State Transition Table Based Control Software Engineering For Micros

[1]Hyacinth C. Inyiama [2]Ifeyinwa Obiora-Dimson [3]Christiana C. Okezie

[1,2,3] *Department of Electronic and Computer Engineering Nnamdi Azikiwe University, Awka. Anambra State, Nigeria.*

Abstract

A State Transition Table based software engineering for micros has been developed in this paper. The method begins with a control specification and a corresponding Algorithmic State Machine (ASM) chart. A State Transition Table (STT) corresponding to the ASM chart is then developed such that there are as many rows in the STT as there are link paths in the ASM chart. The STT is organized such that the state code comes before the qualifiers. This segments the entire STT into various states in the state machine and facilitates the software development. Structured software is then written to match the STT such that there is one program statement per link path. The advantage of this technique is that it begins with an initial disciplined hardware approach up to the STT stage and then generates the structured software that match the rows of the STT on one-on-one basis. This makes debugging easy. This approach is universal and can be applied to any process control system that can be represented as an ASM chart.

Keywords: *Algorithmic State Machine (ASM) chart; Microprocessor/microcomputer; Software engineering; State Transition Table (STT); Link path; Digital system design.*

1. Introduction

Microprocessors and microcontrollers are very flexible for use in complex logic system development because of the extra flexibility provided by programming the software for specific design functions. The features of a microcontroller/microprocessor based system that recommended it to an electronic control engineer are as follows:

1. A microprocessor/microcontroller based system is compact, with low component count. It has few sources of error and a high reliability figure. This is in sharp contrast to the case of a hardware-only system which would be quite complex especially when a complex control algorithm is being implemented, leading to many components, many potential error sources, inflexibility due to rigid interconnections, and low reliability as buttressed by [1] .
2. The use of software in microprocessor/microcontroller based systems not only ensures that pertinent system requirements of each application are met but also makes possible the inclusion of other useful features that can ensure the usefulness of a project.
3. Troubleshooting (i.e. fault finding) in a microprocessor/microcontroller based system involves less complicated re-wiring effort compared to a hardware-only system because quite often, it is only the software that needs be modified in order to correct an error. Even when hardware rewiring is necessary, the components involved are well structured and easy to work with.
4. Because the microprocessor/microcontroller is both versatile and flexible, it can be used in a wide range of dedicated applications while leaving its exact function in each case to be established by the logic designer (or electronic control engineer). This tailoring to a particular application is achieved through programming together with the use of an appropriate digitally controlled interface between the microprocessor/microcontroller and each dedicated application [7]

The technique for the provision of such interfaces have been standardized and typically involve input-output-mapped input/output, memory-mapped input/output, Direct Memory Access (DMA) and the use of interrupts [8].

5. When the dedicated applications are needed to be in operation one at a time, a single microprocessor/microcontroller system can be used to implement two or more of such applications leading to more considerable cost savings. For example, all the control

system laboratory sessions that may involve up to fifteen (15) different real time control experiments done one per week for the semester can all be implemented using just one microprocessor/microcontroller based system as in [9].

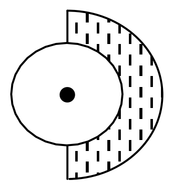
6. The present low cost of microprocessor/microcontroller (about ₦500 per unit) increase the cost effectiveness of microprocessor/microcontroller based system compared to other systems.

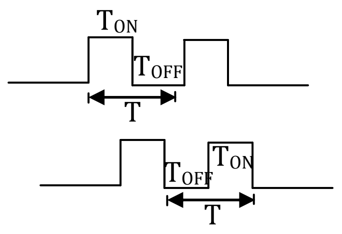
Great programming effort is required in software development for micros because of the close interaction between hardware and software. Thus any approach that can enhance software development speed is worthwhile. The State Transition Table (STT) approach is one of such software engineering methodologies that is both systematic and thorough. This design method will be showcased is this paper.

2. Software Engineering

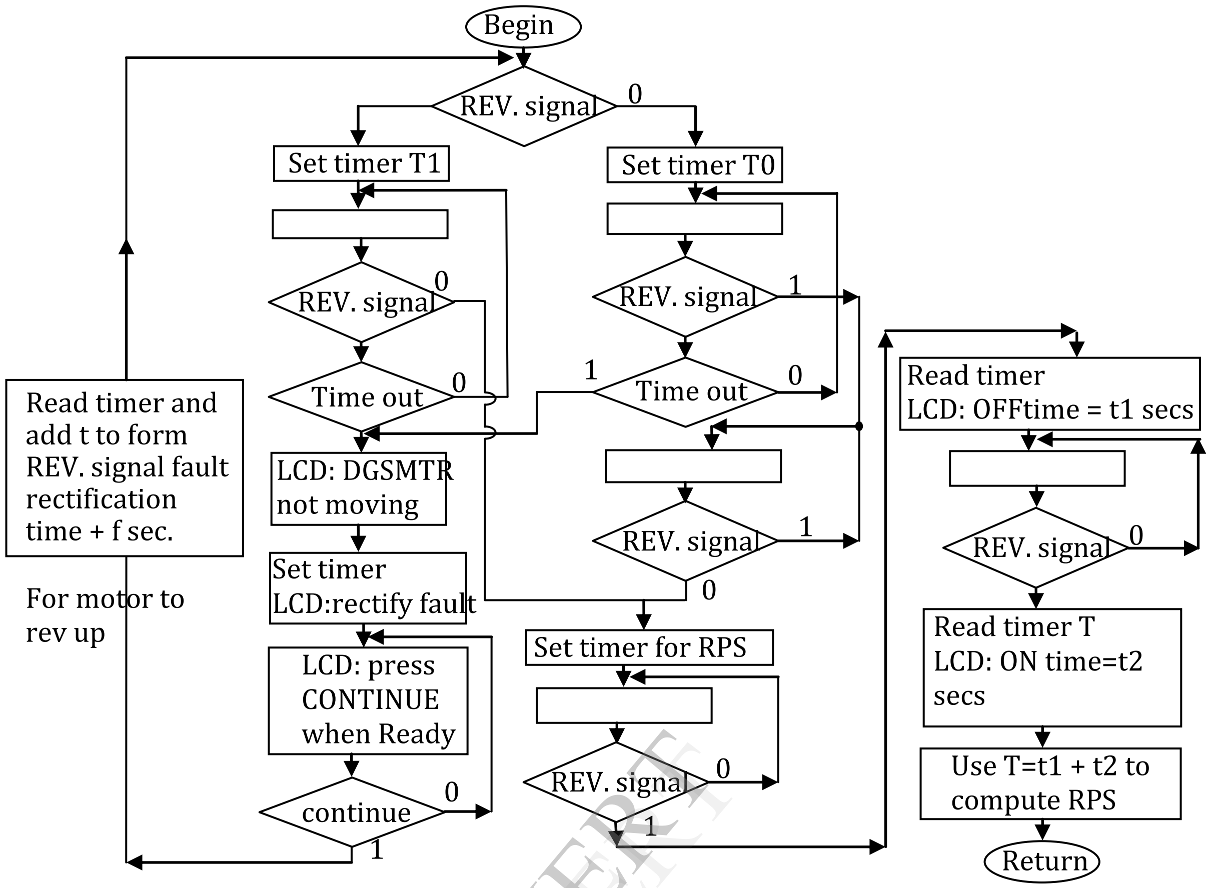
Software can be structured or unstructured. The problem with unstructured software is that it is not easily maintainable. Often such programs feature complicated flowcharts that are difficult to follow and the use of GOTO statements that tie the different portions of the program together in an untidy manner. For this reason, structured programming, often called GOTO-less programming is preferred. In digital systems design using micros, the trend has been to draw a flow chart and then write a software program in high level language or assembly language corresponding to the flowchart directly [2]. The control algorithm can either be structured or unstructured. For structured algorithms, it can be easier to write the program directly from the flowchart but for unstructured systems, it can be tasking and error prone.

Consider the problem to periodically measure motor speed in an application where the current speed of the motor is used to determine its state of efficiency and ability to cope with the load. Suppose there is a digital motor speed encoder (fig 1a), that generates an ON/OFF waveform (fig 1b) for every revolution of the motor. One then needs to develop a program to determine the time the motor spends in a cycle which corresponds to the ON time (T_{ON}) plus the OFF time (T_{OFF}) in the waveform of fig 1b. The motor speed in revolutions/second can then be determined once the time for one revolution is known. Fig 2 shows an unstructured flowchart to determine motor speed and the pseudo code corresponding to this is as shown in table 1. These contrasts sharply with the structured flowchart (fig 3) and the corresponding pseudo code (table 2). Obviously the latter is easier to follow than the former even though both would produce the same result.


"Fig 1a: digital motor speed encoder"



"Fig 1b: digital motor speed Time graph"

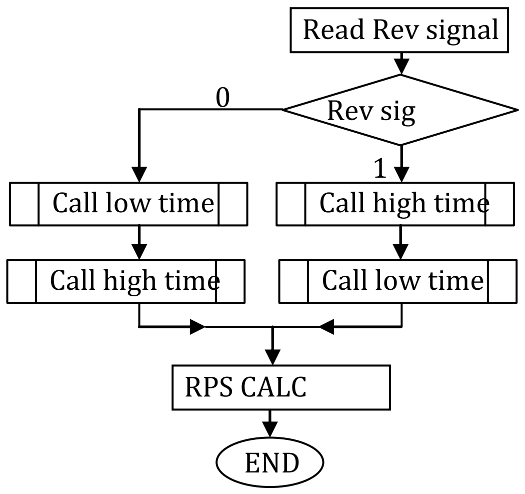

Begin
REV. signal
0
Set timer T1
Set timer T0
REV. signal
0
Time out
0
LCD: DGSMTR not moving
Set timer LCD:rectify fault
LCD: press CONTINUE when Ready
continue
0
1
Read timer and add t to form REV. signal fault rectification time + f sec.
For motor to rev up
REV. signal
1
Time out
1
0
REV. signal
1
0
Set timer for RPS
REV. signal
0
1
Read timer LCD: OFFtime = t1 secs
REV. signal
0
Read timer T LCD: ON time=t2 secs
Use T=t1 + t2 to compute RPS
Return

"Fig 2: unstructured algorithm for Revolution per second (RPS) calc."

"Table 1: Pseudo code for the unstructured software"

Pseudo code

```
If revolution signal (RevSig) is high then
 Begin    wait for the next falling edge
  Start timer to time the OFF time ($T_{OFF}$)
  Wait for the next rising edge
  Read the OFF time ($T_{OFF}$) from timer
  Restart timer to measure ($T_{ON}$)
  Wait for the next falling edge
  Read the ON time ($T_{ON}$) from timer
  Go To 100
 End.
Else
 Begin wait for the next rising edge
  Start timer to time the ON time ($T_{ON}$)
  Wait for the next falling edge
  Read the ON time ($T_{ON}$) from timer
  Restart timer to measure $T_{OFF}$
  Wait for the next rising edge
  Read the OFF time ($T_{OFF}$) from timer
  Go To 100
 End
ENDIF.
100 Begin
  T= $T_{ON}$ +$T_{OFF}$.
  Compute RPM using T display on LCD
End.
```



“Fig 3: A structured algorithm for calculating the RPS of a motor”

"Table 2: Pseudo code for the structured software”

```
Begin
  Test Revsignal
  If logic 1
    Call HIGH TIME
    Call LOW TIME
  ELSE
    Call HIGH TIME
  END IF
  CYCLE TIME=HIGH TIME +LOW TIME
  CALL rpm CALC
END
```

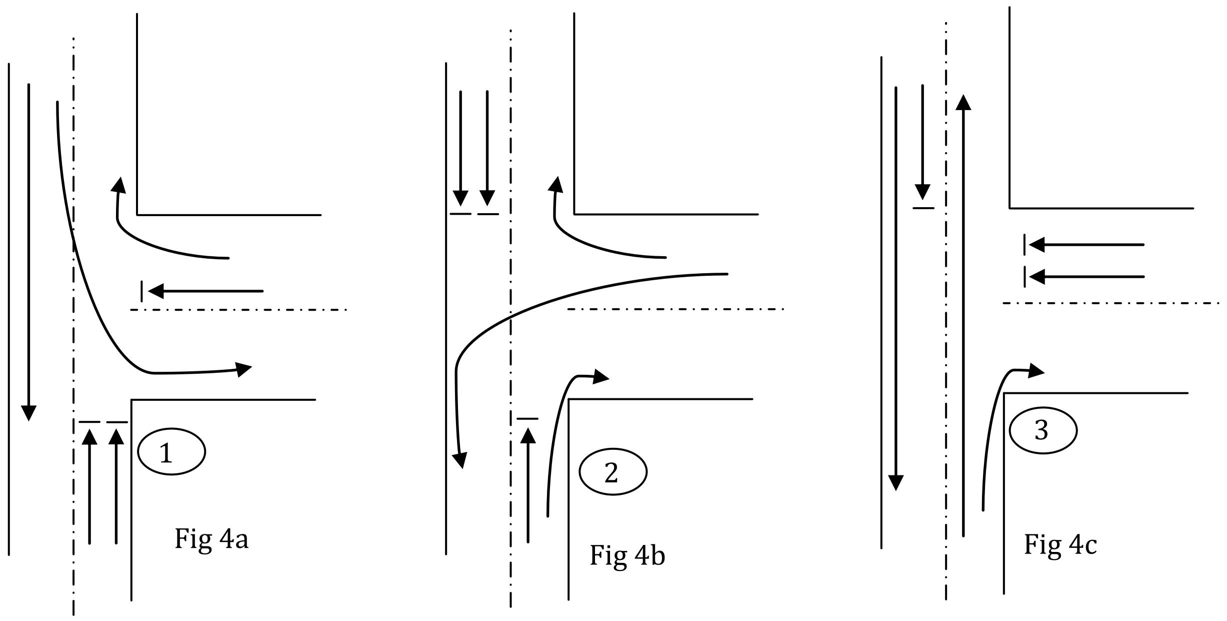
4. State Transition Table Based Control Software design

The steps involved in STT based control software design for micros include:

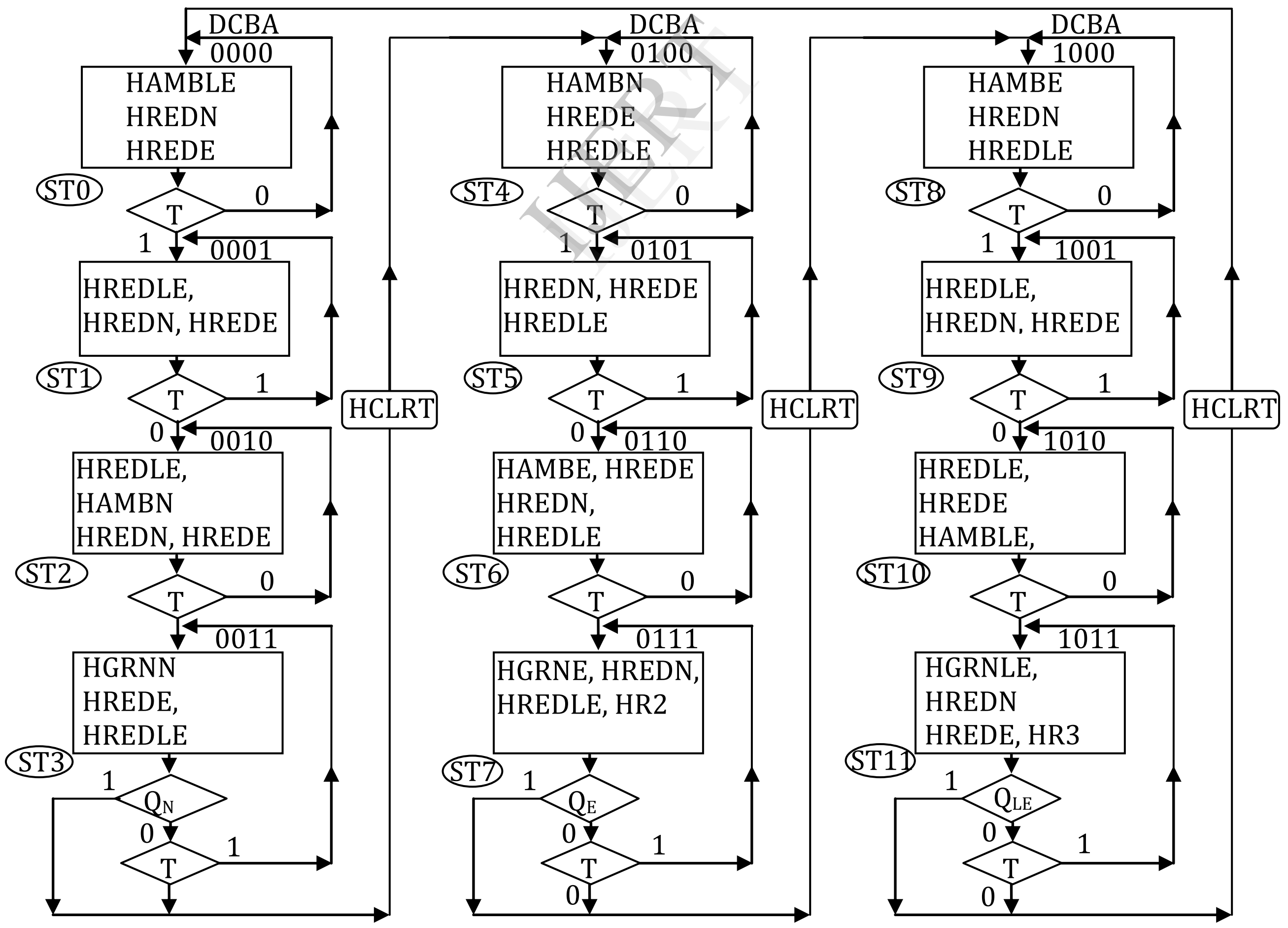
1. Put down the specifications for the envisaged system,
2. Develop the corresponding Algorithmic State Machine (ASM) chart,
3. Derive the STT corresponding to the ASM chart.
4. Reduce input/output line demand of the application using input multiplexing and output decoding as in [6], if need be.
5. Modify STT by placing the State Code column before the Qualifiers’ column.
6. Write Case Construct based implementation of the STT in software.
7. Fit software into the microcontroller as in [5]

5. Control System Example

A traffic light control system is desired for a T-junction. The sequence of movement for vehicles desired is in three phases as shown in Fig 4a through 4c. The arrows without a bar (-) in front in each case indicates the lanes that are passed while the arrows with a bar (-) in front are those on hold. A queue detector is featured in this design such that when there are no more vehicles on a direction that has right of access, the next direction is triggered.



"Fig 4: Traffic flow sequence in a T-junction"



"Figure 5: ASM chart for traffic light control at a T-junction with queue detectors Q_N, Q_E, Q_{LE}"

The Algorithmic State Machine (ASM) chart of figure 5 shows the control program flow for the sequence shown in Fig 4a through 4c. The conditional output HCLRT is used to clear the timing for the present sequence and move to the next when no queue is detected.

DC \ BA	00	01	10	11
00	0000	0001	0010	0011
01	0100	ST5	ST6	ST7
10	1000	1001	1010	1011
11	---	---	---	---

Fig 6A: State Map

State name	State code
ST0	0000
ST1	0001
ST2	0010
ST3	0011
ST4	0100
ST5	0101
ST6	0110
ST7	0111
ST8	1000
ST9	1001
ST10	1010
ST11	1011

Fig 6b: State Assignment

"Fig 6: State Map and State Assignment"

In the ASM chart (fig 5), N, NE and LE are used to differentiate the traffic light in each of the three directions. For example, HGRNLE, HAMBLE, HREDLE represents the Green, Amber and Red lights of one direction; HGRNN, HAMBN, HREDN are for the next direction while HGRNE, HAMBE, HREDE are for the third direction. (see figure 4a, 4b and 4c).

In an ASM chart, a rectangular box is called a state. A state has a state name in the lower left hand corner and state code at the top right hand corner as shown by [3]. Thus the ASM chart of fig 5 has twelve states. Since no two states may have the same state code, a state map followed by a state assignment is used to assign state code to ensure that this is achieved (fig 6).

Typically, when the ASM chart is going to be used for hardware design, the state codes are assigned such that only one bit changes as one moves from one state code to the next either above or below it. However in software based STT for micros, it is more convenient to assign the state codes serially in order that the corresponding case constructs based on state code might be easier to follow.

Another important component of the ASM chart is the decision box which has one entry point (and two exit points). Two or more decision boxes may occur in a sequence when more complicated decisions are to be made. It is also possible to have one state box leading to another without a decision box depending on the algorithm of the control logic.

The variables listed in the state box are the outputs that come ON when the process is in that state. They depend only on state codes. The outputs listed in the rounded boxes are conditional outputs that depend on the state code preceding it and on the logic levels of one or more of the qualifiers following that state. In the ASM chart of fig 5, HCLRT is a conditional output.

The information contained in an ASM chart (fig 5) can be represented as a flat table (table 3) called State Transition Table (STT). A path from a given state to another state or back to itself is called a link path. A link path may or may not contain qualifiers. When all the link paths in an ASM chart are placed in tabular form as in table 3, the table contains exactly the same information as is in the ASM chart. The reason for the transition from ASM chart to STT is that the remaining software or logic design steps are easier to visualize from a table than from the corresponding ASM chart.

The fields in an STT are as follows:

1. The link path column
2. The present state name column
3. The present state code group of columns
4. The qualifiers group of columns
5. The next state name column
6. The next state code group of columns and
7. The output group of columns [3].

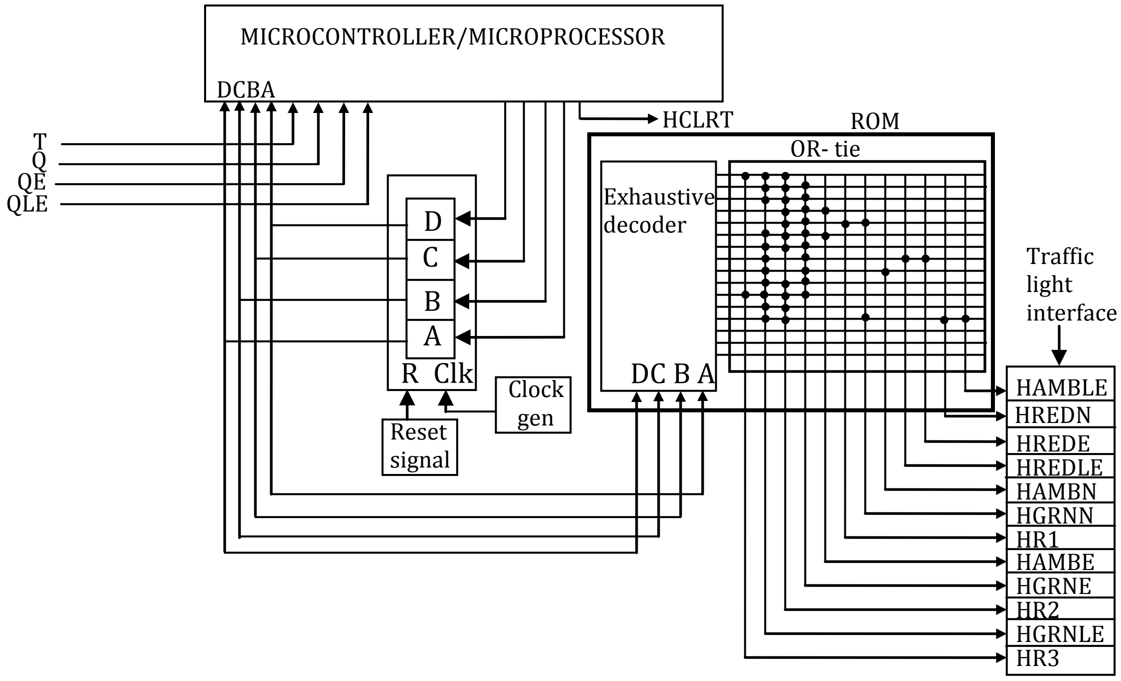
"Table 3: State transition table for the traffic light control"

Link path	Present state name	Present state code	Qualifiers	Next state name	Next state code	State output												Conditional output	Out-put in **HEX**
		DCBA	T Q_N Q_E Q_{LE}		D′C′B′A′	HAMBLE	HREDN	HREDE	HREDLE	HAMBN	HGRNN	HR1	HAMBE	HGRNE	HR2	HGRNLE	HR3	HCLRT	
L1	ST0	0000	0 - - -	ST0	0000	1	1	1	0	0	0	0	0	0	0	0	0	0	00
L2	ST0	0000	1 - - -	ST1	0001	1	1	1	0	0	0	0	0	0	0	0	0	0	02
L3	ST1	0001	1 - - -	ST1	0001	0	1	1	1	0	0	0	0	0	0	0	0	0	02
L4	ST1	0001	0 - - -	ST2	0010	0	1	1	1	0	0	0	0	0	0	0	0	0	04
L5	ST2	0010	0 - - -	ST2	0010	0	1	1	1	1	0	0	0	0	0	0	0	0	04
L6	ST2	0010	1 - - -	ST3	0011	0	1	1	1	1	0	0	0	0	0	0	0	0	06
L7	ST3	0011	1 0 - -	ST3	0011	0	0	1	1	0	1	1	0	0	0	0	0	0	06
L8	ST3	0011	0 0 - -	ST4	0100	0	0	1	1	0	1	1	0	0	0	0	0	1	09
L9	ST3	0011	- 1 - -	ST4	0100	0	0	1	1	0	1	1	0	0	0	0	0	1	09
L10	ST4	0100	0 - - -	ST4	0100	0	0	1	1	1	0	0	0	0	0	0	0	0	08
L11	ST4	0101	1 - - -	ST5	0101	0	0	1	1	1	0	0	0	0	0	0	0	0	0A
L12	ST5	0101	1 - - -	ST5	0101	0	1	1	1	0	0	0	0	0	0	0	0	0	0A
L13	ST5	0110	0 - - -	ST6	0110	0	1	1	1	0	0	0	0	0	0	0	0	0	0C
L14	ST6	0110	0 - - -	ST6	0110	0	1	1	1	0	0	0	1	0	0	0	0	0	0C
L15	ST6	0111	1 - - -	ST7	0111	0	1	1	1	0	0	0	1	0	0	0	0	0	0E
L16	ST7	0111	1 - 0 -	ST7	0111	0	1	0	1	0	0	0	0	1	1	0	0	0	0E
L17	ST7	0111	0 - 0 -	ST8	1000	0	1	0	1	0	0	0	0	1	1	0	0	1	11
L18	ST7	0111	- - 1 -	ST8	1000	0	1	0	1	0	0	0	0	1	1	0	0	1	11
L19	ST8	1000	0 - - -	ST8	1000	0	1	0	1	0	0	0	1	0	0	0	0	0	10
L20	ST8	1000	1 - - -	ST9	1001	0	1	0	1	0	0	0	1	0	0	0	0	0	12
L21	ST9	1001	1 - - -	ST9	1001	0	1	1	1	0	0	0	0	0	0	0	0	0	12
L22	ST9	1001	0 - - -	ST10	1010	0	1	1	1	0	0	0	0	0	0	0	0	0	14
L23	ST10	1010	0 - - -	ST10	1010	1	1	1	1	0	0	0	0	0	0	0	0	0	14
L24	ST10	1010	1 - - -	ST11	1011	1	1	1	1	0	0	0	0	0	0	0	0	0	16
L25	ST11	1011	1 - - 0	ST11	1011	0	1	1	0	0	0	0	0	0	0	1	1	0	16
L26	ST11	1011	0 - - 0	ST0	0000	0	1	1	0	0	0	0	0	0	0	1	1	1	01
L27	ST11	1011	- - - 1	ST0	0000	0	1	1	0	0	0	0	0	0	0	1	1	1	01

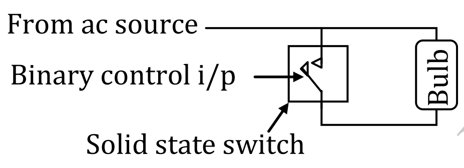
State transition table (Table 3) represents the ASM chart of fig 2. The qualifiers T, QE, QLE QN, and the present states D, C, B, A would constitute the input needed for this operation. Similarly, there are16 output lines, D′, C′, B′ A′, HAMBLE, HREDN, HREDE, HREDLE, HAMBN, HGRNN, HR2, HAMBE, HGRNE, HR1, HGRNLE. To limit the number of direct output lines from the processor to 8 or less, the state outputs are realized by decoding each present state. The next state codes (4-bits) and the conditional output HCLRT (1-bit) are generated directly from the output port. This approach reduced the number of direct output lines needed to just 5. A 4-to-16 line decoder can be used to generate all of the state outputs using the state codes D C B A as the control input to the 4-to-16 line decoder.

However, each output exists in more than one state of the ASM chart and this necessitates ORing of a number of decoded states for each output signal. Since an exhaustive decoder followed by OR-tie constitutes a ROM, it is better to use a ROM to achieve the dual purpose of decoding states and ORing different signals. This leads to the optimized design shown in fig 7 at the output side.

The signals from the ROM on the output side are used to turn the traffic lights ON or OFF via the traffic light interface. The interface could be implemented using Solid State Switches (SSS) as in [4] such that when a signal from the ROM is logic 1, the corresponding light turns ON and when it is logic 0, the light turns OFF. Thus a SSS receives a binary control input and connects Alternating Current (AC) power to the corresponding light if the binary control input is a '1'. The AC power is cut OFF from the corresponding light otherwise (fig 8).



"Fig 7: Microprocessor based implementation of T-junction traffic light control system [4]"



"Fig 8: AC power source switching a bulb"

Since the STT (table 3) already has the present state code before the qualifier bits as desired, one can now proceed to generate STT-based software, using state code in a case construct. When each state code is selected, the output options are determined by the qualifier bits attendant in that state. The pseudo code of the STT-based software now follows.

```
Do case
Case StateCode of
0: If T=0 then output =0 //link path 1
      Else output=2 End if // link path 2.
1: If T=0 then output = 2 //link path 3
       Else output =4 End if // link path 4.
2: If T=0 then output = 4 // link path 5
       Else output =6 End if //link path 6.
3: If T=1 And Qn=0 then output = 6 //link path 7
        Else if T=0 And Qn =0 then output =9 // link path 8
       Else if Qn=1 then output =9 End if // link path 9.
       End if
    End if
4: If T=0 then output = 8 // linkpath 10
       Else output =0Ah End if // link path 11.
5: If T=1 then output = 0Ah // link path 12
        Else output =0Ch End if // link path 13.
6: If T=0 then output = 0Ch // link path 14
       Else output =0Eh End if // link path 15.
7: If T=1 And Qe=0 then output = 0Eh // link path 16
       Else if T=0 And Qe =0 then output =11h // link path 17
          Else if Qe=1 then output =11 End if // link path 18.
      End if
   End if
8: If T=0 then output =10h //link path 19
       Else output =12h End if // link path 20.
9: If T=1 then output = 12h // link path 21
       Else output =14h End if // link path 22.
10: If T=0 then output = 14h // link path 23
         Else output =16h End if // link path 24
11: If T=1 And Qle=0 then output = 16h // link path 25
          Else if T=0 And Qle =0 then output =01h // link path 26
            Else if Qle=1 then output =01 End if // link path 27.
         End if
     End if
End case.
```

5. System Evaluation

If a fully expanded ROM-based design were to be implemented for the STT of table 5, it would have been necessary to first expand table 5 so as to eliminate the dashes (-), replacing them with binary combinations. A dash entry in table 5 implies that the qualifier at the top of that column does not feature in the states where it is represented as a dash. Because a dash cannot be programmed into a ROM, each combination of dashes would have to be replaced by the full binary combinations that are possible. Thus, each row in the STT with two dashes (- -) need to be replicated four times in each of where the two dashes are represented either as 00 or 01 or 10 or 11. Similarly any row with 3 dashes leads to 8 replications in each of which the 3 dashes are represented as either 000 or 001 or 010 or 011 or 100 or 101 or 110 or 111, and so on. Thus for a fully expanded STT needed for a ROM-based design, table 5 would have 192 rows instead of 27 rows in the original STT. This is called combinatorial explosion, which is a drawback for ROM-based designs, where unwanted combinations that contribute nothing to the logic would have to be represented in an exhaustive decoding process associated with ROMs.

Contrast this with Programmable Logic Array (PLA)-based design where a selective address decoder is used to represent only the rows shown in table 5, ignoring dashes. Thus the PLA height would be much shorter than for the corresponding ROM height.

The STT based software engineering shown in the foregoing used just one program statement to represent each row (or link path) of the STT. Thus it closely approximates a PLA-based design by not insisting on exhaustive decoding of unused combinations represented as dashes. Therefore for microprocessor/ microcontroller based design, the STT-based software engineering approach combines the selective decoding feature of PLAs with the flexibility of software associated with micros but not possible with PLAs.

7. Traffic Timing Signal T

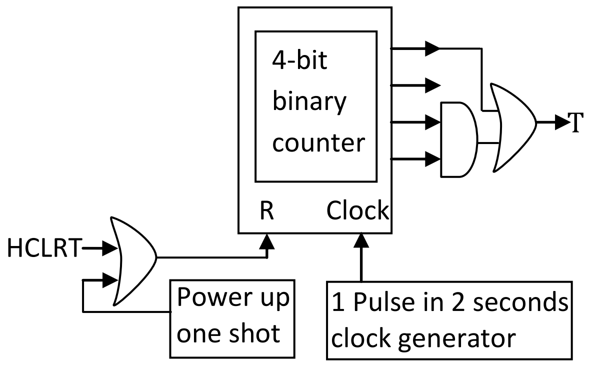
Each set of traffic control lights has duration as follows

6 seconds of amber for the direction about to hand-over right of way

2 seconds of red in the direction about to handover right of way

6 seconds of amber in the direction about to receive right of way

18 seconds of green to the direction that has the right of way.

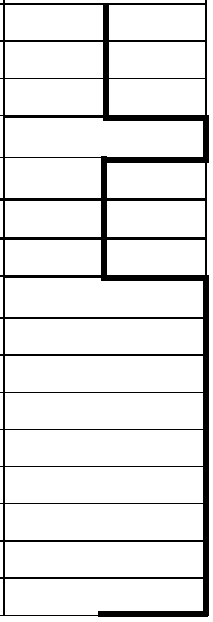


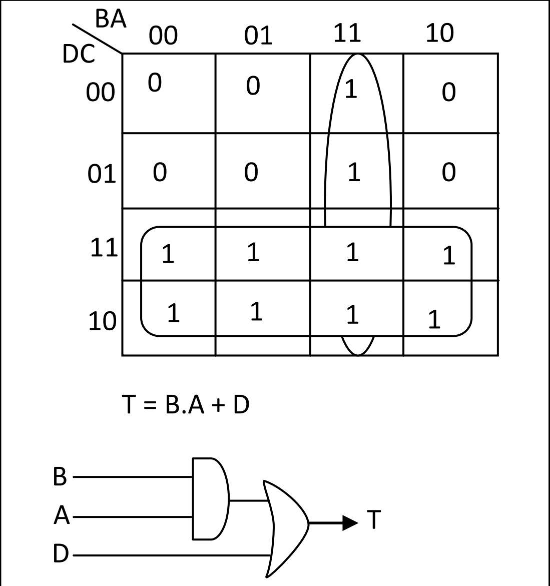
“Fig 9: Timing circuit for the traffic light control”

This timing ratio of 6:2:6:18 is the same as 3:1:3:9. Note that 6+2+6+18=32 whereas 3+1+3+9=16. The former requires a 5-bit counter whereas the latter requires a 4-bit counter for timing each of the three arms in a T-junction. So if one uses a 4-bit counter, clocked every 2 seconds, the timing objective can be realized (fig 9).

Table 4 shows the truth table for T in terms of the 4-bit counter output lines D, C, B, A. A power up one shot which sends a pulse when the power comes ON resets the 4-bit counter at start-up. Since 1 count is achieved every 2 seconds, T stays low for 3 counts (i.e. 6 seconds), then goes high for 1 count (i.e. 2 seconds) and goes low again for 3 counts (i.e. 6 seconds) and finally goes high for 9 counts (i.e. 18 seconds). When the queue in the direction that has right of way is finished before 18 seconds, HCLRT is generated to restart the timing cycle for another direction otherwise, the counter restarts by itself when it reaches 18 seconds in the direction of right of way. The k-map used to minimize the terms in the equation for T is shown in figure 10.

"Table 5: truth table of the 4-bit counter"

M	D C B A	T	T WAVEFORM
	0 0 0 0	0	
	0 0 0 1	0	
	0 0 1 0	0	
	0 0 1 1	1	
	0 1 0 0	0	
	0 1 0 1	0	
	0 1 1 0	0	
	0 1 1 1	1	
	1 0 0 0	1	
	1 0 0 1	1	
	1 0 1 0	1	
	1 0 1 1	1	
	1 1 0 0	1	
	1 1 0 1	1	
	1 1 1 0	1	
	1 1 1 1	1	



DC \ BA	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	1	1	1	1
10	1	1	1	1

T = B.A + D

"Fig 10: k-map, logic diagram and equation for T"

8. Conclusion

An STT table based software engineering for micros has been showcased in the foregoing. The approach is universal and can be applied to any process control system that can be represented as an ASM chart. The advantage of this approach is that it combines a thorough initial hardware design approach leading to the STT before generating the corresponding software that has just one statement per link path in the STT. This structured software is therefore easy to debug because of the one-to-one mapping of each statement with the rows of the STT. It combines the disciplined approach to hardware design with a structured software engineering approach to realize a dependable end product.

9. References

1. Bryan, L.A. and Bryan E. A., (2002). *Programmable Controllers Theory and Implementation,* 2nd edition. Illinois: American technical publishers Inc, 4-26.

2. Floyd, T. L., (2009). *Digital fundamentals,* 10th edition. Prentice hall of India:Pearson, 738-742.

3. Inyiama Hyacinth C.; Okezie Christiana C.; Okafo Ifeyinwa C., (2011a). Digital control of palm fruit processing using ROM based linked state machines. *European Journal of Scientific Research,* 59(4), 594-606.

4. Inyiama Hyacinth C.; Okafo Ifeyinwa C.; Okezie Christiana C., (2011b). PC based process control systems: problems and prospects. *International Journal of Academic reseasch,* 3(6), 183-189.

5. Inyiama Hyacinth C.; Okezie Christiana C.; Okafo Ifeyinwa C, (2012). Agent based process control system design. *Proceedings of the peer reviewed 2012 national conference on infrastructural development and maintenance in the Nigerian environment.* Nnamdi Azikiwe University, Awka 27-28 August. Nigeria.

6. Inyiama Hyacinth C.; Okezie Christiana C.; Okafo Ifeyinwa C, (in print). Complexity reduction in ROM-based process control systems via input multiplexing and output decoding. *International Journal of Engineering Innovations.*

7. RAM, B.,(2008). *Fundamentals of Microprocessor and Microcontrollers.* New-Delhi: Dhanpat Rai publications, 3.1-3.15, 1.46, 9.1-9.7.

8. Tanenbaum, Andrew S., (2006). *Structured computer organisation*, 5th edition. New-Delhi: Prentice Hall of India, 383-386.

9. Uzedhe, G. O. (2009). Design and implementation of a microcontroller based real-time emulator for basic logic gates and structured logic devices. *M.Eng Thesis,* Nnamdi Azikiwe University Awka. Nigeria.