Journal of Engineering and Computer Innovations Vol. 2(7), pp. 125-137, September 2011
Available online at http://www.academicjournals.org/JECI
ISSN 2141-6508 ©2011 Academic Journals

Full Length Research Paper

Versatile solid-state stepper motor controllers

Inyiama Height Chibueze and Okezie Christiana Chikodi*

Department of Electronic and Computer Engineering, Nnamdi Azikiwe University, Awka, Anambra State West Africa, Nigeria.

Accepted 8 June, 2011

Alternative forms of solid-state motor controllers have been presented in this paper including the effective use of read only memories in stepper motor controller implementations. An innovative low-cost universal stepper motor controller capable of controlling any kind of stepper motor has also been featured. The treatment includes a way of achieving a time-multiplexed control of several stepper motors with one universal controller and a method of generating a predetermined sequence of control bit-patterns for a dedicated application.

Key words: Motor controller, memory, stepper motor.

INTRODUCTION

The essential features of a steeper motor (SM) control system are illustrated in Figure 1. Four main parts are discernible, namely: a stepper motor, a stepper motor drive (or power circuit), a stepper motor controller, and an optional opto-coupler (or opto-isolator) between the drive and the controller.

The solid-state switches comprising the drive are logic level operated. A logic 1 control input applied via A, B, C or D closes the corresponding switch, allowing current to flow through the coil controlled by the switch. A logic zero input to any of the switches causes the switch to open, thereby interrupting coil-current flow.

A stepper motor is a device used to achieve incremental motion control and which takes one angular or linear incremental step depending on the design for every valid step command reaching the SM drive from its controller (Pickering, 2010). The size of the angular or linear incremental step and the direction of stepping (clockwise or anticlockwise for rotary motors, forward or backward for linear motors) depend on the nature of the control bit-pattern sequence generated by the SM controller and applied to the drive.

Tables 1 to 3 show the full-step, half-step, and ¼- step control bit-pattern sequences respectively, of a 4-phase stepper motor. Other fractional step sequences (for example, 1/8-step, 1/16-step etc.) are also possible. Note that a 4-phase SM controller is identical to a 2-phase controller but the 4-phase (Unipoar) drive is markedly from a 2-phase (bipolar) drive (Smith, 2009).

The maximum incremental distance for a stepper motor is the full step. The half-step is so called because it corresponds to a distance equal to the half of a full-step. Similarly, a quarter-step implies one quarter of the distance covered in a full-step and so on. Typically, the smaller the step size, the smoother the movement of the motor shaft and hence that of the load coupled to the motor shaft. Also, the torque levels the motor which is able to develop and increase with decreasing step size.

For any step size desired, the appropriate control bit-pattern sequence must be used. The bit-patterns are generated one row at a time and applied to the drive as step commands. The rate at which one row of bit-patterns in a sequence is replaced by the next corresponds to the SM stepping rate, as a step is always taken only at the point in time when one row of bit-patterns is replaced by the next. The clock (or step time) is an external signal applied to the SM controller and used to control the stepping rate. For every clock pulse, a new row of bit-patterns is generated. Therefore, the clock frequency corresponds to the stepping rate.

When the rows of a bit-pattern sequence are generated (and applied to the SM drive) in a top-to-down order, the SM steps in a clockwise or forward direction. In contrast, when the rows of the bit-pattern sequence are generated and applied to the drive in a bottom-up order, a counter-

*Corresponding author. E-mail:christianaokezie@yahoo.com. Tel: 234-8032266191.

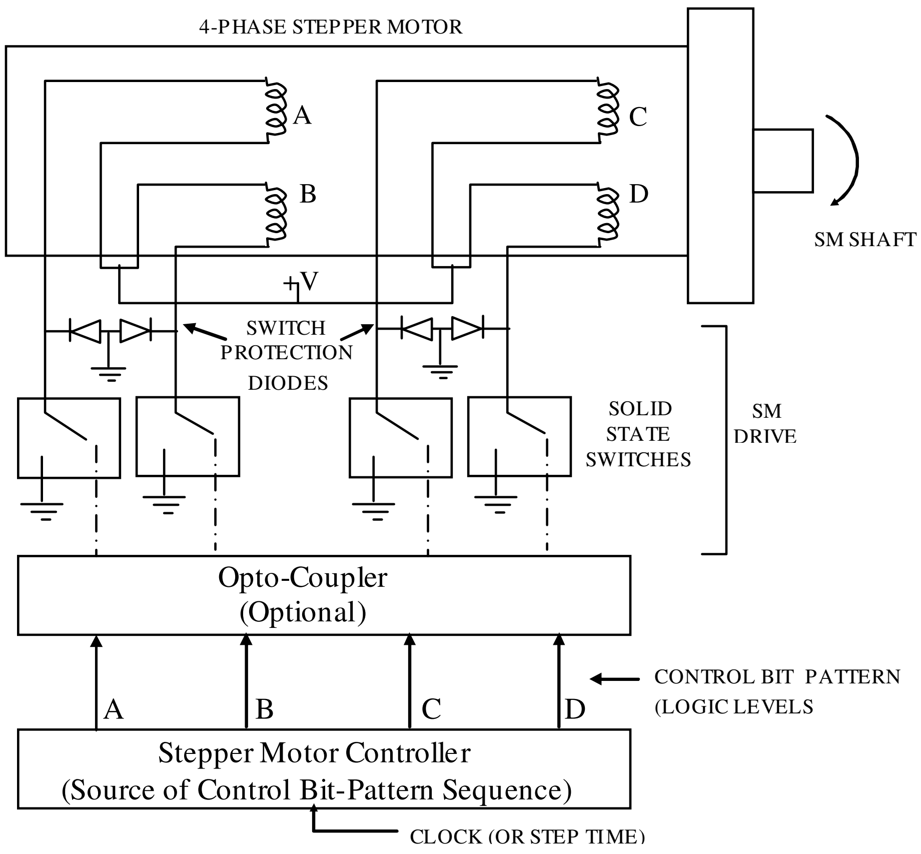


Figure 1. Essential fectures of a stepper motor control system.

Table 1. 4-Phase full-step sequence.

Full step	SM		Coils	
	A	**B**	**C**	**D**
1	1	0	0	0
2	1	0	1	0
3	0	1	1	0
4	0	1	0	1

1 = Full amplitude coil current, 0 = Zero coil current.

Table 2. 4-Phase half-step sequence.

Half step	SM		Coils	
	A	**B**	**C**	**D**
1	1	0	0	1
2	1	0	0	1
3	1	0	1	0
4	0	0	1	0
5	0	1	1	0
6	0	1	0	0
7	0	1	0	1
8	0	0	0	1

Table 3. 4-Phase, 1/4-Step sequence.

Step	SM		Coils	
	A	B	C	D
1	1	0	0	1
2	1	0	0	½
3	1	0	0	0
4	1	0	3/2	0
5	1	0	1	0
6	3/2	0	1	0
7	0	0	1	0
8	0	3/2	1	0
9	0	1	1	0
10	0	1	½	0
11	0	1	0	0
12	0	1	0	½
13	0	1	0	1
14	0	½	0	1
15	0	0	0	1
16	½	0	0	1

clockwise or backward stepping motion results. The rows of a bit-pattern sequence are generated as an endless chain. Thus, in the top-down direction the last row is immediately followed by the first, while in the bottom-updirection, the first row is followed by the last.

The sophistication built into SM controllers differs according to need. A controller may cater for just clockwise (CW) motion or for counterclockwise (CCW) motion. The motion may also be possible in full-steps only or in a particular fractional step only, depending on the Bit-pattern sequence the controller is designed to generate. A versatile SM controller capable of bidirectional full-step or fractional-step control can also be realized. Technique for achieving this are treated in this paper. Also detailed is a method of generating the control bit-pattern sequence for any stepper motor (and for any desired mode of stepping) from one and the same SM controller.

However, SM drives are not detailed here in view of the ready availability of suitable texts on the subject (Pickering, 2010; Smith, 2009; Brown and Vranesic, 2009).

ROM-SEQUENCER-BASED SM CONTROLLERS

As SM controller may take one of several alternative logic structures, however, the reliability and maintainability of the resultant circuit depend on the type of logic system employed.

When a sequential logic system such as that required to generate a stepper motor control bit-pattern sequence is implemented using discrete logic gates (such as AND, NAND, OR, NOR, EX-OR, INVRTERS etc.) and memory flip flops, what is termed a random logic system results (Floyd, 2002). Since such a system involves mostly small scale integration components, the component count is usually high for a fairly complex circuit. This implies several interconnections and many potential sources of error.

Modifications and maintenance are also difficult to achieve when either re-designing or fault-finding becomes necessary. A random logic system is therefore very inflexible.

Fortunately however, logic systems can be implemented in forms more structured than random logic. Such systems employ structured logic devices such as multiplexers (or data selectors) (Inyiama, 1981), and Read-Only-Memories (ROMs) (Floyd, 2002).

A multiplexer-based SM controller requires as many multiplexers as there are columns in the bit-pattern sequence to be generated and each of these multiplexers must have at least as many data input lines as there are rows in the bit-pattern sequence. Thus, if an 8-phase full-step sequence (which has 8 rows and 8 columns of bit patterns) is to be generated by means of multiplexers, a total of 64 data input lines would be involved. This is in addition to other control inputs and outputs necessary in such a system. The rapid proliferation of data input lines (and hence potential sources of error) in multiplexer based complex sequential logic systems is its main disadvantage in such applications. Multiplexers that are being structured by logic devices do, however, have a considerable advantage over random logic in that errors in the design can be corrected simply by altering the logic levels applied to their data inputs. For compact, reliable

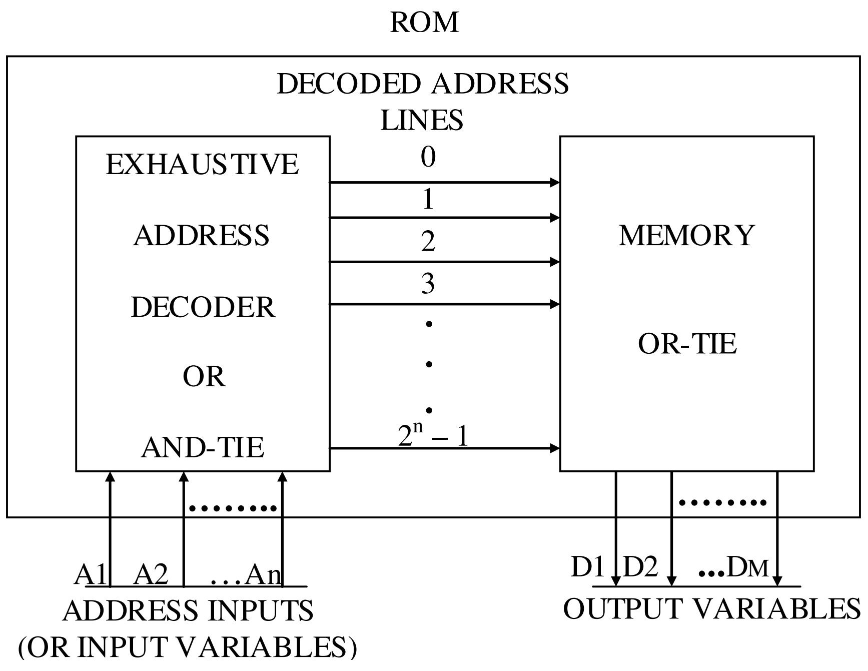


Figure 2. ROM structure.

and easily maintainable implementation of a complex sequential logic ROMs have an edge over multiplexers and are usually preferred.

The read only memory (ROM)

A read only memory (ROM) consists of 2 main sections: the address decoder section and the memory OR-tie section (Figure 2). The address decoder is an exhaustive AND-logic decoding of the control (now referred to as the address) inputs.

It is exactly the same in function as a conventional decoder except that its (active high) outputs are internally fed to the OR-tie section. The OR-tie section performs a similar function as a conventional OR-gate except that a number of OR-gate outputs can be derived from the OR-tie section of one ROM. Typically, the OR-tie section of a ROM is committed to a particular logic function during what is termed a programming process. Like the ROM decoder section the entire OR-logic circuits are internal to the ROM. Only the OR-gate outputs are sticking out on the ROM chip. Thus, the only external signals associated with a ROM are the control (or address) inputs to the decoder section and the OR-gate outputs from the OR-tie section (Figure 2). This is an important advantage for the ROM as the very few external signal lines imply fewer sources of error. Once the OR-tie section of a ROM is programmed, the pattern cannot be altered (that is, rewritten). It can be read, hence the name Read-Only-Memory by which this device is known.

During the initial development of a ROM-based logic circuit errors may occur in the OR-tie pattern programmed into it. Since reprogramming is not possible, the old ROM has to be thrown away and a fresh one programmed with the correct pattern. This is often considered wasteful. To overcome this disadvantage, other versions of ROMs which can be programmed, erased, and reprogrammed have been introduced. Examples of these include the ultra-violet Erasable, Programmable Read-Only-Memory (EPROM) and the Electrically Alterable Read-Only-Memory (EAROM). When all errors have been removed from the ROM pattern, the final correct version is usually programmed into a non-erasable ROM to guard against accidental erasure, a danger to which EPROM- or EAROM- based design are exposed.

ROM/Counter-based SM controllers

Table 4 shows the control bit-pattern sequence for a 6-phase SM while its ROM-sequencer based implementation is shown in Figure 3. The arrangement shown uses six consecutive ROM locations to store the six unique bit patterns in the sequence.

A modulus 6 (that is, divide-by-6) up/down counter (5) is used to generate the row address (that is, location) of the bit pattern to be output to the SM drive. For example, an address input such as C B A-000 selects the first row (that is, A2, B1, C1, A2, B2, C2, = 1, 0, 0, 0, 1, 0) and so on.

Table 4. The 6-Phase sequence.

Full step	SM			Coils		
	A1	B1	C1	A2	B2	C2
1	1	0	0	0	1	0
2	0	0	1	0	1	0
3	0	0	1	1	0	0
4	0	1	0	1	0	0
5	0	1	0	0	0	1
6	1	0	0	0	0	1

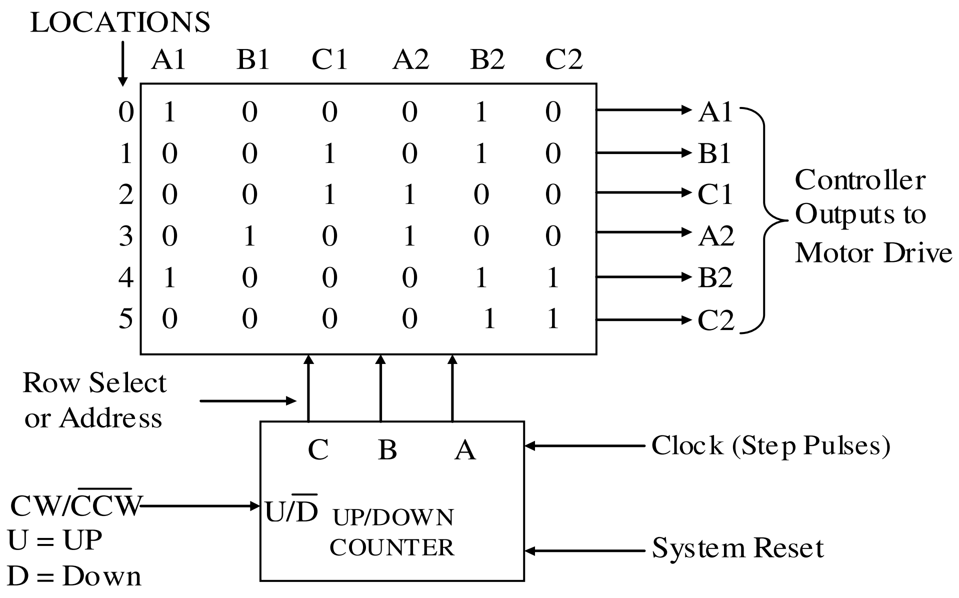


Figure 3. ROM-Sequence-based 6 phase stepper motor controller (CW AND CCW).

The clockwise/counterclockwise (CW/CCW) signal is the up-down input to the modulus counter. A logic 1 at the CW/CCW input means that the stepper motor is to move in a clockwise (or forward) direction while a logic 0 specifies a counter clockwise (or backward) motion. When CW/CCW = 1, the C B A outputs of the modulus counter access the ROM locations in the top-down (that is, clockwise) order, down to the last row and back to the first. When CW/CCW = 0, the ROM locations are accessed and output to the drive in the bottom-up (that is, counter-clockwise) order, starting from the last row, up to the first, and back to the last, in the usual endless-chain fashion. The clock or step pulse applied to the clock input of the modulus counter is derived from a square wave generator. The generator frequency must be chosen to correspond with the stepping rate desired as there is usually one step per clock pulse.

A bidirectional bit-pattern sequence generator may be similarly implemented for any stepper motor with corresponding ease of design. However, the number of ROM locations needed will vary with the number of rows in each bit-pattern sequence. The same is true of the maximum count required of the up/down counter before it restarts form zero.

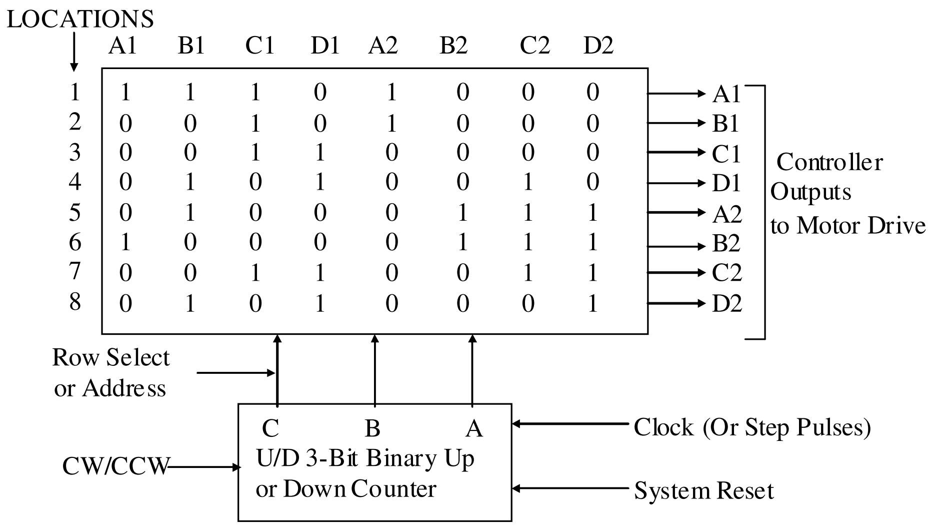
Table 5 shows the full-step sequence for an 8-phase SM. Its ROM-based implementation is shown in Figure 4. Note that the 3-bit ripple up/down. Counter shown in Figure 4 is capable of generation eight unique addresses, corresponding to the eight ROM locations required to store the 8-phase sequence. Any of the ROM versions may be used in these implementations but with the implications high-lighted earlier (The read only memory (ROM)).

Fully expanded SM control sequences

It is possible to design an SM controller in which the present output bit-pattern is used as the address of the next output patter in the sequence. This approach

Table 5. 8-Phase full-step sequence.

Full step	SM			Coils		
	A1	**B1**	**C1**	**A2**	**B2**	**C2**
1	1	0	0	0	1	0
2	0	0	1	0	1	0
3	0	0	1	1	0	0
4	0	1	0	1	0	0
5	0	1	0	0	0	1
6	1	0	0	0	0	1



LOCATIONS	A1	B1	C1	D1	A2	B2	C2	D2
1	1	1	1	0	1	0	0	0
2	0	0	1	0	1	0	0	0
3	0	0	1	1	0	0	0	0
4	0	1	0	1	0	0	1	0
5	0	1	0	0	0	1	1	1
6	1	0	0	0	0	1	1	1
7	0	0	1	1	0	0	1	1
8	0	1	0	1	0	0	0	1

Figure 4. ROM sequencer-based bi-directional b-Phase stepper motor controller.

Table 6. 5-Phase full-step sequence.

Half	SM		Coils			Full-step codes
	A	**B**	**C**	**D**	**E**	
1	1	0	1	0	1	1
2	0	0	1	0	1	
3	0	1	1	0	1	2
4	0	1	0	0	1	
5	0	1	0	1	1	3
6	0	1	0	1	0	
7	1	1	0	1	0	4
8	1	0	0	1	0	
9	1	0	1	1	0	5
10	1	0	1	0	0	

obviates the need for modulus up/sown counters but requires the use of much more expanded bit-pattern sequence table.

Table 6 shows the half-step bit-pattern sequence for a 5-Phase stepper motor while Table 7 shows its fully expanded version. Table 7 covers four possible modes of SM operation and is comprised of four main sections as follows:

(a) Section (1): when the bit pattern under the CW/CCW and F/H (that is, full-step or half-step) columns are both zero (or CW/CCW, F/H = 0, 0) signifying the

Table 7. Fully expanded state transition table for a CW/CCW full/half step 5-Phase stepper motor controller.

HEX	Qualifiers		Present state					Next states					HEX′
	CW/CCW	F/H	A	B	C	D	E	A′	B′	C′	D′	E′	
15	0	0	1	0	1	0	1	1	0	1	0	0	14
05	0	0	0	0	1	0	1	1	0	1	0	1	15
00	0	0	0	1	1	0	1	0	0	1	0	1	05
09	0	0	0	1	0	0	1	0	1	1	0	1	0D
08	0	0	0	1	0	1	1	0	1	0	0	1	09
0A	0	0	0	1	0	1	0	0	1	0	1	1	08
1A	0	0	1	1	0	1	0	0	1	0	1	0	0A
12	0	0	1	0	0	1	0	1	1	0	1	0	1A
16	0	0	1	0	1	1	0	1	0	0	1	0	12
14	0	0	1	0	1	0	0	1	0	1	1	0	16
35	0	1	1	0	1	0	1	1	0	1	1	0	16
25	0	1	0	0	1	0	1	1	0	1	0	0	14
20	0	1	0	1	1	0	1	1	0	1	0	1	15
29	0	1	0	1	0	0	1	0	0	1	0	1	05
28	0	1	0	1	0	1	1	0	1	1	0	1	0D
2A	0	1	0	1	0	1	0	0	1	0	0	1	09
3A	0	1	1	1	0	1	0	0	1	0	1	1	08
32	0	1	1	0	0	1	0	0	1	0	1	0	0A
36	0	1	1	0	1	1	0	1	1	0	1	0	1A
34	0	1	1	0	1	0	0	1	0	0	1	0	12
55	1	0	1	0	1	0	1	0	0	1	1	0	05
45	1	0	0	0	1	0	1	0	1	1	0	0	0D
40	1	0	0	1	1	0	1	0	1	0	0	1	09
49	1	0	0	1	0	0	1	0	1	0	0	1	08
4B	1	0	0	1	0	1	1	0	1	0	0	1	0A
4A	1	0	0	1	0	1	0	1	1	0	0	1	1A
5A	1	0	1	1	0	1	0	1	0	0	1	1	12
52	1	0	1	0	0	1	0	1	0	1	1	0	16
56	1	0	1	0	1	1	0	1	0	1	1	0	14
54	1	0	1	0	1	0	0	1	0	1	1	0	15
75	1	1	1	0	1	0	1	0	1	1	0	1	0D
65	1	1	0	0	1	0	1	0	1	0	0	1	09

Table 7. Contd.

60	1	1	0	1	1	0	1	0	1	0	1	1	08
69	1	1	0	1	0	0	1	0	1	0	1	0	0A
6B	1	1	0	1	0	1	1	1	1	0	1	0	1A
6A	1	1	0	1	0	1	0	1	0	0	1	0	12
7A	1	1	1	1	0	1	0	1	0	1	1	0	16
72	1	1	1	0	0	1	0	1	0	1	0	0	14
76	1	1	1	0	1	1	0	1	0	1	0	1	15
74	1	1	1	0	1	0	0	0	0	1	0	1	05

counter-clockwise, half-step mode
(b) Section (2): when CW/CCW, F/H = 01, signifying the counter clockwise, full-step mode
(c) Section (3): when CW/CCW, F/H = 10, which is the clockwise, half-step mode; and
(d) Section (4): when CW/CCW, F/H = 13, the clockwise, full-step mode.

The control bit-pattern sequence for each of the four modes of operation appears under the column headed by the label "Present States" and in the section for that mode.

The columns of bit patterns under Present States are labelled A, B, C, D, and E. Each row of bit-patterns under Next States (labeled A^1, B^1, C^1, ^{1} and E^1) is obtained by determining the next appropriate control bit-pattern following that under Present States (on the same row) when the mode of stepping is as defined by the CW/CW, F/H bit-pattern. For example, when CW/CCW, F/H -0, 0 (counter clockwise half-step mode) and the present states ABCDE = 10101, The Next States A^1 B^1 C^1D $^1E^1$ = 10100. This is because in the reverse order of the 5-Phase bit-pattern sequence (Table 6) 10101 (top row) is followed by 10100 (bottom row). Similarly, when CW/CCW, F/H -11 (clockwise, full-step) and ABCDE = 10101, the next states $A^1B^1C^1D^1E^1$ = 01101, which is the next bit-pattern in the full-step sequence in a top-down direction (Table 6) and so on.

The preparation of the present states and next states table as illustrated in Table 7 is a necessary step used first in the realization of a versatile SM controller. Such a table is often referred to, as a fully expanded state transition table (STT) for the particular motor whose bit-pattern sequence is so expanded. When such tables are used in SM controller design, error-free transitions between the full-step and half-step modes are made possible.

In ROM-based designs, a fully expanded table such as Table 7 can be generated by viewing the bit-pattern under CW/CCW, F/H A, B, C, D and E as a ROM address, storing each control bit-pattern under A^1, B^1, C^1, ^{1} and E^1 in the address on the same row as that control bit-pattern and using an appropriate number of presentable flip flops both to define the first bit-pattern to be generated and to transform next states into present states when the step command (clock pulse) occurs. Looking back at Table 5, the hexadecimal value of each ROM address is shown under the column labeled HEX while its content is shown on the same row under HEX^1 A suitable ROM-based implementation of Table 7 is shown in Figure 5. The result is a content addressable, bidirectional full-step or half-step, ROM/EPROM/EAROM based, 5-phase, SM controller.

Multiplexer based stepper-motor controllers

ROM-based design is not the only approach to stepper-motor controller design. Note that each SM bit-pattern sequence is comprised of a number of bit patterns. Each column can be implemented by means of a multiplexer (MUX) or data selector. Thus a 4 phase or 4 phase motor with 4 columns in their bit-pattern sequence will require 4 multiplexers; a 5-phase SM with 5 multiplexers, and so on.

To implement a column of bit-pattern with a multiplexer, apply the logic level of each bit in the pattern to the corresponding multiplexer data input pin. To obtain the bit pattern for each motor step, the outputs of all the multiplexers are used parallelly. Having this in mind, the multiplexer based on SM controller for a 2 phase or 4 phase is able to direct the full step sequence (Tables 1 to 4, and Figure 6).

Note that the first multiplexer implements the first column (column A) of Tables 1 to 4, the second multiplexer implements column B, the

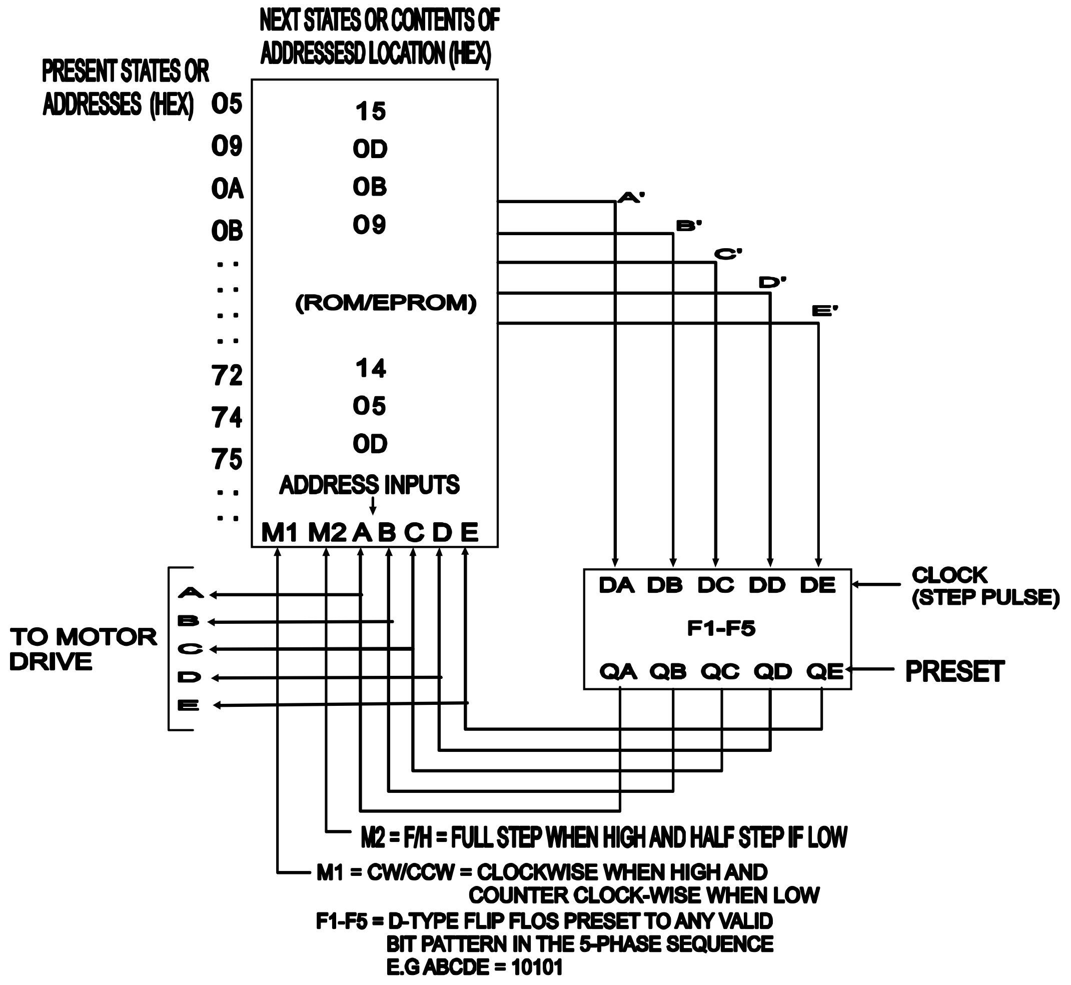

NEXT STATES OR CONTENTS OF ADDRESSESD LOCATION (HEX)
PRESENT STATES OR ADDRESSES (HEX)
05
09
0A
0B
72
74
75
15
OD
OB
09
(ROM/EPROM)
14
05
OD
ADDRESS INPUTS
M1 M2 A B C D E
A'
B'
C'
D'
E'
TO MOTOR DRIVE
A
B
C
D
E
DA DB DC DD DE
F1-F5
QA QB QC QD QE
CLOCK (STEP PULSE)
PRESET
M2 = F/H = FULL STEP WHEN HIGH AND HALF STEP IF LOW
M1 = CW/CCW = CLOCKWISE WHEN HIGH AND COUNTER CLOCK-WISE WHEN LOW
F1-F5 = D-TYPE FLIP FLOS PRESET TO ANY VALID BIT PATTERN IN THE 5-PHASE SEQUENCE E.G ABCDE = 10101

Figure 5. Content addressable bidirectional full step of half step, ROM/EPROM – Based, 5-Phase stepper motor controller.

third column C and the fourth column D. Note that the bit pattern under each column in Tables 1 to 4 was applied as data input to the multiplexer used to implement it. To feed logic O as data input, the circuit ground is simply connected to that point. To connect a logic, + 5V is connected via 1 K_2 resistor as shown in Figure 6.

All of the SM controllers mentioned so far can be similarly implemented, except that the number of multiplexers used must match the number of columns of bit patterns, and the height data input pins as there are bits in a column. If a 1-out-of-4 MUX is not tall enough for a given SM controller, use a 1-out-of 8 MUX (plus a 3-bit up/down counter), and if a 1-out-of 8 MUX is not tall enough, use a 1-out-of 16 MUX for each column (plus a 4-bit up/down counter). In each implementation unused MUX data input pins are connected to ground in those cases where all the available data input pins do not have corresponding bit in the column of bit pattern being implemented. This happens when the MUX is taller that the column of bit pattern in terms of the number of data input pin positions.

THE SOLID-STATE UNIVERSAL SM CONTROLLER

Many applications, for example, Research and Development (R & D) work, require the use of different stepper motors of varying number of phased. At a time of austerity, R & D funding is often slashed and it may not be possible to acquire separate SM controllers for the various motors in use. The high capacity of ROMs, EPROMS, and EAROMs relative to the number of unique bit-patterns to be generated per stepper motor suggest the use of a single memory to store the bit pattern sequence of all the stepper motors in use. The bit-pattern sequence for any particular stepper motor would then be reached by supplying address inputs which access only the portion of memory where it is stored. Such a device is what is termed a solid state universal stepper motor controller (USMC).

The USMC would be particularly useful in applications where only one of the different types of stepper motors is in use at any given time. One USMC could then be time-multiplexed among all of them. The USMC idea would

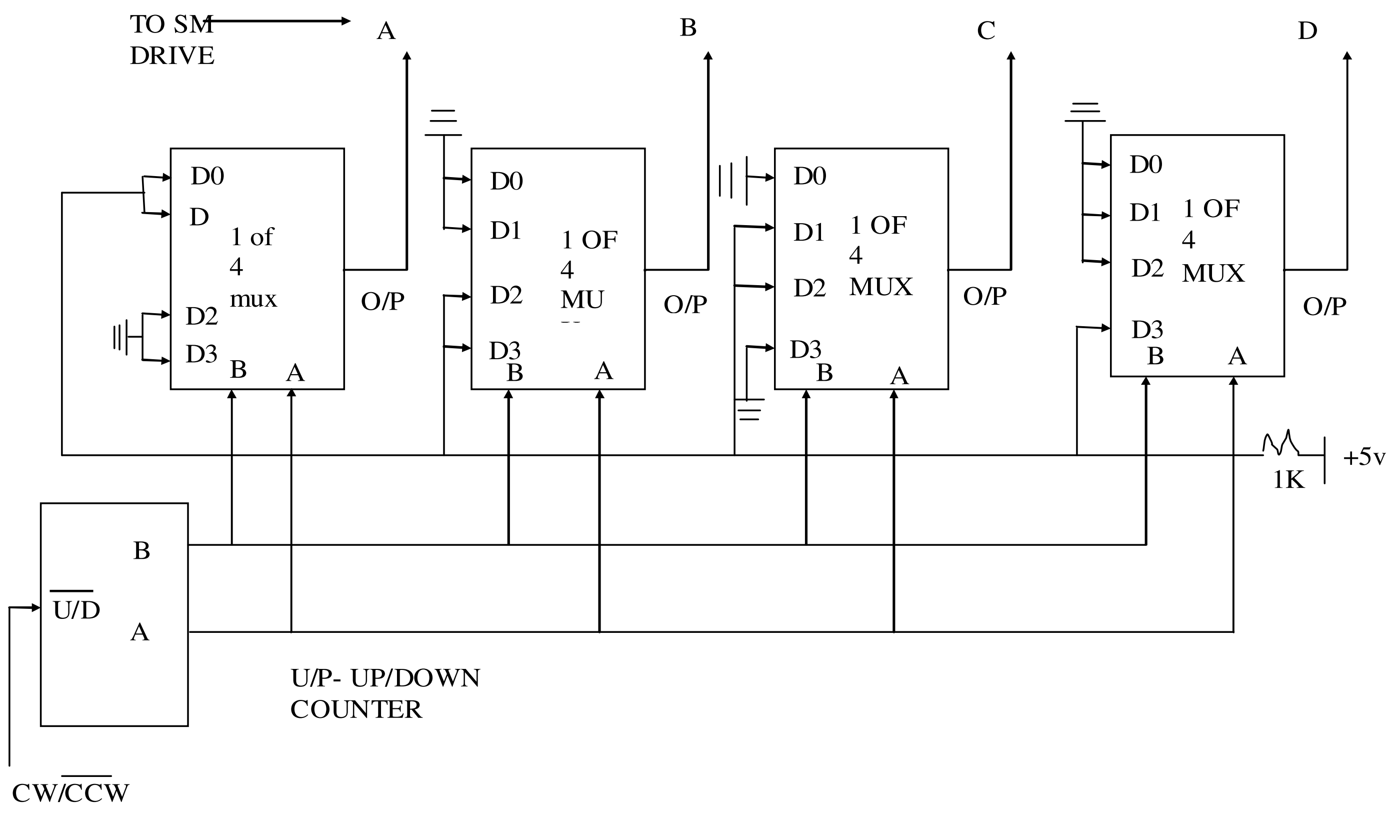


Figure 6. Multiplexer-based bidirectional full step 2-Phase/4-Phase SM controller.

also be very useful in stepper-motor-intensive applications where groups of identical stepper motors can be formed. All the SM drives in any one group would then be controlled simultaneously ad the various groups can also be driven simultaneously be the same USMC in a time-multiplexed fashion. Alternatively, a USMC may be dedicated to just one group of stepper motors especially where not all the motors in one group need to be operated concurrently. There would then be as many USMCs as there are groups of stepper motors, a feasible approach in view of the present low-cost of all the types of ROMs that may be used.

The use of up/down modulus counters to generate the next state addresses would be cumbersome in a USMC approach due to the need to change the counter when its maximum count is unsuited to the motor to be driven next. It is therefore preferable to use the current outputs of the controller as the next-state address inputs, an approach which necessitates the development of a fully expanded state transition table for each stepper motor to be controlled by the USMC.

The advantages associated with the USMC more than adequately compensate for this extra design effort. Figure 7 illustrates a solid-state Universal SM controller capable of generating the following sequences:

(1) Bidirectional half-step/full-step sequences for 2-phase, 4-phase and 5-phase motors and
(2) Bidirectional full-step sequences for 3-phase, 6-phase and 8-phase stepper motors.

The control bit-pattern sequences for these motors have been presented earlier (Tables 1, 2, 3, 5, 6, 7 and 8) with the exception of the 3-phase SM with control bit-pattern sequences for the six types of motors listed earlier can be developed in the same way as for the 5-phase SM (Multiplexer based stepper-motor controllers). Tables 9 to 11 indicate that the low inputs/outputs of the USMC may be interpreted for each type of motor provided for. There is room in the same piece of 2 Kbyete ROM used in this implementation for three additional types of stepper motors. The USMC as shown would cost less than ten pounds sterling to implement. Any desired number of different stepper motors may be catered for using the USMC approach, although the size of ROM used may vary depending on the exact number of motors.

In some applications, the stepper motor may be required to do a definite sequence of clockwise and counter-clockwise combinations of full-steps or half-steps or both. This pre-determined sequence is usually activated by means of a start command, and once the sequence is completed, the SM controller waits for another start command before repeating the sequence. ROM, EPROM or EAROM based designs are very useful

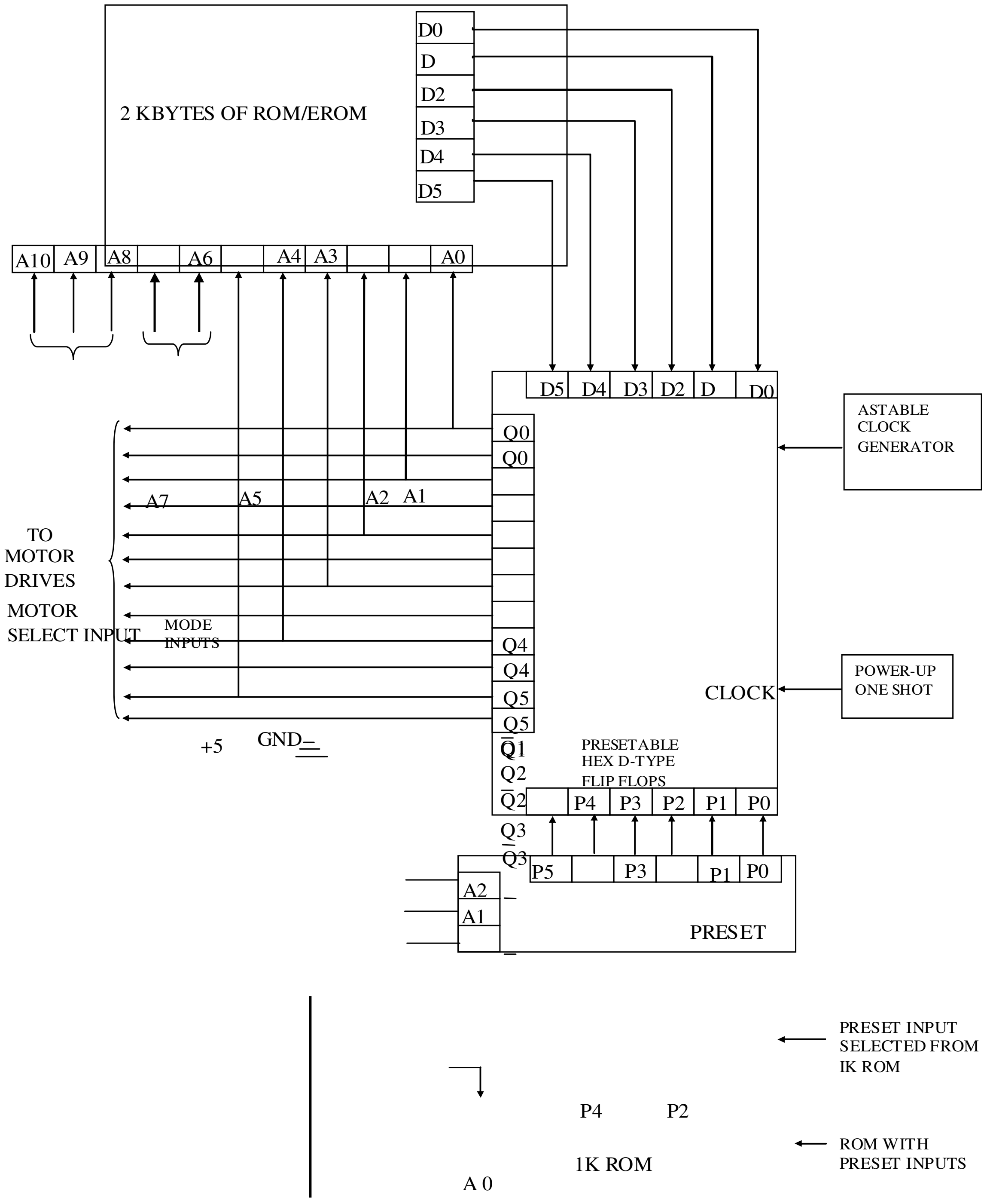


Figure 7. The universal hardwired stepper motor controller.

Table 8. 3-Phase full step sequence.

Full step	SM	Coils	
	A	B	C
1	1	0	0
2	0	0	1
3	0	1	0

Table 9. Address and present-input programming.

Address inputs			Number of phases	Preset inputs					
A10	A9	A8		P5	P4	P3	P2	P1	P0
0	0	0	2 or 4	0	0	1	0	0	1
0	0	1	3	0	0	0	1	0	0
0	1	0	5	0	1	0	1	0	1
0	1	1	6	1	0	0	0	1	0
1	0	0	8	0	0	1	1	1	1
1	0	1		Available for expansion					
1	1	1							

Table 10. Interpretation of mode inputs.

M1 = CW/CCW, M2 = F/H		Meaning of mode inputs
0	0	Counter clockwise, half step
0	1	Counter clockwise, full step
1	0	Clockwise, half step
1	1	Clockwise, full step

Table 11. Connections of USMC outputs to motor-drive inputs.

z	A5	A4	A3	A2	A1	A0	A3	A2	A1	A0
2 or 4			A2	A1	B1	B2				
3				A	B	C				
5		A	B	C	D	E				
6	A1	B1	C1	A2	B2	C2				
8			A1	B1	C1	D1	A2	B2	C2	D1

to such applications since the entire bidirectional combinations of full-step and/or half-step control bit-patterns can be stored in consecutive memory locations in the order required.

The present outputs of the memory device in use may be used as address inputs for the next bit-pattern to be generated. With this approach the memory location corresponding to the last code in the sequence may be made to contain another copy of the last code. The last SM controller outputs bit-pattern is thus retained and the controlled SM will remain in its current position.

The arrangement should be such that when a start signal is added to the last control bit-pattern, the address of the first code in the sequence is obtained and the stepping sequence restarts. It should be remembered that the clock frequency ought to be doubled for the half-step bit-pattern if the motor speed is to be maintained. Otherwise, the motor speed will drop by a factor of two during half-steps.

SUMMARY AND CONCLUSIONS

The basic operating principles of a stepper motor including the implications of full-step and fractional-step control-bit-pattern sequences have been explored. The usefulness in SM controller implementations of various forms of logic devices has been reviewed leading to a preference for ROM-based SM controllers.

The relative ease with which various types of ROMs can be combined with modulus counters to realize effective SM controllers has been demonstrated. Also highlighted are the design steps leading to the realization of an innovative, low cost, solid-state, universal SM controller, a unit which can be used to control any type of stepper motor and which is capable of bidirectional (incremental) motion control in either full steps or fractional steps.

The ever increasing automation in the world of today underlines the need for control engineers and allied professionals to become conversant with automatic control devices. The ease of design coupled with the low-cost feature of versatile SM controllers recommend the study of SM-based control as a convenient starting point in the quest for a master of the vast field of automation.

REFERENCES

Pickering A (2010). The Cybernetic Brain: Sketches of Another Future University Of Chicago Press.

Brown S, Vranesic Z (2009). Fundamentals of Digital Logic with VHDL Design. 3rd ed. New York, N.Y.: Mc Graw Hill.

Floyd TL (2002). Digital Fundamentals, 8th edition (Prentice Hall); ISBN 978-0130942005

http://books.google.co.uk/books?id=TxKynbyaIAMC&dq=Instrument+Engineers%27+Handbook&pg=PP1&ots=jvrdPR7wxJ&sig=1hOUpQQDQH_8drYjW1yPVocJSYI&hl=en&sa=X&oi=book_result&resnum=1&ct=result.

Inyiama HC (1981). Stepper Motor Control Handbook", A technical report prepared for the Faculty of Technology, University of Manchester Institute of Science and Technology (UMIST), England, pp. 1-100.

Smith AO (2009). The AC's and DC's of Electric Motors (PDF). http://www.aosmithmotors.com/uploadedFiles/AC-DC%20manual.pdf. Retrieved 2009-12-07.