1974

AN-722
Application Note

REPLACING SEQUENTIAL LOGIC WITH ROMS

Prepared by:
Jerry E. Prioste
Computer Applications Engineering
Motorola Semiconductor Products, Inc.

As ROM sizes increase, the cost per bit continues to drop resulting in new uses for ROMs previously thought not practical or too costly. This note discusses how to use ROMs to replace random logic with feedback loops. The advantage of using ROMs is reduced package count and a more cost effective system. Three design examples are given to show the implementation techniques involved in achieving a reliable design without pitfalls. The techniques shown will hopefully inspire others to use these techniques in present designs and to develop new innovative techniques for using ROMs.

MOTOROLA *Semiconductor Products Inc.*

REPLACING SEQUENTIAL LOGIC WITH ROMS

INTRODUCTION

Read-Only Memories (ROMs) have become increasingly popular in logic system design. The number of storage bits on a ROM chip have increased greatly each year resulting in lower costs per bit. In turn, these lower costs have stimulated new applications for ROMs, such as replacing random logic devices. Microgramming, for example, is being widely used in the design of computers: it replaces the control logic with ROMs. This results in reduced package count and lower costs while adding system versatility. The larger sized ROMs also have been used in the macroprogramming area for storing often-used software programs; in fact, many of the new microprocessors depend on this technique. ROMs are also used in more traditional applications such as look-up tables, code conversion, and character generation.

This article describes the design methods for using ROMs to replace sequential logic. Sequential logic is considered as any design that requires random or combinational logic containing feedback loops. Design techniques for implementing a sequential logic design in gates and flip-flops are described in Reference 7. The MC5003, 64 word x 8 bit programmable ROM, will be used to illustrate the techniques involved in implementing sequential designs with ROMs. Larger ROMs are available that make the methods described here even more advantageous.

COMBINATIONAL LOGIC

The technique for the replacement of combinational logic ROMs will be discussed first before looking at sequential logic design with feedback. Conversion of combination logic to a ROM can be accomplished if any of the following are available:

(1) Logic Equations
(2) Truth Table
(3) Logic Drawing

If logic equations are given, the equations must be converted to a Karnaugh map in order to define each output for all combinations of inputs. Then, the content of the Karnaugh maps is placed in a truth table to establish the pattern that the ROM must contain. If a logic drawing must be converted to a ROM, the equations for each output can be written. Then the procedure follows as discussed above until the truth table is established.

To determine the number of gates that must be replaced before the ROM becomes economical for replacing random or combinational logic can be estimated by this formula:

$$\text{number of gates replaced per ROM} > \frac{\$\text{ROM} + \text{associated cost/ROM}}{\$\text{GATE} + \text{associated cost/gate}}$$

This formula states that the number of gates replaced for each ROM used should be greater than or equal to the cost of the ROM plus associated cost divided by the cost of one gate with its associated overhead. The associated cost per gate must include the cost of the PC card, power supply fans, insertion, inventory, connector, capacitors, check out time, and number of interconnects affecting the reliability. The associated cost per gate can vary from system to system, but $.50 is a figure that many designers would agree upon as reasonable cost. Assume that the cost of the gate plus the associated costs is $.50 and that the associated cost of the ROM is $2.00. The equation now develops as:

$$\text{The number of gates replaced per ROM} > \frac{\$\text{ROM} + \$2.00}{\$.50}$$

As an example, if the cost of the MCM5004 a 64 x 8 PROM is $15 (3 cents/bit) in large quantities, then the number of gates replaced for each PROM should be more than 34. A BCD to binary converter using the MCM5004 replaces approximately 54 gates; this would be a savings of 20 gates or approximately $10.00. Another advantage of using a ROM is that the system size is reduced and reliability should improve as a result of fewer packages.

DESIGN EXAMPLES

Unlike simple combinational circuits, many sequential logic design systems require feedback loops where the outputs are a function of the input signals and the present state of the outputs. There are basically two types of sequential circuit design, synchronous and asynchronous.

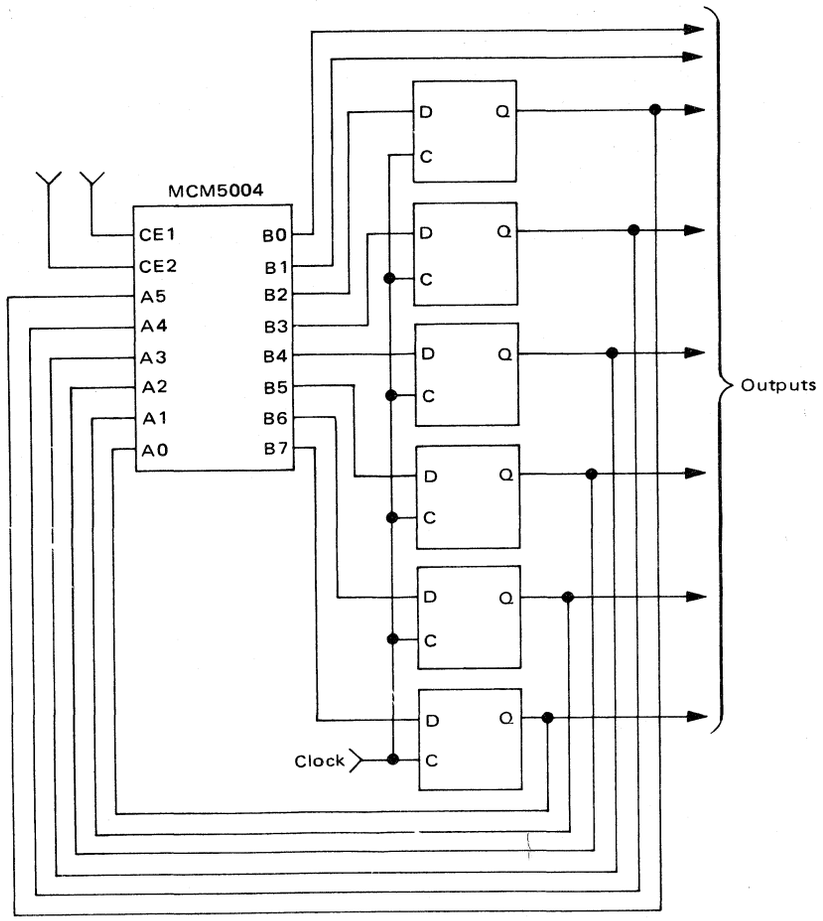
Synchronous sequential design requires data inputs that are synchronized with a clock. When using ROMs, there are two ways of designing synchronous sequential circuits as shown in Figure 1. Clocked storage feedback or direct feedback can be selected. The microprogramming technique basically uses the clocked storage technique with latches in a master slave configuration.

The clocked storage technique shown in Figure 1(a) requires six additional D-flip-flops, but it results in a more efficient usage of the ROM bits. As an example, the asynchronous circuit of Figure 1(a) can be used as a divide by 64 (or less) counter with any sequence of states possible (more than one output can change at a time). The pro-

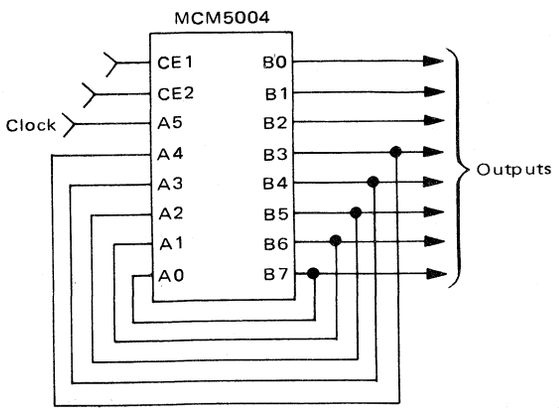
grammable ROM is used to store the next state table.

In contrast, the sequential circuit with direct feedback, as shown in Figure 1(b), can be used as a divide by 16 (or less) counter with three outputs that can be used as decoders or to generate a programmable word sequence. Only one feedback output can change at a time so that race conditions will be avoided in the design.

Synchronous data inputs could be added to the address inputs of the ROMs to produce other types of sequential circuits. Also, more than one ROM can be used where more storage is required. Reference 2 gives an example of the design of a synchronous sequential circuit using clock storage feedback for generating an output whenever four bits, and only four, of a 5 bit serial word are logic ones. The design is straightforward with the PROM storing the next state table.



(a) Synchronous sequential circuit using clocked storage feedback.



(b) Synchronous sequential circuit with direct feedback.

FIGURE 1 – Synchronous Sequential Circuit Types

COUNTER DESIGN – SYNCHRONOUS EXAMPLE

A divide by 16 counter design illustrates the technique for designing a synchronous sequential circuit using direct feedback. The divide by 16 counter is used as a programmable word generator, since only one output can change for each state, a minimum change, reflected code must be used.

The first step is to draw a timing diagram of the word statement, see Figure 2. Note that 32 states exist when both the low and the high states of the clock are included even though this is a divide by 16 counter. The programmable word sequence, B0, B1, and B2, were chosen arbitrarily. The B0 output generates a number of pulses of 1, 2, 3, 3, and, 2 during each 16 bit word. The B1 output generates a pulse synchronized with the last pulse occurring on the B0 otuput. Output B2 generates a pulse width that is 1, 2, 3, and 4 clock periods long in sequence for each 16 bit word.

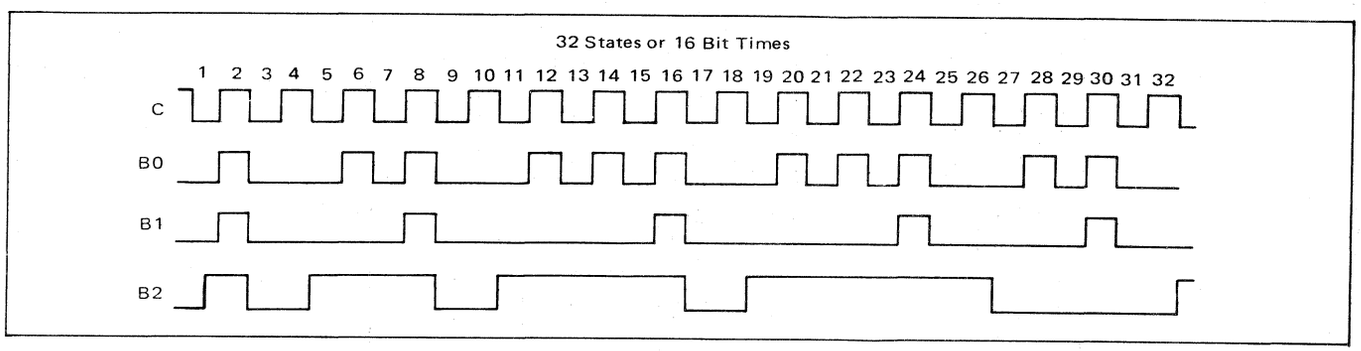


FIGURE 2 – Timing Diagram for Programmable Word Sequences

The next step is to generate the primitive flow table, see Figure 3, using the timing diagram (Figure 2). The circles around the numbers represent stable states, and numbers without circles represent unstable states. If the circuit is in stable state 1 and the clock changes from an "0" to a "1" the circuit goes to unstable state 2 and finally locks up in stable state 2.

Next, the transition map is prepared as shown in Figure 4. A merger diagram is not required in this example, since merging does not occur. A five variable Karnaugh map is used to represent each of the 32 states in the flow table (Figure 3). The five variable y1, y2, y3, y4, and y5 represent the secondaries of the circuit; these are the counter flip-flop outputs in this example. State 1 is placed in the square of the map where y1, y2, y3, y4, and y5 are all in the "1" state. This provides the initialization state when the chip enable is disabled. Since state 1 must go to state 2 when the clock switches from a "0" to a "1", the state 2 is placed in a square of the map where there is a one variable change. If the 2 was placed in a square of the map where there was more than one variable change a race condition could result. Race conditions

C

0	1	Outputs B2	B1	B0
(1)	2	1	0	0
3	(2)	1	1	1
(3)	4	0	0	0
5	(4)	0	0	0
(5)	6	1	0	0
7	(6)	1	0	1
(7)	8	1	0	0
9	(8)	1	1	1
(9)	10	0	0	0
11	(10)	0	0	0
(11)	12	1	0	0
13	(12)	1	0	1
(13)	14	1	0	0
15	(14)	1	0	1
(15)	16	1	0	0
17	(16)	1	1	1

C

0	1	Outputs B2	B1	B0
(17)	18	0	0	0
19	(18)	0	0	0
(19)	20	1	0	0
21	(20)	1	0	1
(21)	22	1	0	0
23	(22)	1	0	1
(23)	24	1	0	0
25	(24)	1	1	1
(25)	26	1	0	0
27	(26)	1	0	0
(27)	28	0	0	0
29	(28)	0	0	1
(29)	30	0	0	0
31	(30)	0	1	1
(31)	32	0	0	0
1	(32)	0	0	0

FIGURE 3 – Primitive Flow Table

y3	y4	y5 \ y1, y2	00	01	11	10
0	0	0	22	21	6	11
0	0	1	21	28	5	12
0	1	1	24	25	8	9
0	1	0	23	27	7	10
1	0	0	19	30	3	14
1	0	1	20	29	4	13
1	1	1	17	32	1	16
1	6	0	18	31	2	15

FIGURE 4 – Transition Map

should be avoided since an indeterminate state could result. Similarly, the rest of the states are placed on the transition map resulting in a minimum change reflection code.

Next, the secondary assignments from the transition map are transferred to the flow table (Figure 5). The timing diagram for the secondary assignments is then drawn, as shown in Figure 6. This shows the counter outputs which are the secondary variables.

Next, the excitation map is prepared, (Figure 7), to define the output excitation variables Y1, Y2, Y3, Y4, Y5 in terms of the input secondary variables and the clock input. A six variable Karnough map is required with the word number located in the upper right hand corner of

Secondary Variables					C		Outputs		
y1	y2	y3	y4	y5	0	1	B2	B1	B0
1	1	1	1	1	(1)	2	1	0	0
1	1	1	1	0	3	(2)	1	1	1
1	1	1	0	0	(3)	4	0	0	0
1	1	1	0	1	5	(4)	0	0	0
1	1	0	0	1	(5)	6	1	0	0
1	1	0	0	0	7	(6)	1	0	1
1	1	0	1	0	(7)	8	1	0	0
1	1	0	1	1	7	(8)	1	1	1
1	0	0	1	1	(9)	10	0	0	0
1	0	0	1	0	11	(10)	0	0	0
1	0	0	0	0	(11)	12	1	0	0
1	0	0	0	1	13	(12)	1	0	1
1	0	1	0	1	(13)	14	1	0	0
1	0	1	0	0	15	(14)	1	0	1
1	0	1	1	0	(15)	15	1	0	0
1	0	1	1	1	17	(16)	1	1	1

					C				
y1	y2	y3	y4	y5	0	1	B2	B1	B0
0	0	1	1	1	(17)	18	0	0	0
0	0	1	1	0	19	(18)	0	0	0
0	0	1	0	0	(19)	20	1	0	0
0	0	1	0	1	21	(20)	1	0	1
0	0	0	0	1	(21)	22	1	0	0
0	0	0	0	0	23	(22)	1	0	1
0	0	0	1	0	(23)	24	1	0	0
0	0	0	1	1	25	(24)	1	1	1
0	1	0	1	1	(25)	26	1	0	0
0	1	0	1	0	27	(26)	1	0	0
0	1	0	0	0	(27)	28	0	0	0
0	1	0	0	1	29	(28)	0	0	1
0	1	1	0	1	(29)	30	0	0	0
0	1	1	0	0	31	(30)	0	1	1
0	1	1	1	0	(31)	32	0	0	0
0	1	1	1	1	1	(32)	0	0	0

FIGURE 5 – Flow Table With Secondary Assignments

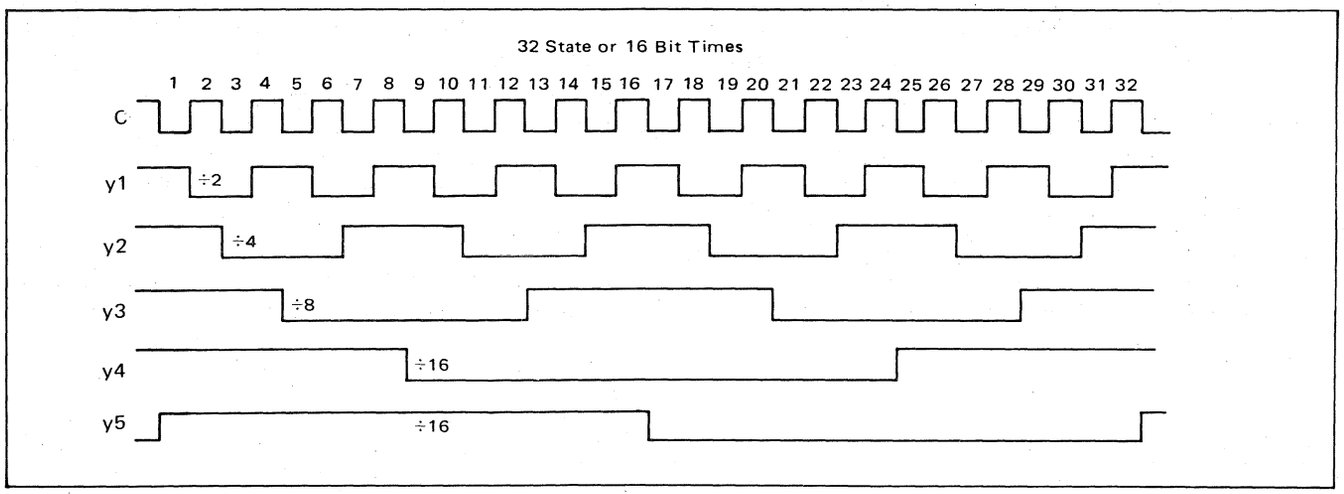


FIGURE 6 – Timing Diagram for Counter Outputs

C = 0

32 y3	16 y4	8 y5	2 y1 / 1 y2: 0 0	0 1	1 1	1 0
0	0	0	00010 0	(01000) 1	11010 3	(10000) 2
0	0	1	(00001) 8	01101 9	(11001) 11	10101 10
0	1	1	01011 24	(01011) 25	10011 27	(10011) 26
0	1	0	(00010) 16	01000 17	(11010) 19	10000 18
1	0	0	(00100) 32	01110 33	(11100) 35	10110 34
1	0	1	00001 40	(01101) 41	11001 43	(10101) 42
1	1	1	(00111) 56	11111 57	(11111) 59	00111 58
1	1	0	00100 48	(01110) 49	11100 51	(10110) 50

C = 1

y1 y2: 0 0	0 1	1 1	1 0
(00000) 4	(01001) 5	(11000) 7	10001 6
00000 12	(01001) 13	11000 15	(10001) 14
(00011) 28	01010 29	(11011) 31	10010 30
00011 20	(01010) 21	11011 23	(10010) 22
00101 36	(01100) 37	11101 39	(10100) 38
(00101) 44	(01100) 45	(11101) 47	10100 46
00110 60	(01111) 61	11110 63	(10111) 62
(00110) 52	01111 53	(11110) 55	10111 54

Y1 Y2 Y3 Y4 Y5

FIGURE 7 – Excitation Map for Counter

each square. The weights of the input variables were arbitrarily chosen, with y2, y1, C, y5, y4, and y3 have the weights of 1,2,4,8,16, and 32 respectively. Each stable state in the map has been circled, and has the same code as the corresponding code of the secondary state. Each unstable state is represented with the code of the next secondary state that will eventually make it a stable state. As an example, word 59 in Figure 7 has a circle around the code 11111 which represents stable state 1 in Figure 5. When the clock switches to the "1" state, word 63 is selected. Word 63 has the code 11110 which is an unstable state, 2, and represents the next transfer to a stable state, which is word 55 representing stable state 2. The rest of the codes are filled in the squares of the excitation map using Figure 5 as a reference.

Next, the output map is prepared, as shown in Figure 8, in order to define the B2, B1, and B0 outputs in terms of the inputs. Each stable state in the map has been circled, and represents the same state as was circled in the excitation map. As an example, word 59 in Figure 8 has a circle around the code 100 which represents stable state 1 in Figure 5. When the clock switches to the "1" state, word 63 is selected which is unstable state 2. Since stable state 2, located at word 55, has an output of 111, a 111 code is placed in the square for word number 63. The last two outputs for this code in word 63 are actually optional conditions and could be either a "0" or a "1". For minimum time delay, the outputs are chosen to switch immediately during the unstable state condition. The rest of the codes are similarly placed in the output map using the flow table of Figure 5 as a reference.

tation at the PROM outputs. As an example, word 0 of Figure 7 and 8 is transferred to word 0 in Figure 9. Word 0 in Figure 7 is 00010 and in Figure 8 it is 100, while Figure 9 is marked with 00010100. Similarly, the rest of the words are transferred to the program sheet.

Figure 10 shows the pin numbers and connections for the MCM5003 used in the counter design, while Figures 11 and 12 show the actual waveforms at clock frequencies of 100 kHz and 1 MHz respectively. Note that the waveforms are the same as the ones shown in Figure 2. There are some possible problem areas that should be understood when using direct feedback. Although only one address input changes at a time, differences in the internal delays in the address decoding of the PROM can cause a hazard condition to exist in which a false word location is momentarily selected. In the MCM5003, the false word location will be selected for approximately 10 ns or 1 gate delay during the hazard condition.

Programmable ROMs using ECL logic internally for the decoding such as the MCM10149, minimize the hazard condition. The ECL logic has approximately equal delays through the OR and NOR outputs of a gate, while TTL logic requires an extra gate (and delay) to perform the complement function. There are ways around the hazard condition by sacrificing speed. The decoding glitches are evident in Figure 12 on the y1 output when in the low state. The decoding glitches are approximately 0.5 volts and are a possible problem only when the output is in the low state. The amplitude of the decoding glitches varies from device to device. If the amplitude is large enough to be detected when fed back at the input, the PROM could

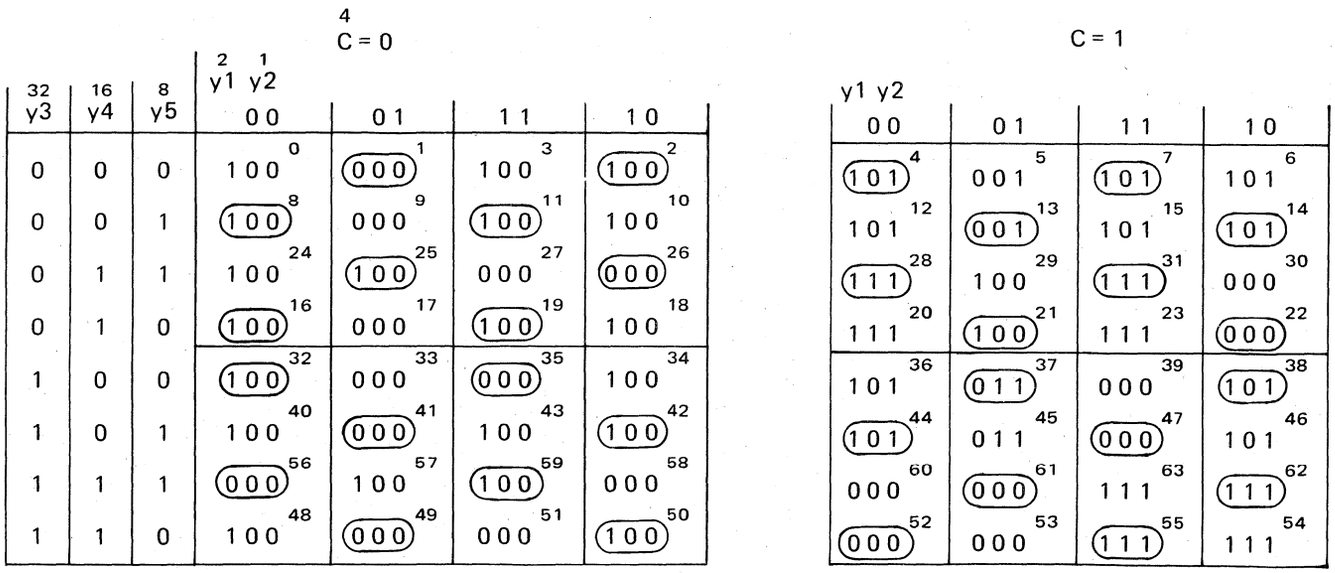


FIGURE 8 – Output Map for Counter Defining the Outputs B2, B1, and B0

Finally, the codes in the excitation and output maps are transferred to the program sheet truth table (Figure 9) for the programmable ROM, the MCM5003. The excitation variable Y1, Y2, Y3, Y4, and Y5 are chosen to represent the B7, B6, B5, B4, and B3 PROM outputs respectively. The B2, B1, and B0 outputs are represented by the same no-

switch to an improper state. The solution to the problem for the MCM5003 is the addition of capacitance, C_L, for the outputs that are fed back as shown in Figure 10. If a 510 ohm load resistor, R_L, is used, then a 100 pF load capacitor, C_L, should be used. If R_L = 3.3 k, a load capacitor is not required for the typical device although

No.	Address						Data							
	A5	A4	A3	A2	A1	A0	B7	B6	B5	B4	B3	B2	B1	B0
	y3	y4	y5	C	y1	y2	Y1	Y2	Y3	Y4	Y5			
0	0	0	0	0	0	0				X		X		
1	0	0	0	0	0	1		X						
2	0	0	0	0	1	0	X					X		
3	0	0	0	0	1	1	X	X		X		X		
4	0	0	0	1	0	0						X		X
5	0	0	0	1	0	1		X			X			X
6	0	0	0	1	1	0	X				X	X		X
7	0	0	0	1	1	1	X	X				X		X
8	0	0	1	0	0	0					X	X		
9	0	0	1	0	0	1		X	X		X			
10	0	0	1	0	1	0	X		X		X	X		
11	0	0	1	0	1	1	X	X			X	X		
12	0	0	1	1	0	0						X		X
13	0	0	1	1	0	1		X			X			X
14	0	0	1	1	1	0	X				X	X		X
15	0	0	1	1	1	1	X	X				X		X
16	0	1	0	0	0	0				X		X		
17	0	1	0	0	0	1		X						
18	0	1	0	0	1	0	X					X		
19	0	1	0	0	1	1	X	X		X		X		
20	0	1	0	1	0	0				X	X	X	X	X
21	0	1	0	1	0	1		X		X		X		
22	0	1	0	1	1	0	X			X				
23	0	1	0	1	1	1	X	X		X	X	X	X	X
24	0	1	1	0	0	0		X		X	X	X		
25	0	1	1	0	0	1		X		X	X	X		
26	0	1	1	0	1	0	X			X	X			
27	0	1	1	0	1	1	X			X	X			
28	0	1	1	1	0	0				X	X	X	X	X
29	0	1	1	1	0	1		X		X		X		
30	0	1	1	1	1	0	X			X				
31	0	1	1	1	1	1	X	X		X	X	X	X	X
32	1	0	0	0	0	0			X			X		
33	1	0	0	0	0	1		X	X	X				
34	1	0	0	0	1	0	X		X	X		X		
35	1	0	0	0	1	1	X	X	X					
36	1	0	0	1	0	0			X		X	X		X
37	1	0	0	1	0	1		X	X				X	X
38	1	0	0	1	1	0	X		X			X		X
39	1	0	0	1	1	1	X	X	X		X			
40	1	0	1	0	0	0					X	X		
41	1	0	1	0	0	1		X	X		X			
42	1	0	1	0	1	0	X		X		X	X		
43	1	0	1	0	1	1	X	X			X	X		
44	1	0	1	1	0	0			X		X	X		X
45	1	0	1	1	0	1		X	X				X	X
46	1	0	1	1	1	0	X		X			X		X
47	1	0	1	1	1	1	X	X	X		X			
48	1	1	0	0	0	0			X			X		
49	1	1	0	0	0	1		X	X	X				
50	1	1	0	0	1	0	X		X	X		X		
51	1	1	0	0	1	1	X	X	X					
52	1	1	0	1	0	0			X	X				
53	1	1	0	1	0	1		X	X	X	X			
54	1	1	0	1	1	0	X		X	X	X	X	X	X
55	1	1	0	1	1	1	X	X	X	X		X	X	X
56	1	1	1	0	0	0			X	X	X			
57	1	1	1	0	0	1	X	X	X	X	X	X		
58	1	1	1	0	1	0			X	X	X			
59	1	1	1	0	1	1	X	X	X	X	X	X		
60	1	1	1	1	0	0			X	X				
61	1	1	1	1	0	1		X	X	X	X			
62	1	1	1	1	1	0	X		X	X	X	X	X	X
63	1	1	1	1	1	1	X	X	X	X		X	X	X

"X" represents a bit to be programmed to a logic "1".

FIGURE 9 – MCM5003 Program Sheet for Counter

25 to 50 pF must be used for some devices.

If the MCM5004 (with a 2 k internal collector resistor) is used instead of the MCM5003, a 50 pF capacitor should be used. The RC time constant is enough to squelch the decoding glitches with a slight sacrifice in speed. The decoding glitches are smallest in amplitude at higher V_{CC} voltages and at higher temperatures. The synchronous sequential circuit using clocked storage feedback (see Figure 1(a) does not have the problem of squelching the decoding glitches, since the storage devices see only the information that is present when the clock changes from a "0" to a "1".

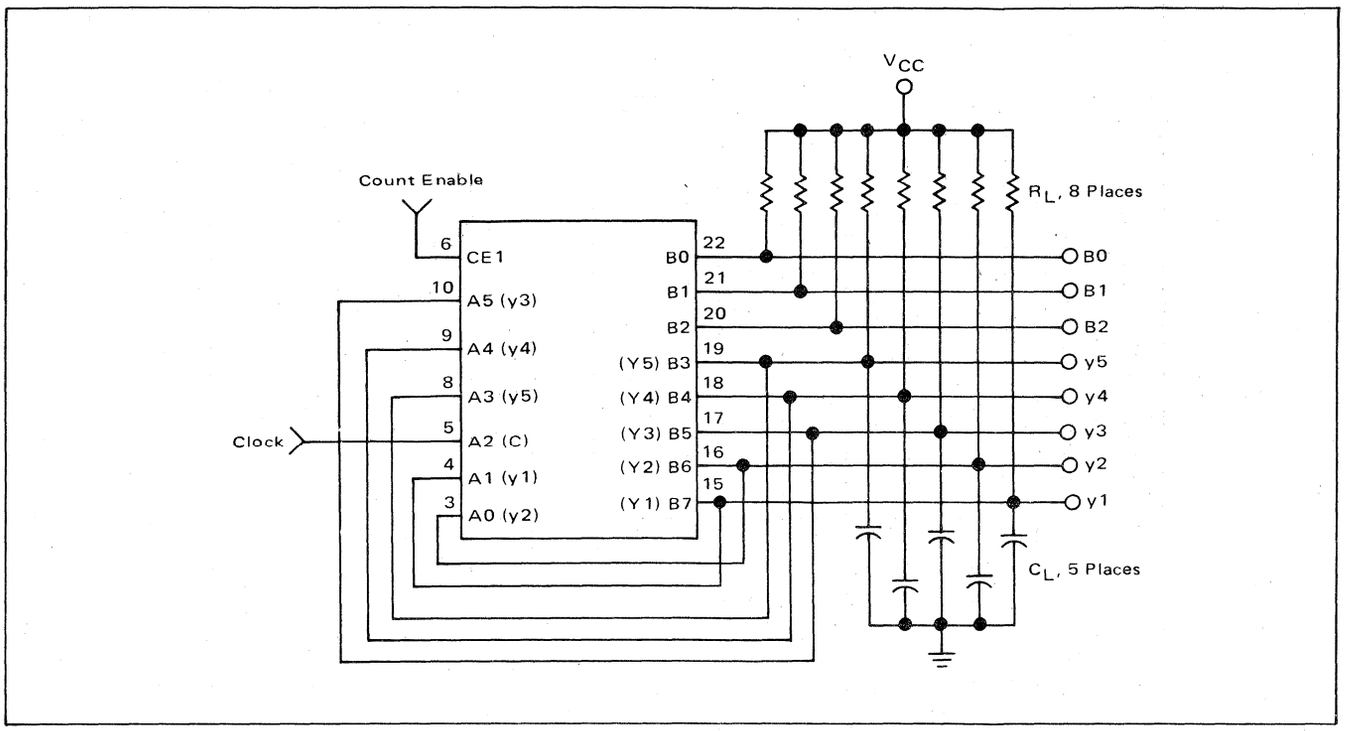
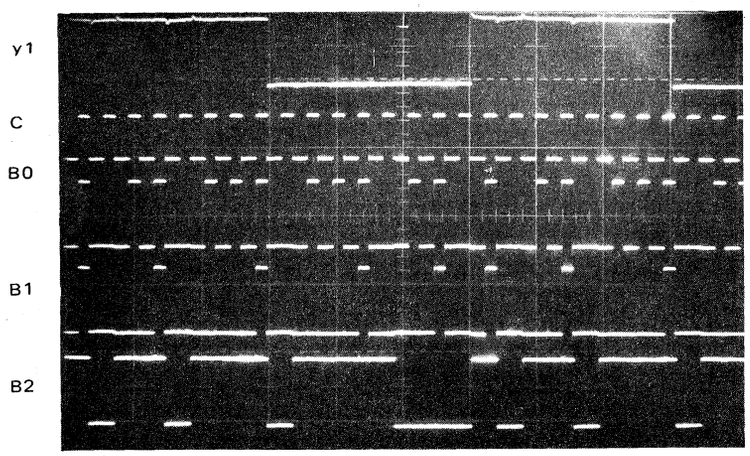


FIGURE 10 – PROM Connections for Counter Design with Direct Feedback



Vertical Scale = 5 V/cm
Horizontal Scale = 25 μs/cm

R_L = 3.3 kΩ
C_L = 10 pF
Stray Capacitance

FIGURE 11 – Counter and Output Waveforms of Figure 10 for Clock Frequency of 100 kHz

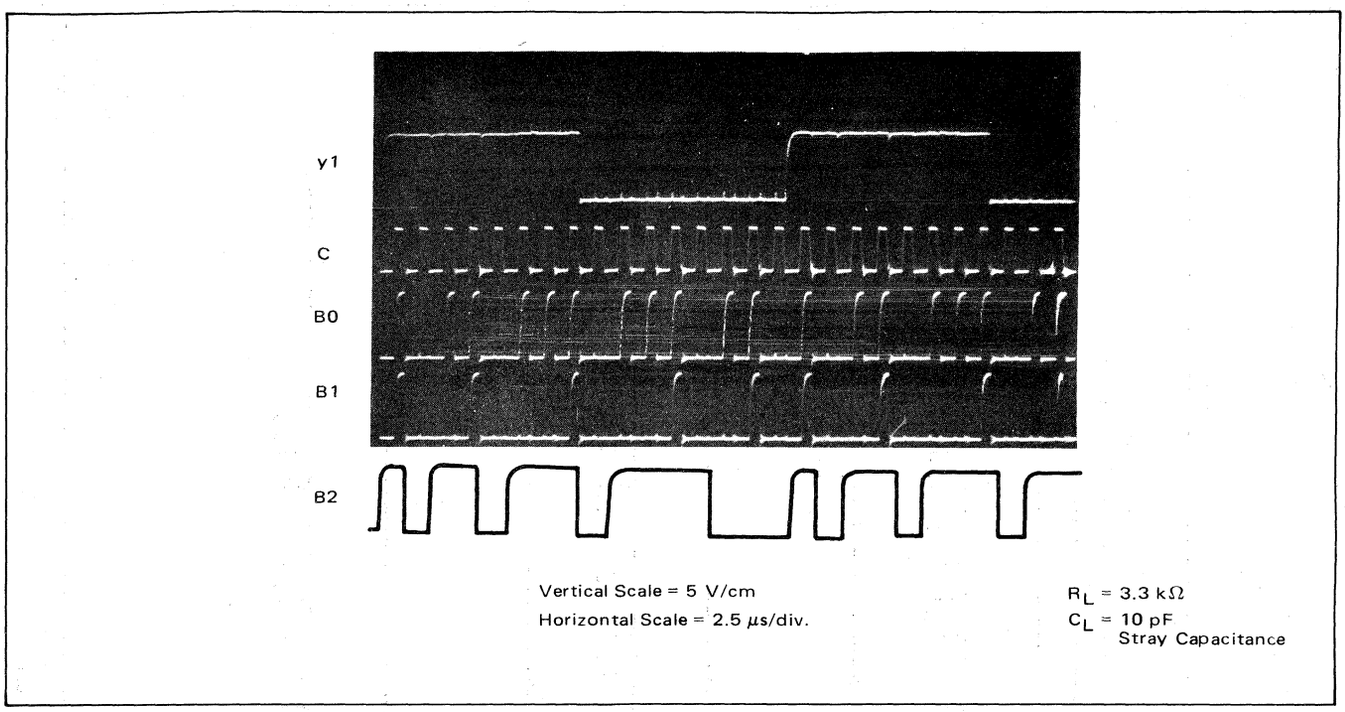


FIGURE 12 – Counter and Output Waveforms of Figure 10 for Clock Frequency of 1 MHz

CLOCK SYNCHRONIZER AND CONTROL DESIGN – COMBINATION ASYNCHRONOUS AND SYNCHRONOUS EXAMPLE

A combination asynchronous and synchronous sequential circuit can be built, using the MCM5003, in the design of a clock synchronizer and control. This circuit is very useful in the control section of a computer for timing and control. The synchronous inputs are the clock and the start/stop control, while the asynchronous input is the start button. The purpose of the circuit is to synchronize the clocking and timing signals for the control section.

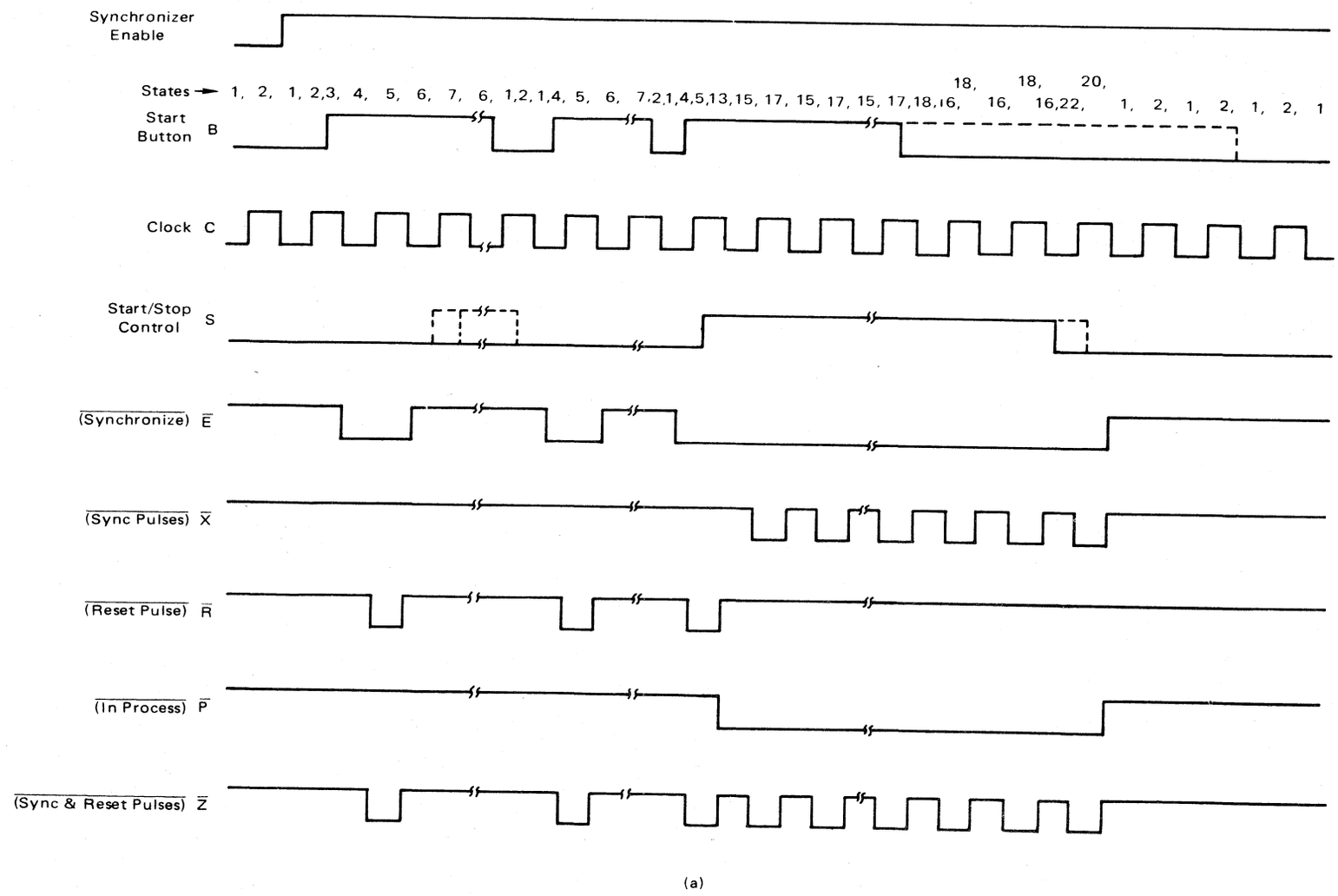
Figure 13(a) and (b) shows the timing diagram for the synchronizer circuit. The synchronizer enable initializes the circuit and can be implemented directly using the chip enable of the PROM. The leading edge of the signal from the start button switch initiates the timing sequence. The start/stop control in Figure 13(a) is the output of a latch or clocked flip-flop: it is normally synchronized with the clock. One synchronized reset pulse occurs whenever the start button is depressed. If the start/stop control goes "high" just after the reset pulse is initiated, sync pulses will occur until the start/stop control goes low. Note that the trailing edge of the start button signal has no effect on the timing signals. The in-process signal occurs only if sync pulses occur. The synchronize signal engulfs all reset and sync pulses. Another output is available having the reset and sync pulses ORed together.

Figure 13(b) shows the timing diagram under a different set of conditions. Here, the start/stop control signal is normally "high" and is the output of a decoder or gate circuit. One reset and any number of sync pulses occur until the start/stop signal goes "low" after having decoded a stop condition. The synchronizer enable can be used to start the sequence over if the start button signal is "high". If the start button signal is "low" when the synchronizer enable is brought "low", the sequence of timing signals immediately halts.

The design of the synchronizer circuit is a little more complicated than the counter example. Additional design information will be clarified in the following discussion to enable the reader to design other sequential circuits using PROM's.

From the timing diagram and an understanding of the design goals of the circuit, a primitive flow table is derived as shown in Figure 14. Each row in the table can have one, and only one non-redundant stable state. The states for the table, with the numbering being chosen arbitrarily, are derived from the timing diagram. Some of the states are listed in sequence in the timing diagram of Figure 13(a). Only one stable state can exist in each column for a given output. All possible combination of inputs must be present in the table with the don't care conditions being represented by a dash. Note that the outputs on the table are the complement of the timing diagram. The reason for the complement output signals on the timing diagram was that the initialization state occurs when all outputs are high.

The next step is to generate the merged flow table, shown in Figure 15, by combining rows that contain like state numbers for all combinations of inputs. The outputs of the rows do not have to be the same when merging two rows; although in this example, the merged rows did have the same outputs. The reason for merging is to reduce the number of states required. In this example, the number of states is reduced from 24 to six.



(a)

FIGURE 13a – Timing Diagram For Synchronizer Circuit

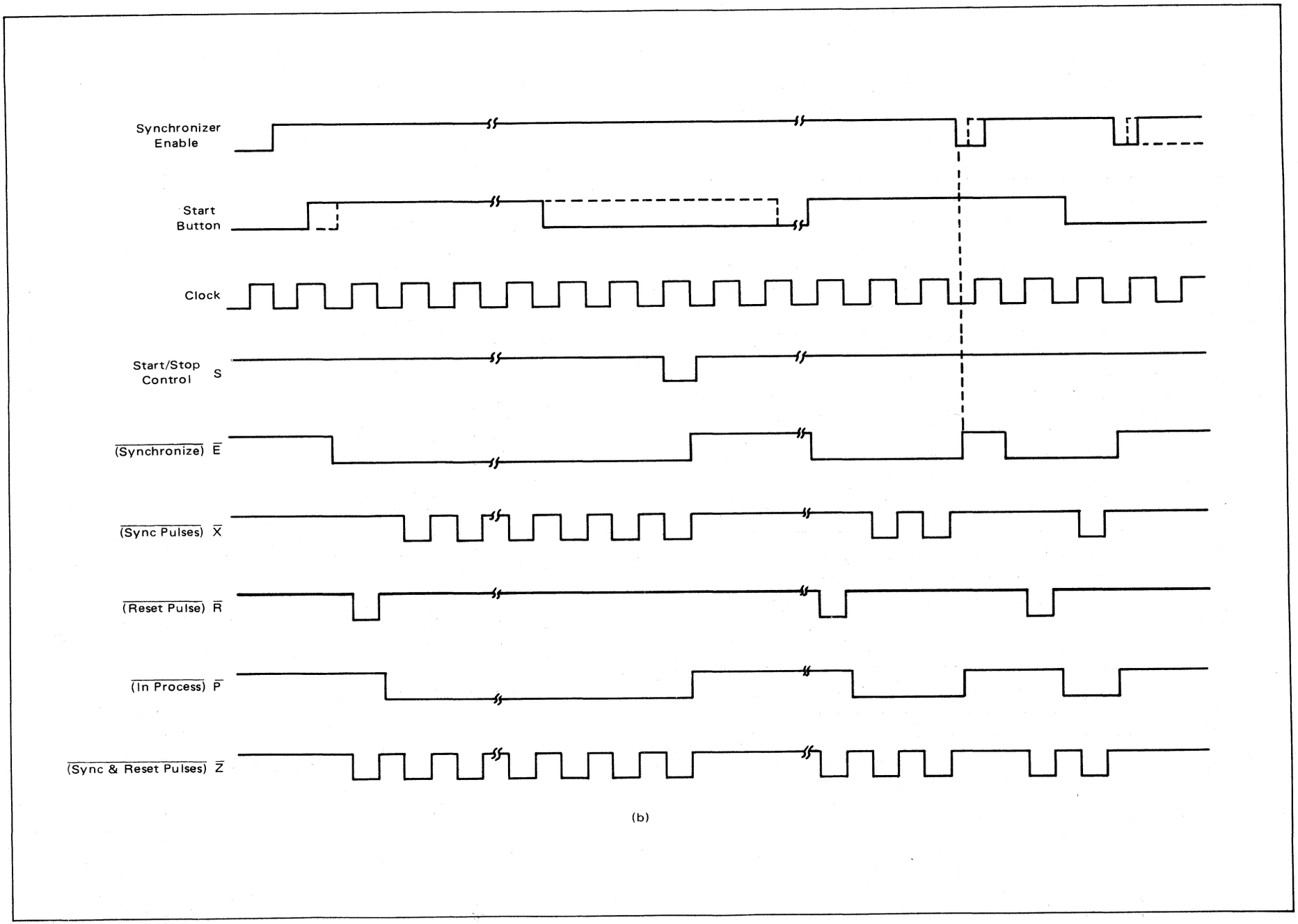


FIGURE 13b – Timing Diagram For Synchronizer Circuit

SCB	Inputs							Outputs				
000	001	011	010	100	101	111	110	Z	P	R	X	E
(1)	4	–	2	9	12	11	10	0	0	0	0	0
1	4	3	(2)	9	12	11	10	0	0	0	0	0
1	4	(3)	2	9	12	11	10	0	0	0	0	0
1	(4)	5	2	9	12	13	10	0	0	0	0	1
1	6	(5)	8	16	15	13	14	1	0	1	0	1
1	(6)	7	2	9	23	24	10	0	0	0	0	0
1	6	(7)	2	9	23	24	10	0	0	0	0	0
1	6	5	(8)	16	15	13	14	1	0	1	0	1
1	4	3	2	(9)	12	11	10	0	0	0	0	0
1	4	3	2	9	12	11	(10)	0	0	0	0	0
1	4	3	2	9	12	(11)	10	0	0	0	0	0
1	4	5	–	9	(12)	13	–	0	0	0	0	1
–	–	5	8	16	15	(13)	14	1	0	1	0	1
–	–	5	8	16	15	13	(14)	1	0	1	0	1
22	21	19	20	16	(15)	17	18	0	1	0	0	1
22	21	19	20	(16)	15	17	18	0	1	0	0	1
1	6	19	20	16	15	(17)	18	1	1	0	1	1
1	6	19	20	16	15	17	(18)	1	1	0	1	1
1	6	(19)	20	–	–	17	18	1	1	0	1	1
1	6	19	(20)	–	–	17	18	1	1	0	1	1
22	(21)	19	20	16	15	–	–	0	1	0	0	1
(22)	21	19	20	16	15	–	–	0	1	0	0	1
1	6	7	2	9	(23)	24	10	0	0	0	0	0
1	6	7	2	9	23	(24)	10	0	0	0	0	0

FIGURE 14 – Primitive Flow Table for Synchronizer

	SCB	Inputs								Outputs				
	000	001	011	010	100	101	111	110	Z	P	R	X	E
a	(1)	4	3	(2)	(9)	12	(11)	(10)	0	0	0	0	0
b	1	(4)	5	2	9	(12)	13	10	0	0	0	0	1
c	1	(6)	(7)	2	9	(23)	(24)	10	0	0	0	0	0
d	1	6	(5)	(8)	16	15	(13)	(14)	1	0	1	0	1
e	(22)	(21)	19	20	(16)	(15)	17	18	0	1	0	0	1
f	1	6	(19)	(20)	16	15	(17)	(18)	1	1	0	1	1

FIGURE 15 – Merged Flow Table for Synchronizer

The next step is to prepare a transition map in order to assign the secondary variables. Each of the rows are arbitrarily assigned a letter in Figure 15 to represent a secondary state. A three variable map in Figure 16 is required to define the six states. The assigning of states in the map is the same as in the previous counter example by assigning states with a one variable change. However, in this example cycles are required in order that more than one secondary variable doesn't change at one time. Two spare secondary states, G and H, are utilized to avoid critical races. The cycles are shown with arrows, indicating the movement from one unstable state to another unstable state. The absence of an arrow at an unstable state indicates a direct movement to a stable state.

	Y1 Y2			
Y3	00	01	11	10
0	c	e	f	h
1	d	G	a	b

FIGURE 16 – Transition Map

A cycle from one unstable state to another unstable state with the same number cannot be done for all cases, even though there is a one variable change. This condition exists, when in the middle of a cycle, another input changes causing the circuit to go to the wrong state. As an example, assume the circuit is in stable state six (see Figure 17) and B goes to a logic "0". A cycle must occur since Y1, Y2, and Y3 are presently 000 and must go to stable state 1 with the secondaries changing to 111, a three variable change. Assume that a cycle was chosen instead from unstable state 1 in row C to unstable state 1 in row D. When the circuit reached row D, a change in the clock from a "0" to a "1" would cause the circuit to end up in stable state 8 instead of state 2. By making the cycle from C to H, (Figure 1), a change in the clock would cause the circuit to go to the correct state, 2. After the cycles and transition of states have been assigned in Figures 15 and 16, the secondary assignments are assigned to merged flow table (Figure 17).

The excitation and output maps are filled out in Figures 18 and 19, using the same method that was described previously in the counter design example. In the output map, the complement of the outputs in Figure 17 were used to arrive with the correct waveforms as shown in Figure 13.

Finally, the program sheet for the programmable ROM, the MCM5004 is filled out with the codes from the excitation and output maps. The corresponding code in the word number of the map is transferred to the same word number in the program sheet.

The circuit connections are shown in Figure 33. The 50 pF capacitors are used to squelch the decoding glitches from the outputs that are fed back to the inputs.

	Secondary Variables			SCB	Inputs							Outputs				
	y1	y2	y3	000	001	011	010	100	101	111	110	Z	P	R	X	E
a	1	1	1	(1)	4	(3)	(2)	(9)	12	(11)	(10)	0	0	0	0	0
b	1	0	1	1	(4)	5	2	9	(12)	13	10	0	0	0	0	1
c	0	0	0	1	(6)	(7)	2	9	(23)	(24)	10	0	0	0	0	0
d	0	0	1	1	6	(5)	(8)	16	15	(13)	(14)	1	0	1	0	1
e	0	1	0	(22)	(21)	19	20	(16)	(15)	17	18	0	1	0	0	1
f	1	1	0	1	6	(19)	(20)	16	15	(17)	(18)	1	1	0	1	1
G	0	1	1	1	–	–	–	16	15	–	–					
h	1	0	0	1	6	–	2	9	23	–	10					

FIGURE 17 – Merged Flow Table With Secondary Assignments

	Secondary Variables			SCB	Inputs						
	y1	y2	y3	000	001	011	010	100	101	111	110
c	0	0	0	100 0	(000) 1	(000) 3	100 2	100 4	(000) 5	(000) 7	100 6
d	0	0	1	011 8	000 9	(001) 11	(001) 10	011 12	011 13	(001) 15	(001) 14
G	0	1	1	111 24	– 25	– 27	– 26	010 28	010 29	– 31	– 30
e	0	1	0	(010) 16	(010) 17	110 19	110 18	(010) 20	(010) 21	110 23	110 22
h	1	0	0	101 32	000 33	– 35	101 34	101 36	000 37	– 39	101 38
b	1	0	1	111 40	(101) 41	001 43	111 42	111 44	(101) 45	001 47	111 46
a	1	1	1	(111) 56	101 57	(111) 59	(111) 58	(111) 60	101 61	(111) 63	111 62
f	1	1	0	111 48	100 49	(110) 51	(110) 50	010 52	010 53	(110) 55	(110) 54

FIGURE 18 – Excitation Map for Y1, Y2, Y3

	Secondary Variables			SCB	Inputs							
	y1	y2	y3	000	001	011	010	100	101	111	110
c	0	0	0	11111 0	(11111) 1	(11111) 3	11111 2	11111 4	(11111) 5	(11111) 7	11111 6
d	0	0	1	11111 8	11110 9	(01010) 11	(01010) 10	11110 12	11110 13	(01010) 15	(01010) 14
G	0	1	1	11111 24	– 25	– 27	– 26	11110 28	11110 29	– 31	– 30
e	0	1	0	(10110) 16	(10110) 17	-01-0 19	-01-0 18	(10110) 20	(10110) 21	-01-0 23	-01-0 22
h	1	0	0	11111 32	10110 33	– 35	11111 34	11111 36	10110 37	– 39	11111 38
b	1	0	1	11111 40	(11110) 41	-1-10 43	11111 42	11111 44	(11110) 45	-1-10 47	11111 46
a	1	1	1	(11111) 56	11110 57	(11111) 59	(11111) 58	(11111) 60	11110 61	(11111) 63	(11111) 62
f	1	1	0	00100 48	00100 49	(00100) 51	(00100) 50	-01-0 52	-01-0 53	(00100) 55	(00100) 54

$\overline{Z}$ $\overline{P}$ $\overline{R}$ $\overline{X}$ $\overline{E}$

FIGURE 19 – Output Map

Address							Data							
No.	A5	A4	A3	A2	A1	A0	B7	B6	B5	B4	B3	B2	B1	B0
	y1	y2	y3	S	C	B	y1	y2	y3	$\overline{Z}$	$\overline{P}$	$\overline{R}$	$\overline{X}$	$\overline{E}$
0	0	0	0	0	0	0	X			X	X	X	X	X
1	0	0	0	0	0	1				X	X	X	X	X
2	0	0	0	0	1	0	X			X	X	X	X	X
3	0	0	0	0	1	1				X	X	X	X	X
4	0	0	0	1	0	0	X			X	X	X	X	X
5	0	0	0	1	0	1				X	X	X	X	X
6	0	0	0	1	1	0	X			X	X	X	X	X
7	0	0	0	1	1	1				X	X	X	X	X
8	0	0	1	0	0	0		X	X	X	X	X	X	X
9	0	0	1	0	0	1				X	X	X	X	
10	0	0	1	0	1	0			X		X		X	
11	0	0	1	0	1	1			X		X		X	
12	0	0	1	1	0	0		X	X	X	X	X	X	
13	0	0	1	1	0	1		X	X	X	X	X	X	
14	0	0	1	1	1	0			X		X		X	
15	0	0	1	1	1	1			X		X		X	
16	0	1	0	0	0	0		X		X		X	X	
17	0	1	0	0	0	1		X		X		X	X	
18	0	1	0	0	1	0	X	X				X		
19	0	1	0	0	1	1	X	X				X		
20	0	1	0	1	0	0		X		X		X	X	
21	0	1	0	1	0	1		X		X		X	X	
22	0	1	0	1	1	0	X	X				X		
23	0	1	0	1	1	1	X	X				X		
24	0	1	1	0	0	0	X	X	X	X	X	X	X	X
25	0	1	1	0	0	1								
26	0	1	1	0	1	0								
27	0	1	1	0	1	1								
28	0	1	1	1	0	0		X		X	X	X	X	
29	0	1	1	1	0	1		X		X	X	X	X	
30	0	1	1	1	1	0								
31	0	1	1	1	1	1								
32	1	0	0	0	0	0	X		X	X	X	X	X	X
33	1	0	0	0	0	1				X		X	X	
34	1	0	0	0	1	0	X		X	X	X	X	X	X
35	1	0	0	0	1	1								
36	1	0	0	1	0	0	X		X	X	X	X	X	X
37	1	0	0	1	0	1				X		X	X	
38	1	0	0	1	1	0	X		X	X	X	X	X	X
39	1	0	0	1	1	1								
40	1	0	1	0	0	0	X	X	X	X	X	X	X	

FIGURE 20 – MCM5004 Program Sheet for Clock Synchronizer and Control

	Address						Data							
No.	A5	A4	A3	A2	A1	A0	B7	B6	B5	B4	B3	B2	B1	B0
	y1	y2	y3	S	C	B	y1	y2	y3	$\overline{Z}$	$\overline{P}$	$\overline{R}$	$\overline{X}$	$\overline{E}$
41	1	0	1	0	0	1	X		X	X	X	X	X	
42	1	0	1	0	1	0	X	X	X	X	X	X	X	X
43	1	0	1	0	1	1			X		X		X	
44	1	0	1	1	0	0	X	X	X	X	X	X	X	X
45	1	0	1	1	0	1	X		X	X	X	X	X	
46	1	0	1	1	1	0	X	X	X	X	X	X	X	X
47	1	0	1	1	1	1			X		X		X	
48	1	1	0	0	0	0	X	X	X			X		
49	1	1	0	0	0	1	X					X		
50	1	1	0	0	1	0	X	X				X		
51	1	1	0	0	1	1	X	X				X		
52	1	1	0	1	0	0		X				X		
53	1	1	0	1	0	1		X				X		
54	1	1	0	1	1	0	X	X				X		
55	1	1	0	1	1	1	X	X				X		
56	1	1	1	0	0	0	X	X	X	X	X	X	X	X
57	1	1	1	0	0	1	X		X	X	X	X	X	
58	1	1	1	0	1	0	X	X	X	X	X	X	X	X
59	1	1	1	0	1	1	X	X	X	X	X	X	X	X
60	1	1	1	1	0	0	X	X	X	X	X	X	X	X
61	1	1	1	1	0	1	X		X	X	X	X	X	
62	1	1	1	1	1	0	X	X	X	X	X	X	X	X
63	1	1	1	1	1	1	X	X	X	X	X	X	X	X

"X" represents a bit to be programmed to a logic "1".

FIGURE 20 – MCM5004 Program Sheet for Clock Synchronizer and Control

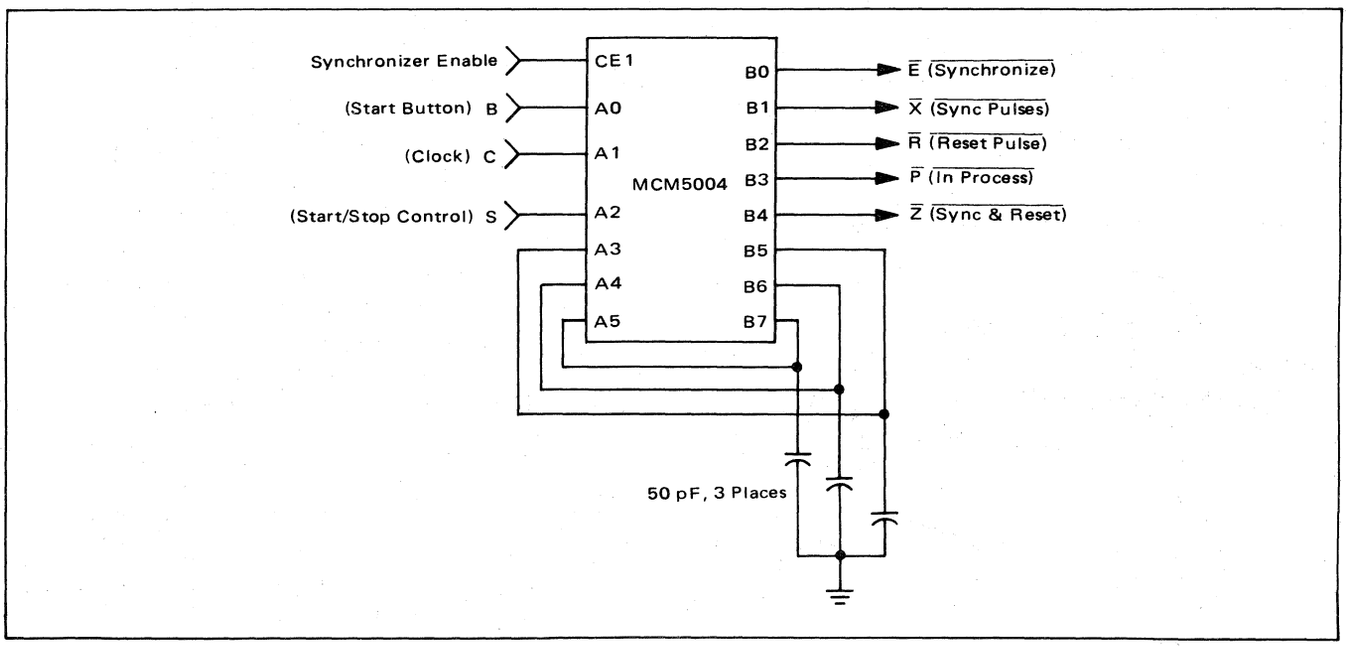


FIGURE 21 – Clock Synchronizer and Control

DIGITAL PULSE SUBTRACTOR – ASYNCHRONOUS EXAMPLE

An asynchronous circuit, by definition, has no clock to synchronize the input signals. A digital pulse subtractor will be designed to show the potential problem areas that must be considered when designing asynchronous circuits. One potential problem arises because of the propagation delay differences between positive and negative going output edges. The faster access time when switching to the low state (in part due to the resistor pullup outputs) need not cause problems when extra design steps are implemented. In this example, the use of another design tool, the differential mode state table, will also be shown.

A timing diagram for a pulse subtractor, useful in communication and peripheral areas such as data recovery in disc and tape systems, is shown in Figure 22. The detection of a pulse is done on the leading edge while the trailing edge and the pulse width have no effect. The output pulse

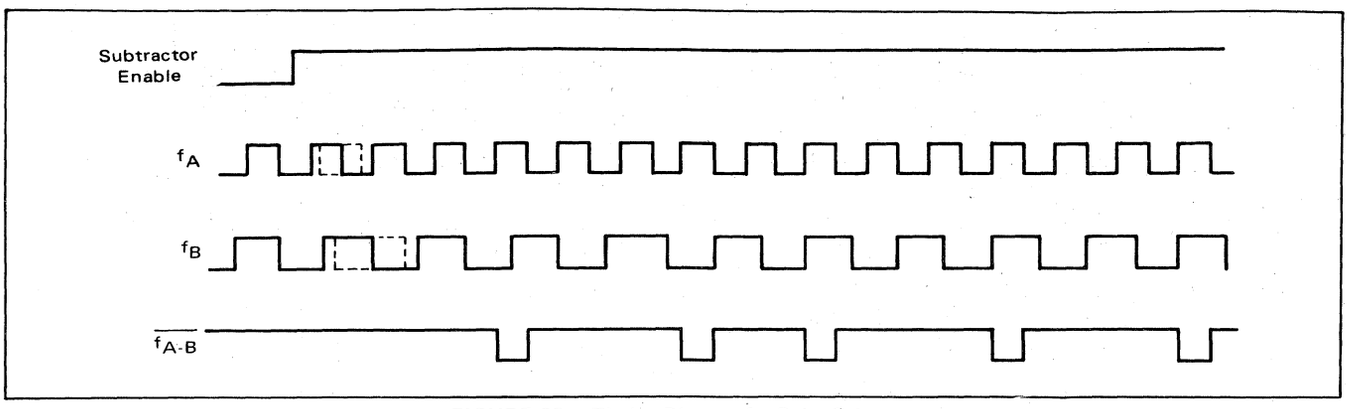


FIGURE 22 – Timing Diagram for Pulse Subtractor

width (negative going) is equal to the pulse width of highest frequency. If Frequency B was higher than Frequency A, however, the output would stay in the logic "1" state and no subtraction would take place.

A differential mode state diagram for the pulse subtractor is shown in Figure 23, and is derived from the timing diagram and word statement of the problem. Reference 3 describes the diagram in more detail. The diagram shows that four states, S0 thru S3, are needed to define the problem. The output logic level is under the slashed line of each circled state. Each line with an arrow indicates a transition of one input and resulting state change, and it is labeled with the type of transition. For instance, a ΔA refers to the transition of input A from a logic "0" to a logic "1". A ∇A refers to the transition of input A from a logic "1" to a logic "0".

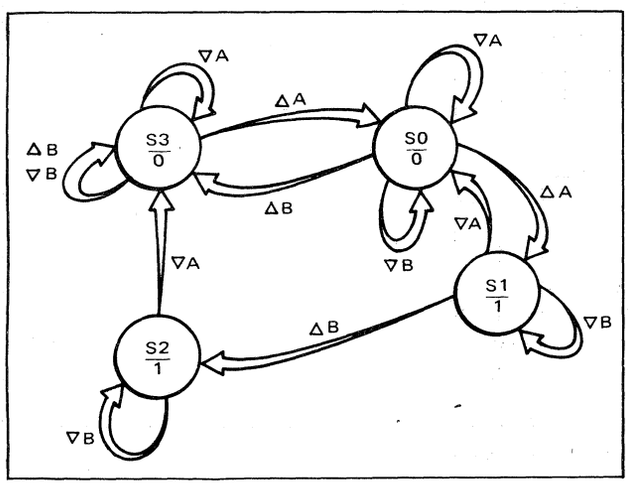


FIGURE 23 – Differential Mode State Diagram for Pulse Subtractor

The primitive flow table is derived as previously shown, but using Figure 23 as a reference. The states from the differential mode state diagram are labeled for reference next to each row of the primitive flow table.

The merged flow table with secondary assignments is shown in Figure 24 using methods previously discussed. States M and N were assigned to prevent lock-up conditions.

	Inputs				Output
	A B				
State	0 0	0 1	1 1	1 0	f_{A-B}
S0	(1)	3	–	2	0
S1	1	3	5	(2)	1
S3	8	(3)	4	6	0
S0	1	9	(4)	6	0
S2	8	3	(5)	7	1
S0	1	3	10	(6)	0
S2	8	3	5	(7)	1
S3	(8)	3	10	6	0
S0	1	(9)	11	2	0
S3	8	3	(10)	12	0
S1	1	9	(11)	12	1
S3	8	3	10	(12)	0

FIGURE 24 – Primitive Flow Table

Figure 25 shows the waveforms observed in the lab for the programmed PROM using the flow table of Figure 25. A proolem area is illustrated when frequencies A and B are the same and the leading edge of B occurs slightly ahead of the leading edge of A. The resulting output fre-

					A B				
y1	y2	y3	y4		0 0	0 1	1 1	1 0	f_{A-B}
0	1	0	1	a	(1)	3	5	(2)	0,1
1	1	0	1	b	8	(3)	4	6	0
1	0	0	1	c	1	9	(4)	6	0
0	1	0	0	d	8	3	(5)	(7)	1
0	1	1	1	e	1	3	10	(6)	0
1	1	1	1	f	(8)	3	10	6	0
0	0	0	1	G	1	(9)	(11)	2	0,1
0	1	1	0	H	8	3	10	(12)	0
1	0	1	1	I	1	–	–	6	0
0	0	1	1	J	1	–	–	6	0
1	1	1	0	K	8	3	–	–	0
1	1	0	0	L	8	3	–	–	0
0	0	0	0	M	1	9	10	12	0
0	0	1	0	N	1	9	10	12	0

FIGURE 25 – Merged Flow Table with Secondary Assignments

quency is one half the input frequency, but should be zero frequency. The reason is due to the faster access time of the MCM5004 when the output is switching to the low state, t_{pdL}, compared to switching to the high state, t_{pdH}. One reason for the difference is the much larger RC time constant when switching to the high state. When the circuit is in state 8 and input B changes to the high state, there is enough time for the circuit to go to state 3 before input A goes high and the circuit finally goes to state 4. The reason there is enough time to get into state 3 is that the secondary variable, Y3, switches from a logic "1" to a logic "0"

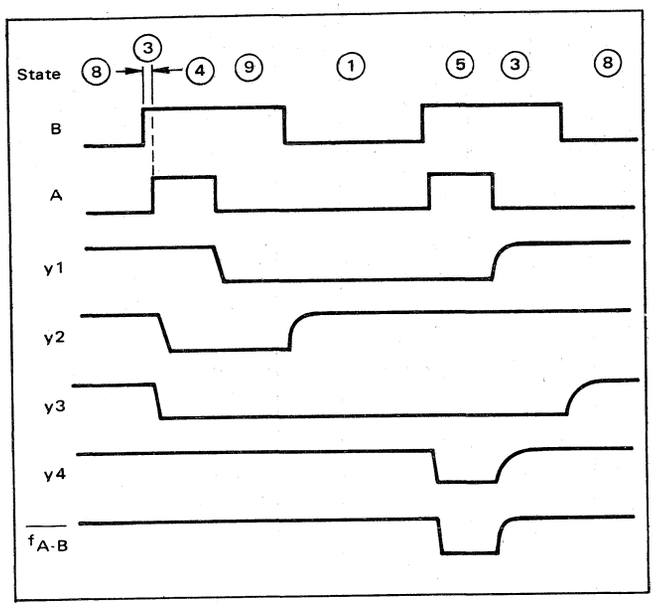


FIGURE 26 – Waveforms for Subtractor Design

The problem occurs in the next cycle when the circuit is in state 1 and input B again changes first, but the circuit ends up in state 5 causing an output. The circuit should have gone to state 3 first and then to state 4 when input A changes to the high state. The reason that the circuit did not get into state 3 is that the secondary variable, Y1, has to change from a logic "0" to a logic "1". Due to the longer propagation delay when the output goes to the high state, input A changed before the circuit gets into state 3, it looks to the circuit like both A and B changed simultaneously causing the change to state 5. This problem can be solved by going back to the primitive flow table in Figure 24, and changing the don't care condition in the first row to unstable state 4. This results in the revised merged flow table with secondary assignments in Figure 27. States K, L, and M were added to prevent lock-up of the circuit. The excitation and outputs maps are shown in Figure 28. Note that the output is complemented in the output map to agree with the timing diagram. The complement output aids the initialization of the circuit, since the chip enable is used to make the output go to the high state when initialized. The program sheet shown in Figure 29 was derived from Figure 28. The circuit connections are shown in Figure 30. The circuit as tested in the lab verified that the previous problem did not occur. The

					AB				
y1	y2	y3	y4		00	01	11	10	f_{A-B}
0	1	0	1	a1	(1)	3	4	2	0
0	1	0	0	a2	1	3	5	(2)	1
1	1	0	1	b	8	(3)	4	6	0
1	0	0	1	c	1	9	(4)	6	0
1	1	0	0	d	8	3	(5)	(7)	1
0	1	1	1	e	1	3	10	(6)	0
1	1	1	1	f	(8)	3	10	6	0
0	0	0	1	G	1	(9)	(11)	2	0,1
1	1	1	0	H	8	3	(10)	(12)	0
1	0	1	1	I	1	–	–	6	
0	0	1	1	J	1	–	–	6	
0	0	0	0	K	1	9	10	12	
0	0	1	0	L	1	9	10	12	
0	1	1	0	M	1	9	10	12	

FIGURE 27 – Revised Merged Flow Table with Secondary Assignments

					AB			
	y1	y2	y3	y4	00	01	11	10
K	0	0	0	0	0001 0	0001 1	0010 3	0010 2
G	0	0	0	1	0101 4	(0001) 5	(0001) 7	0101 6
J	0	0	1	1	0001 12	– 13	– 15	0111 14
L	0	0	1	0	0000 8	0000 9	0110 11	0110 10
a2	0	1	0	0	0101 16	0101 17	1100 19	(0100) 18
a1	0	1	0	1	(0101) 20	1101 21	1101 23	0100 22
e	0	1	1	1	0101 28	0101 29	1111 31	(0111) 30
M	0	1	1	0	0010 24	0010 25	1110 27	1110 26
d	1	1	0	0	1110 48	1101 49	(1100) 51	(1100) 50
b	1	1	0	1	1111 52	(1101) 53	1001 55	1111 54
f	1	1	1	1	(1111) 60	1101 61	1110 63	0111 62
H	1	1	1	0	1111 56	1100 57	(1110) 59	(1110) 58
	1	0	0	0	– 32	– 33	– 35	– 34
C	1	0	0	1	0101 36	0001 37	(1001) 39	1011 38
I	1	0	1	1	0011 44	– 45	– 47	0011 46
	1	0	1	0	– 40	– 41	– 43	– 42

(a) Excitation Map for Y1, Y2, Y3, and Y4

				AB			
y1	y2	y3	y4	00	01	11	10
0	0	0	0	1	1	1	1
0	0	0	1	1	(1)	(0)	0
0	0	1	1	1	–	–	1
0	0	1	0	1	1	1	1
0	1	0	0	1	1	0	(0)
0	1	0	1	(1)	1	1	0
0	1	1	1	1	1	1	(1)
0	1	1	0	1	1	1	1
1	1	0	0	1	1	(0)	(0)
1	1	0	1	1	(1)	1	1
1	1	1	1	(1)	1	1	1
1	1	1	0	1	1	(1)	(1)
1	0	0	0	–	–	–	–
1	0	0	1	1	1	(1)	1
1	0	1	1	1	–	–	1
1	0	1	0	–	–	–	–

(b) Output Map for f_{A-B}

FIGURE 28 – Excitation and Output Maps

Address							Data							
No.	A5	A4	A3	A2	A1	A0	B7	B6	B5	B4	B3	B2	B1	B0
	y1	y2	y3	y4	A	B	Y1	Y2	Y3	Y4	$\overline{Z}$			
0	0	0	0	0	0	0				X	X			
1	0	0	0	0	0	1				X	X			
2	0	0	0	0	1	0			X		X			
3	0	0	0	0	1	1			X		X			
4	0	0	0	1	0	0		X		X	X			
5	0	0	0	1	0	1				X	X			
6	0	0	0	1	1	0		X		X				
7	0	0	0	1	1	1				X				
8	0	0	1	0	0	0					X			
9	0	0	1	0	0	1					X			
10	0	0	1	0	1	0		X	X		X			
11	0	0	1	0	1	1		X	X		X			
12	0	0	1	1	0	0				X	X			
13	0	0	1	1	0	1								
14	0	0	1	1	1	0		X	X	X	X			
15	0	0	1	1	1	1								
16	0	1	0	0	0	0		X		X	X			
17	0	1	0	0	0	1		X		X	X			
18	0	1	0	0	1	0		X						
19	0	1	0	0	1	1	X	X						
20	0	1	0	1	0	0		X		X	X			
21	0	1	0	1	0	1	X	X		X	X			
22	0	1	0	1	1	0		X						
23	0	1	0	1	1	1	X	X		X	X			
24	0	1	1	0	0	0			X		X			
25	0	1	1	0	0	1			X		X			
26	0	1	1	0	1	0	X	X	X		X			
27	0	1	1	0	1	1	X	X	X		X			
28	0	1	1	1	0	0		X		X	X			
29	0	1	1	1	0	1		X		X	X			
30	0	1	1	1	1	0	X	X	X	X				
31	0	1	1	1	1	1	X	X	X	X	X			
32	1	0	0	0	0	0								
33	1	0	0	0	0	1								
34	1	0	0	0	1	0								
35	1	0	0	0	1	1								
36	1	0	0	1	0	0		X		X	X			
37	1	0	0	1	0	1				X	X			
38	1	0	0	1	1	0	X		X	X	X			
39	1	0	0	1	1	1	X			X	X			
40	1	0	1	0	0	0								
41	1	0	1	0	0	1								
42	1	0	1	0	1	0								
43	1	0	1	0	1	1								
44	1	0	1	1	0	0			X	X	X			
45	1	0	1	1	0	1								
46	1	0	1	1	1	0			X	X	X			
47	1	0	1	1	1	1								
48	1	1	0	0	0	0	X	X	X		X			
49	1	1	0	0	0	1	X	X		X	X			
50	1	1	0	0	1	0	X	X						
51	1	1	0	0	1	1	X	X						
52	1	1	0	1	0	0	X	X	X	X	X			
53	1	1	0	1	0	1	X	X		X	X			
54	1	1	0	1	1	0	X	X	X	X	X			
55	1	1	0	1	1	1	X			X	X			
56	1	1	1	0	0	0	X	X	X	X	X			
57	1	1	1	0	0	1	X	X			X			
58	1	1	1	0	1	0	X	X	X		X			
59	1	1	1	0	1	1	X	X	X		X			
60	1	1	1	1	0	0	X	X	X	X	X			
61	1	1	1	1	0	1	X	X		X	X			
62	1	1	1	1	1	0		X	X	X	X			
63	1	1	1	1	1	1	X	X	X		X			

"X" represents a bit to be programmed to a logic "1".

FIGURE 29 – MCM5004 Program Sheet for Pulse Subtractor

subtraction of two frequencies was found to be very accurate to the nearest readable digit. Two pulse generators (EH139B) were used for the frequency sources in the test setup.

Other application ideas are possible such as the digital pulse adder design in Figure 31. The dotted lines around the states S1 and S3 in the differential mode state diagram indicates unstable states. The heavy arrow leading from an unstable state indicates a change to a stable state although no other input transition occurs. Note, that a leading edge transition of the B input when in the unstable state S1 cause a movement to state S2. The output pulse width of the pulse adder is equal to the access time of the PROM. Note that arrows and rows without stable states are required in the primitive flow table resulting in another design tool when designing sequential circuits.

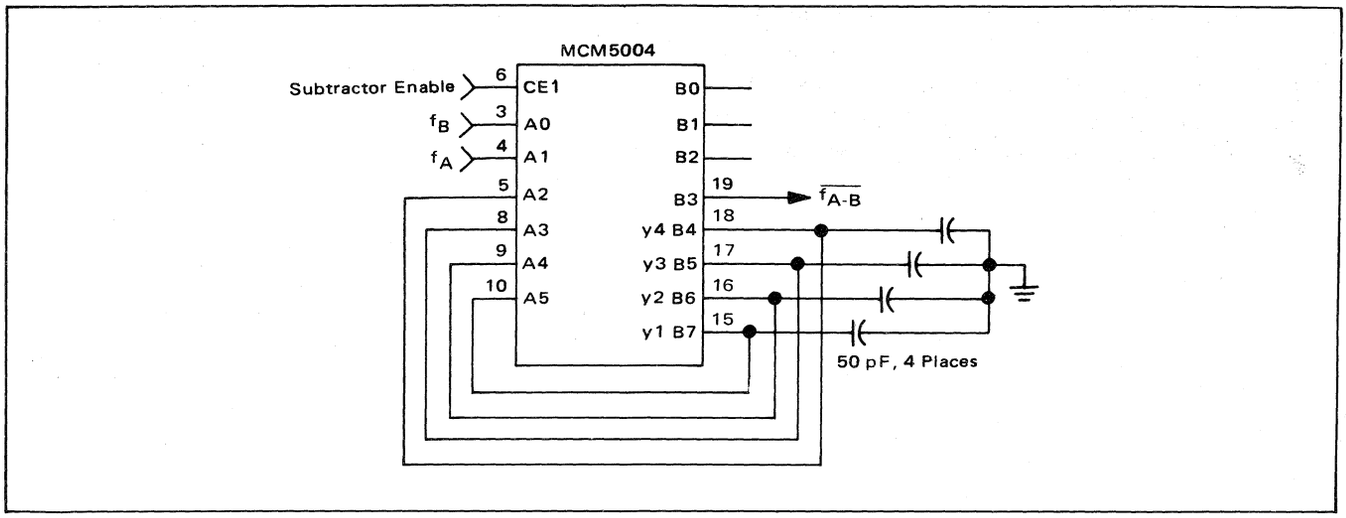
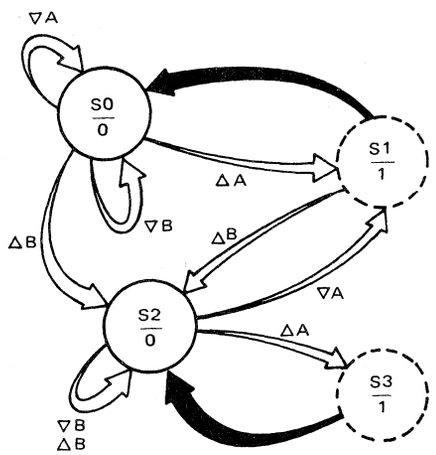


FIGURE 30 – Pulse Subtractor

SUMMARY

Practical applications for the ROM have significantly increased, since the introduction of the field programmable ROM. The number and variety of creative designs is unlimited. General applications have been illustrated to provide incentive to investigate these and other applications.

The use of ROMs and PROMs can result in reduced costs and decreased number of packages in the final system. As the ROM sizes get larger and the selling price gets lower, more application areas will open. Applications discussed included replacement of random logic, and asynchronous and synchronous sequential circuit design using ROMs with feedback. Many standard logic devices using ROMs may become available in the future for applications where volume is sufficient and where a ROM can perform the logic function at a lower cost. The growth potential for read-only memories looks promising, indeed, with possible usage in all areas of logic design. It is hoped that this article will inspire new techniques for using ROMs in replacing logic with the emphasis on economics and ease of use.



(a) DM State Diagram

State	AB Inputs 00	01	11	10	Output f_{A+B}
S0	(1)	3	6	2	0
S1	1	3	6	2	1
S0	1	3	6	(2)	0
S2	4	(3)	6	5	0
S2	(4)	3	6	5	0
S3	–	–	6	5	1
S2	1	3	6	5	0
S2	1	7	(6)	5	0
S0	1	(7)	8	2	0
S1	1	7	8	2	1
S0	1	7	(8)	2	0

(b) Primitive Flow Table

FIGURE 31 – DM State Diagram and Primitive Flow Table for Digital Pulse Adder

REFERENCES

1. Mitchell P. Marcus, Switching Circuits for Engineerings, Prentice-Hall, Englewood Cliffs, New Jersey, 1967.
2. William I. Fletcher and Alvin M. Despain, "Simplify Sequential Circuit Designs with Programmable ROMs," Electronic Design 14, July 8, 1971, P. 70.
3. John R. Smith, Jr., and Charles H. Roth, Jr., "Analysis and Synthesis of Asynchronous Sequential Networks using Edge-Sensitive Flip-Flops", IEEE Transaction on Computers, Vol. C-20, No. 8, August 1971, pp. 847-855.
4. "Programming the MCM5003/5004 Programmable Read-Only Memory," Motorola Application Note AN-550.
5. Jerry Prioste, "Sequential Design Techniques Using ROMs," IEEE Intercon, Session 28, No. 2, March 1974.

MOTOROLA Semiconductor Products Inc.

Printed in Switzerland