architecture must be developed. The sequential controller design described in this paper utilizes a state machine and state table architecture. The sequential controller is capable of implementing control algorithms based on either Petri net, state transition diagram, or functional state table descriptions.

1. INTRODUCTION

The majority of sequential control applications currently use a programmable controller (PC) as the sequential controller. The PC uses a programming language called relay ladder logic to activate discrete outputs based on the current values of discrete inputs. The PC operates as a program scanning translator, in which the entire program is rapidly scanned in an effort to simultaneously execute the combinational logic statements. This architecture has inherently slow response times, which will increase linearly with program size. Relay ladder logic is not well suited as a sequential control programming language. It is a combinational method and does not describe the sequential progression of the operation [1,2,3,5].

State transition techniques have shown the greatest potential as a synthesis technique for sequential control. They have been used for design and analysis of digital computer systems and have recently been adapted to industrial sequential control. To reap the real benefits of state transition techniques, a new type of architecture must be developed for programmable controllers. Response times are expected to be several orders of magnitude faster with the state machine and state table architecture presented in this paper.

2. FUNCTIONAL REQUIREMENTS

The sequential controller rnust execute the basic control functions common among programmable controllers. These functions include discrete I/O checking and manipulation, counter operations, timer alarms, and user-data register manipulation.

All functions required by state transition techniques must be executable on the sequential controller. Transferring from a state transition technique representation of the control algorithm to the required application data must be easily accomplished. The coded application data should be compact and require minimal processing time.

A terminal display of discrete I/O and currently active states is necessary for debugging the control algorithm and monitoring the system.

3. CONTROLLER ARCHITECTURE

The state machine is the processor which operates on data contained in the state table to implement a particular sequential control algorithm. Block diagrams of both the state machine and state table architectures presented in this paper are shown in Figure 1.

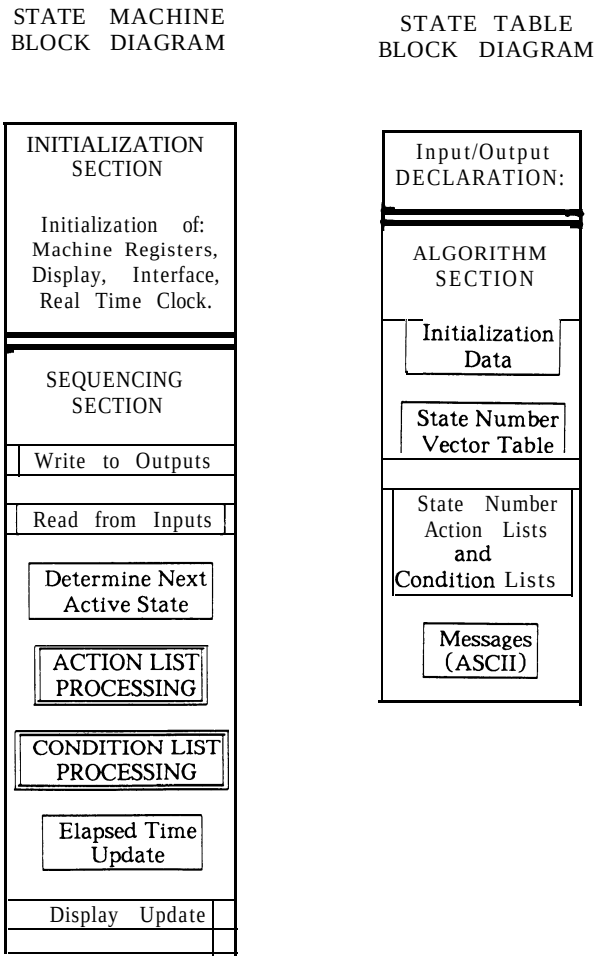
3.1. The State Machine

The state machine consists of four sections: the initialization section, the sequencing section, the action list processing section, and the condition list processing section. The initialization section performs all operations required to initialize the state machine for a particular control application. The sequencing section performs the sequencing required to implement the control algorithm. The action list and condition list processing sections are called as subroutines within the sequencing section. When a state is first activated, action list processing occurs; otherwise, condition list processing occurs.

Global control of parallel asynchronous systems can be implemented by the state machine if the capability exists for executing multiple active states. The state machine described in this paper provides this capability. The number of active states can be increased or decreased, just as the token count can be changed for the Petri net. Therefore, the state machine architecture described in this paper is based on the token player concepts of the Petri net [1, 2, 4, 5]

CH2484-4/87/0000/0477$01.00

Figure 1: Sequential Controller Architecture



State machine registers are RAM locations required for operation of the state machine. These RAM locations are used to represent pointers, active-state registers, flags, input and output data buffers, and user-data registers. State machine registers are the only RAM locations required for a sequential controller implementation, because the state machine program and state table data are accessed in read-only mode.

3.2. The State Table

The state table contains machine coded data defining a sequential control algorithm. The name state table has also been used in the literature to refer to a table formatted algorithm description. For the purposes of this paper, the table formatted algorithm description will be referred to as the the functional state table. The state table contains the I/O declaration section and the algorithm section. The I/O declaration section assigns a label to all possible states of inputs and outputs. The algorithm section contains the data describing the sequential control application. The algorithm section may be generated from a functional state table, a Petri net, or a state transition diagram representation of the control algorithm.

A functional state table format is the best choice for generating the algorithm section. Graphical methods may be used to design a sequential control algorithm; however, to base an input-data format on a graphical format would limit the sequential controller's capabilities [2].

4. STATE MACHINE OPERATION

The state machine begins execution by initializing registers, the display, the real time clock, and the I/O interface, as described by initialization data in the state table.

Sequential control begins as the state machine enters the sequencing section. A flow chart of the sequencing operation is shown in Figure 2. Outputs are activated as specified by the output data buffer, and input conditions are stored in the input data buffer. The next active state is determined by incrementing the pointer to the next active-state register. If a transition to this state has just occurred, the state's action list is processed; otherwise, the state's condition list is processed. The location of the state's action list or condition list is determined from the state number vector table as shown in Figure 1. Action list processing can manipulate outputs, increment counters, set timer alarms, change the number of active states, or manipulate user-data registers. These actions will occur only when the state is first entered. After an action list has been executed, transition conditions are checked every time this active state is called by the state machine. These transition conditions may be based on inputs, timer alarms, counters, the value of user-data registers, or the state number of other active states. After action or condition list processing has occurred for an active state, the elapsed time and display are updated. After all active states have been executed, inputs are read, outputs are activated, and the active-state pointer is reset.

The described state machine uses polling to check inputs. Polling was chosen over an interrupt-driven structure, because it minimizes external hardware and eliminates potential race conditions.

5. DATA FORMATS

Specific data formats have been chosen for the state machine and state table to minimize memory

requirements and maximize processing speeds. The data formats presented are for an 8-bit data word.

5.1. The State Machine

The state machine uses specific data formats for state machine registers. The active-state registers contain the currently active-state numbers as specified by the format shown in Table 1. Bits 7 through 1 specify the state number. Bit "0" identifies whether action or condition list processing should occur. State number "0" is reserved as an end of active states marker. Using this format, a maximum of 127 states could be specified for a state machine designed for an 8 bit word length. If the word length of the state machine was increased to 16 bits, up to 32,767 states could be specified. Flags are byte accessible so that processing time will be minimized. Flags are only checked for zero or non-zero values. User-data registers are byte accessible.

Figure 2: Sequencing Section Flow Chart

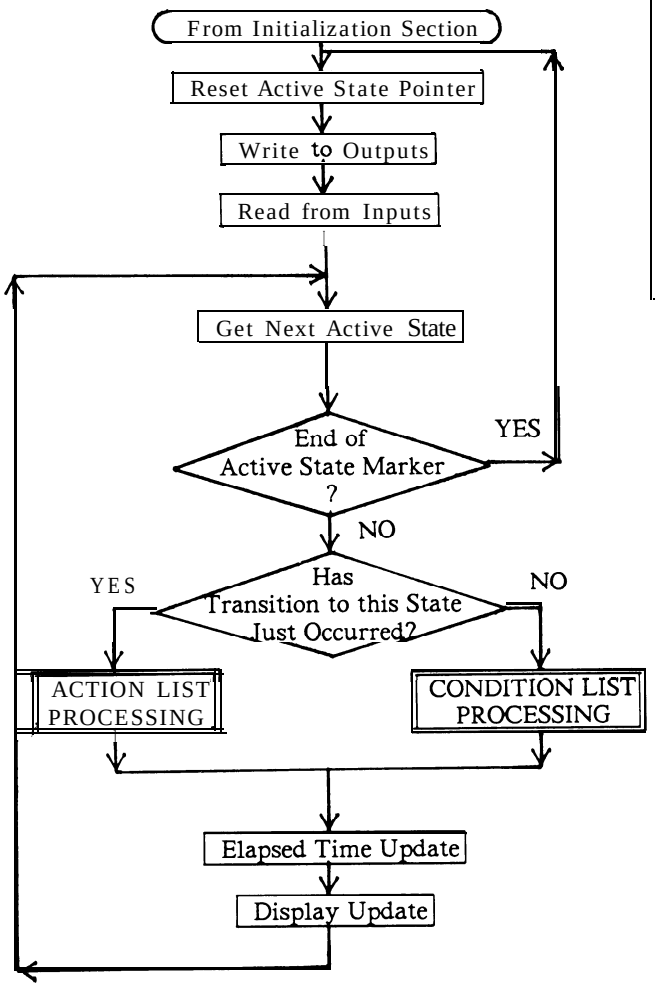


Table 1: Data Formats for Controller

ACTIVE STATE REGISTER FORMAT

```
Bit # 76543210
     ::::::::
     00000000 = End of Active States Marker
     xxxxxxxl = Action List Processing
     xxxxxxxo = Condition List Processing

            xxxxxxx = State Number (1 - 127)
```

ACTION LIST COMMAND FORMAT

```
Bit # 76543210
     ::::::::
     00rrrrrr = Force Off Discrete Output
     01rrrrrr = Force On Discrete Output
     laaaaaaa = Action List Function

            rrrrrr = Output Bit Number
            aaaaaaa = Action List Function
                      Operation Code
```

CONDITION LIST COMMAND FORMAT

```
Bit # 76543210
     ::::::::
     oouuuuuu = Check for Off Discrete Input
     01uuuuuu = Check for On Discrete Input
     lccccccc = Condition List Function

          uuuuuu = Input Bit Number
          ccccccc = Condition List Function
                    Operation Code
```

5.2. The State Table

The most common action or condition list command will be discrete I/O manipulation and checking. Therefore, a single byte instruction will be used to manipulate a discrete output for action list processing and check a discrete input for condition list processing. As shown in Table 1, the format for discrete I/O commands always begins with a "0" for bit 7. If the discrete I/O is either forced on or checked for on, bit 6 will equal "1". If the discrete I/O is either forced off or checked for off, bit 6 will equal "0". Bits 5 through 0 represent the discrete I/O bit number. Up to 64 discrete I/O points can be accessed using this format for an 8-bit data word. If a 16-bit data word is used, up to 16,384 discrete I/O points are accessible.

Action list and condition list functions are represented by a single byte command with bit 7 equal to a "1", as shown in Table 1. Each list function has a mnemonic label and operation code, as illustrated in Tables 2 and 3. Operands follow the list function commands, just as operands follow assembly language commands. Up to 128 different

action list commands and 128 condition list commands may be supported by a state machine with an 8-bit data word.

Basic action list functions are described in Table 2. User-data register functions are included for arithmetic operations and register transfer operations. A timer alarm function is provided to set a real-time alarm. The EXPAC function is used to increase the total number of active states for parallel asynchronous control algorithms. HLTSQ is used to halt sequencing and send a message to the operator. A no-operation function is provided to simply take up byte space during the debugging process, and the ACEND command marks the end of action list processing.

Basic condition list functions are described in Table 3. Functions are provided for comparing user-data registers and to check if a data value has been reached. These commands are especially useful for counters. A timer alarm function is included to test if a previously set real-time alarm is now in the alarm condition. The BLKOR command marks the end of a set of conditions which are logically OR'ed with the next set of conditions. TRANS marks the end of a set of conditions and specifies which state control is transferred to, if these conditions are met. The command CONAC is used to decrease the total number of active states if the specified state numbers are now active. The CNEND command marks the end of the condition list.

Table 2: Action List Functions

MNEMONIC SYMBOL	OP. CODE	DESCRIPTION
LOADR	80H	Load a user-register with immediate data.
MOVER	81H	Move data between user-registers.
INCRG	82H	Increment data in user-register.
DECRG	83H	Decrement data in user-register.
ADDIM	84H	Add immediate data to user-register.
SUBIM	85H	Subtract immediate data from user-register.
ADDRG	86H	Add two register values together.
SUBRG	87H	Subtract first register value from second.
TIMES	88H	Set real time alarm.
EXPAC	89H	Expand the number of active states.
HLTSQ	8AH	Halt sequencing and send message.
ACNOP	8BH	No operation.
ACEND	8CH	Marks the end of action list.

6. CONTROLLER IMPLEMENTATION

A sequential controller was constructed in the laboratory using the architecture described in this paper [1]. It was implemented on an MC6809 based microcomputer system. All software was written in assembly language to generate the most compact and efficient code. An assembler was used to convert a functional state table representation of the control algorithm into machine code for the state table.

Table 3: Condition List Functions

MNEMONIC SYMBOL	OP. CODE	DESCRIPTION
CMPGE	80H	Compares user-registers for greater or equal.
CMPGT	81H	Compares user-registers for greater than.
CMPEQ	82H	Compares user-registers for equal.
CMPNE	83H	Compares user-registers for not equal.
CIMGE	84H	Immed. data to register compare greater or equal.
CIMGT	85H	Immed. data to register compare greater.
CIMEQ	86H	Immed. data to register compare equal.
CIMNE	87H	Immed. data to register compare not equal.
CIMLE	88H	Immed. data to register compare less or equal.
CIMLT	89H	Immed. data to register compare less than.
TIMET	8AH	Real time alarm activated?
BLKOR	8BH	Logically or two conditional blocks.
TRANS	8CH	If conditions true, transfer to new state.
CONAC	8DH	Contract the total number of active states.
CNNOP	8EH	No operation.
CNEND	8FH	Marks the end of condition list.

The state machine program was compact and efficient. The program required less than 1700 bytes of memory. Response time was minimized, because the state machine checked only the conditions which would cause a state transition to occur.

State tables were easily generated for several sequential control systems. For all systems, the state table code was extremely compact. Two parallel asynchronous system examples were found in references [2] and [5]. The state table for the container transfer system of reference [2] required only 148 bytes of memory, and the state table for the three station automated drilling system of reference [5] required 215 bytes of memory.

7. CONCLUSION

A sequential controller was designed using a state machine and state table architecture. The sequential controller is optimized in the sense that the control algorithm machine code is exceedingly compact and the response time is minimal. The control algorithm is entered using a functional state table format, so any state transition technique could be used to generate the control algorithm. Testing with several sequential control systems was successful, and the application state tables were obtained quickly and without difficulty.

6. REFERENCES

[l] Laurie, Robert M., 'An Optimized Sequential Controller For Implementing State Transition Techniques", M. S. Thesis, Department of Electrical Engineering, Michigan Technological University, Houghton, MI., February 1986.

[2] Laurie, Robert M. and Paul H. Lewis, "Sequential Control Synthesis Using State Tables", IEEE Industrial Electronics Society Conference, Cambridge, MA., November 1987.

[3] Hopkins, Albert L., Jr., "Software Issues in Redundant' Sequential Control, IEEE Transaction on Industrial Electronics, Vol IE-29, No 4, pp 273-278, November 1982.

[4] Johnsonbaugh, Richard and Tadao Murata, "Petri Nets and Marked Graphs - Mathematical Models of Concurrent Computations", The American Mathematical Monthly, Vol 89, No 8, pp 552-556, October 1982.

[5] Lloyd, Mike, 'Graphical Function Chart Programming for Programmable Controllers", Control Engineering, Vol 32, No 10, pp 73-76, October 1985.