NAME (PRINT)	STUDENT ID	LAB DATE	GRADE

Experiment 6

Four-bit multipliers

In this lab, we implement in hardware some algorithms that multiply two 4-bit numbers to yield an 8-bit result. The operating speed and hardware complexity of these circuits is explored.

In digital electronics there are usually several ways to express a design in hardware. Generally, the various approaches to solving the problem involve a trade-off between the speed of operation (clock cycles or gate delays) and the hardware complexity (gate count) of the resulting circuit. To implement an elementary mathematical operation, namely the multiplication of two 4-bit binary numbers, several possibilities exist: a brute-force repeat addition with counting, a shift-and-add scheme similar to the decimal multiplication scheme we learned in the elementary school, *etc.* Some algorithms naturally lend themselves to a sequential implementation, using counters to sequence, or *loop*, the hardware a specific number of times, and an array of flip/flops known as an *accumulator*, to store intermediate results. Some other algorithms are naturally parallel, and their implementations can be in the form of a purely combinatorial circuit. Hybrid versions also exist.

? What is the range of binary values that multiplying two 4-bit numbers can yield? How many bits may be required to store/display the result of such an operation?

6.1 A summing multiplier

The most basic multiplication algorithm simply involves repeated addition of a multiplicand, with the number of times the addition is performed specified by a multiplier, *i.e.* $3 * 5 = 5 + 5 + 5$. A possible implementation of this algorithm in a 4-bit adder circuit is shown in Fig.6.1.

For proper operation, any sequential circuit requires:

- initialization to some known state; and
- a clock to sequence the logic.

In the circuit of Fig.6.1, a low on the reset line clears the accumulator so that the initial sum is zero. The rising edge of the clock latches the data into the accumulator, and an add/store operation is performed each clock cycle. The clock signal can originate from a switch that single-steps the circuit, or automatically from a square-wave oscillator.

To perform the addition of each bit, a full adder is used. The carry output of each adder ripples to the carry input of the adder representing the next-most-significant bit. The output of the adder is latched by a D-type flip/flop with clear. The word to be added (multiplicand) provides one input to the adder; the other input consists of the data currently stored in the accumulator.

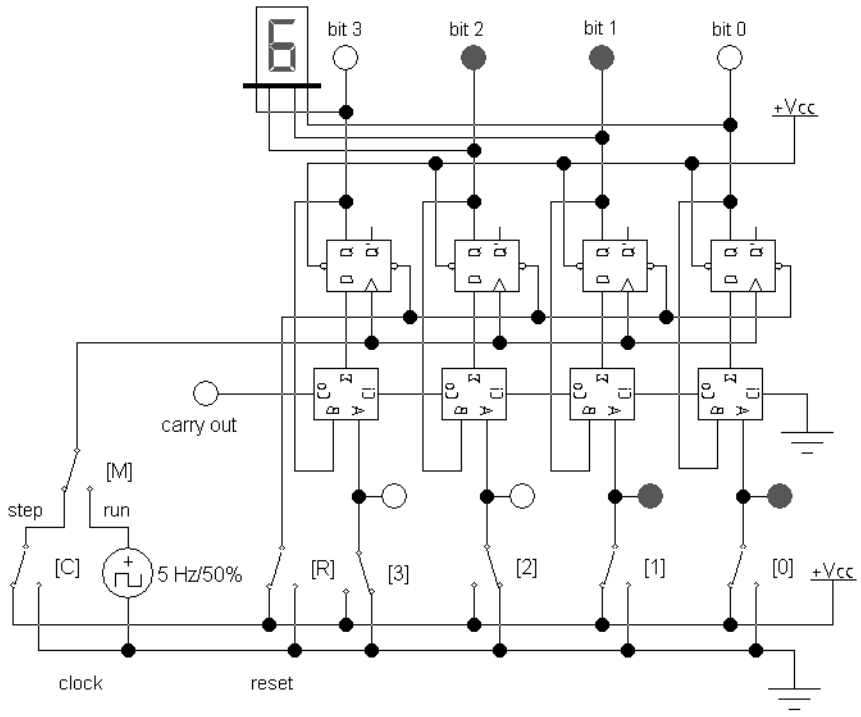

Figure 6.1: A 4-bit adder with accumulator

(!) Simulate this circuit. Try some input word values and step the circuit repeatedly by hand. What happens when the 4-bit capacity of the accumulator is exceeded?

[?] Given that the logic components used have a 10-ns propagation delay, what is the total propagation delay for the circuit? Which signal path determines the operating speed of this circuit? You might want to use the logic analyzer to monitor the timing relationship of the various signals.

(!) Increase the frequency of the clock to and beyond the operating frequency of the circuit so that a *signal race* condition occurs. What happens? Does the noted frequency agree with your estimate from the propagation delay?

The circuit will continue to add the multiplicand to itself as long as the clock continues to oscillate. In this 'manual' mode, you are responsible for counting how many clock cycles to provide to implement a particular multiplication operation. In addition, the limited number of bits quickly produces an overflow condition. We need to address both of these limitations.

(!) Extend the word size of the accumulator to double precision (8 bits). You can do this quickly by copying and pasting parts of the current circuit and making the proper connections. Verify that you can now serially multiply two arbitrary 4-bit numbers by setting the multiplicand via the four switches and clocking the circuit a number of times given by the multiplier.

Replace the manual clocking operation with an automatic one, using D-type latches connected as a down counter, as shown in Fig.6.2. Note the additions that have been made to the 8-bit adder circuit. The D-latches have active low preset and clear inputs. Switches set the 4-bit multiplier value which is loaded into the counter on reset. The rising edges of the clock pulses decrement the counter to zero and the circuit is then locked so that no further changes take place until a reset pulse restarts the circuit and another multiplication is performed.

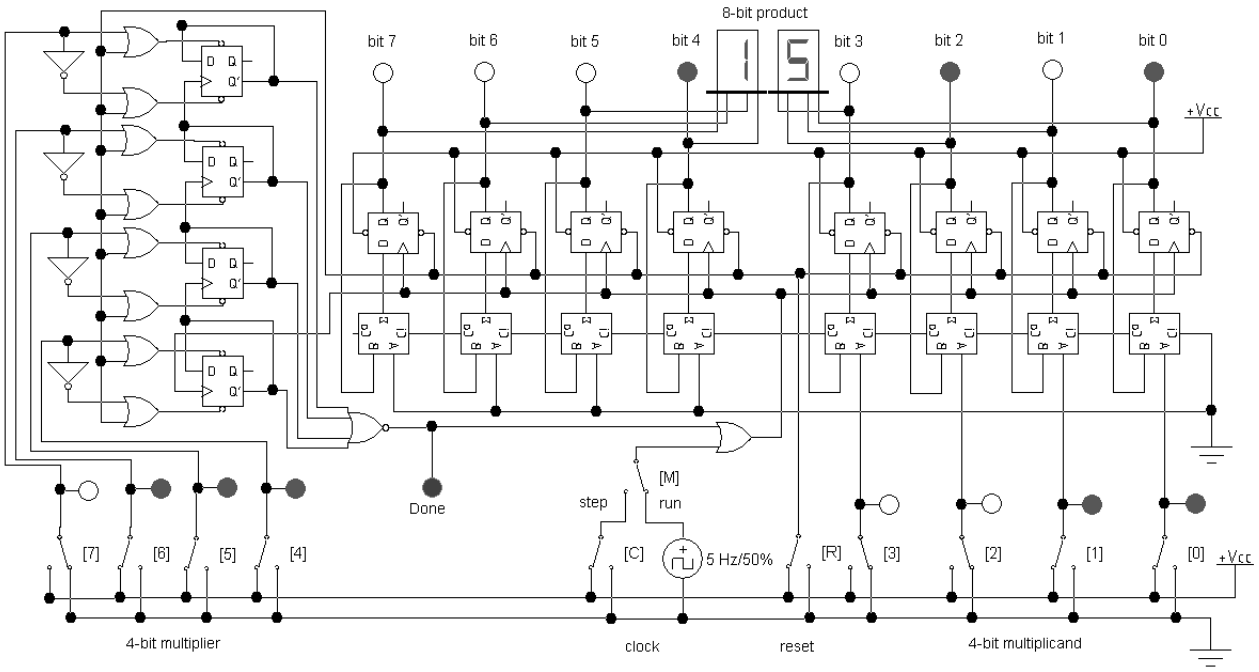

Figure 6.2: A 4x4 bit summing multiplier

(!) Modify your 8-bit adder circuit to match Fig.6.2 and verify its operation.

(!) Describe using truth tables and timing diagrams the operation of the logic used to load the counter with an initial value, and the logic used to lock the circuit so that a valid result can be read.

(!) *Optional:* Simplify this serial summing circuit. Hint: consider how the upper bits (bits 4–7) of the multiplicand enter the 8-bit adder circuit, as well as what happens to the overflow pattern of bits 0–3.

6.2 A shift/add multiplier

With some minor changes to our 8-bit adder circuit, a more efficient multiplication circuit can be realized. Binary multiplication can be performed in a manner analogous to the well-known procedure of decimal multiplication by hand, in which a series of shifts and additions — one for every digit of the multiplier — yield the product. In binary multiplication, this procedure is simplified by the fact that each binary digit has only two possible values, 0 and 1. To perform binary multiplication:

- initialize to zero both the accumulator and the loop counter;
- shift the contents of the accumulator, this is equivalent to a multiplication (or a division) by 2;
- for each bit of the multiplier in sequence: if the bit is a one, add the multiplicand to the accumulator; otherwise skip this step, *i.e.* add zero;
- repeat the above steps until the loop counter has counted the number of bits in the word.

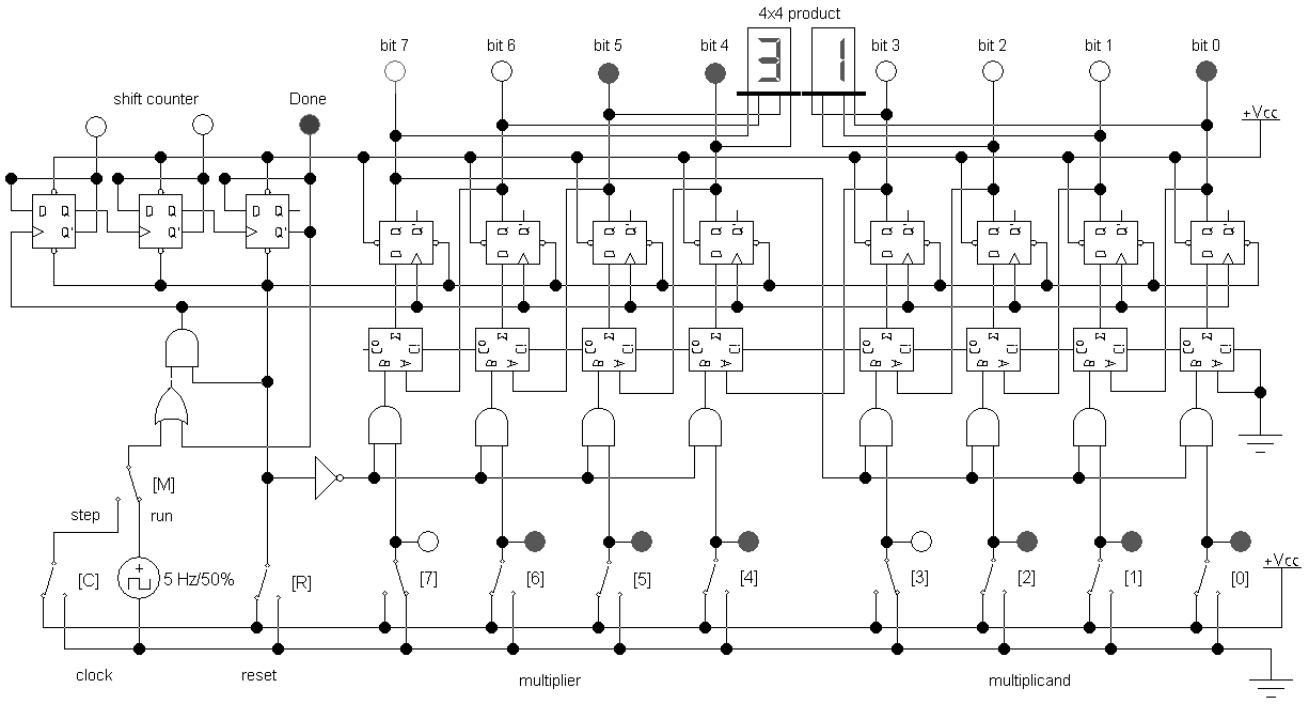

Figure 6.3: A 4x4 bit shift/add multiplier

The product is stored in the accumulator.

Fig. 6.3 shows an implementation of a 4x4 multiplier that uses a shift/add algorithm. The circuit is quite complex and we need to spend some time understanding its operation.

Let us begin by considering the operation of the circuit on reset. With a logic zero on the reset line, the accumulator and shift counter latches are cleared and the clock circuit is disabled. One input to the AND gates for bits 4–7 is set high so that the multiplier word can pass through the AND gates to the B inputs of the adders for bits 4–7, where it is added to the current value in the accumulator bits 4–7, which are zeros initially. Since bit 7 of the accumulator is zero, the AND gates for bits 0–3 are forced into a low state, setting the outputs of adder bits 0–3 to zero.

When the reset line makes a low-to-high transition, the multiplier word is latched into bits 4–7 of the accumulator, while bits 0–3 are loaded with zeros. AND gate outputs for bits 4–7 are set to zeros and the clock circuit is enabled.

Note that each bit in the accumulator is wired to the next-most-significant bit of the full adder array. Hence a left shift operation is being performed every clock cycle. The multiplicand is always added to the accumulator as a four-bit value, that is, the four most significant bits added are zero. This is exploited to reduce the total gate count: the four bits of the multiplier value can be initially loaded into the upper four bits of the accumulator without affecting the result as these bits will have shifted out of the accumulator.

On every low-to-high transition of the clock, the multiplicand (or zero, if the appropriate multiplier bit is zero) is added to the accumulator value and stored in the accumulator latches. The deciding condition is the state of bit 7 of the accumulator, which of course is reporting each of the bits of the multiplier as it gets shifted out of the upper four bits of the accumulator — which had been pre-loaded there by the reset. Note that the order in which the multiplier bits are processed is MSB-to-LSB, as the accumulator contents are shifted *left*. At the same time, the shift counter is incremented. In this case, use of an up counter is preferable since no preset logic is required for

it. A down counter would have to be loaded with the count value and use some logic to detect a zero output. When the shift counter reaches four, it sets the OR gate high, disabling further counts until a reset re-initializes the circuit.

(!) Simulate and verify the proper operation of the shift/add multiplier circuit. Consider some improvements to the circuit. For example, you may want to terminate the multiplication prematurely when one or both of the data words are zero.

[?] Since the multiplier value is set by switches, it is the accumulator value that is being left-shifted. Choose an arbitrary pair of 4-bit values for the multiplicand and the multiplier, and step through an entire multiplication cycle, recording the state of the accumulator after each step. Identify what happens to the multiplier bits, initially at bits 4–7, through the cycle.

[?] An alternate strategy would select the multiplier bit to test (is it =1 or =0) using a multiplexer driven by a down counter at its inputs. Sketch the relevant part of the circuit based on this idea. Compare this strategy to the one used in the circuit of Fig. 6.3.

[?] Compare the summing and the shift/add circuit implementations. How does the number of cycles required vary for the two circuits?

[?] Itemize the changes that you would need to make to the two circuits in order to perform an 8x8 multiplication resulting in a 16-bit product. How does the timing and gate count of the circuits vary with the size of the data word?

6.3 An array multiplier

Consider Figure 6.4, a design for a 4-bit array multiplier. Here, the four cycles of shift/add operations have been piggybacked into several stages that perform the series of operations all at once, combinatorially rather than sequentially.

(!) Simulate and verify the proper operation of the array multiplier circuit.

[?] Does this circuit offer a better performance than the shift/add multiplier?

6.4 Look-up table multiplier

Another possibility, especially when performing complicated arithmetic calculations such as evaluating the sine of an angle, is to use a lookup table. Here, a matrix of latches generically known as storage memory, is loaded with all the possible combinations of output values for a series of two input words. To implement a 4-bit multiplier, a memory array of 256 8-bit words is required. The multiplier sets bits 0–3 of the memory address while the multiplicand sets to bits 4–7. The resulting product on the output is simply the the data stored at the address selected by this combination of two 4-bit inputs.

[?] Evaluate the advantages and disadvantages of using this approach.

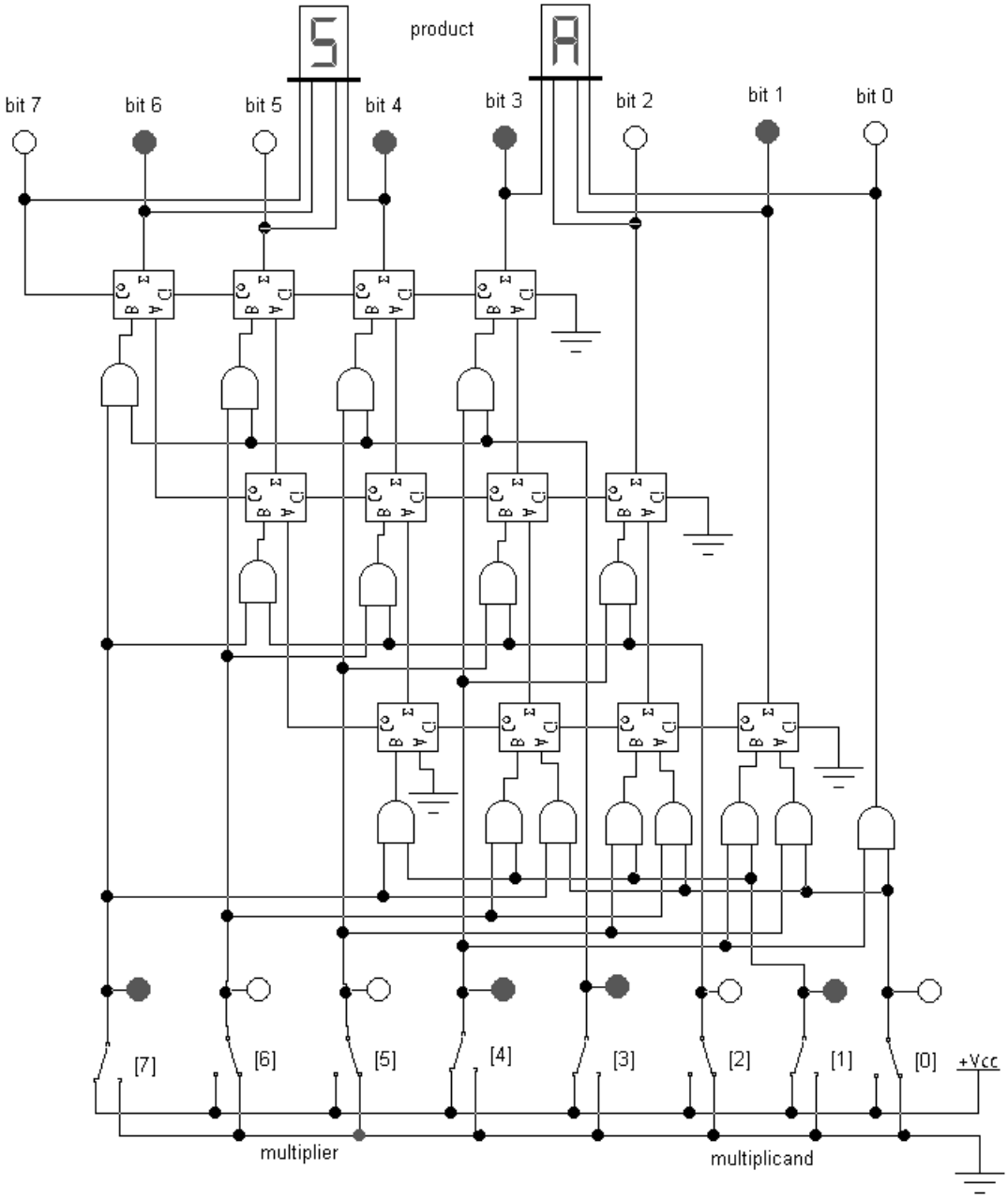

Figure 6.4: A 4x4 bit array multiplier