CPRE/COMS 583x Adaptive Computing Systems — Fall Semester 1998

Lecture 4: Thursday September 3, 1998
Lecture 5: Tuesday September 8, 1998

Instructor: Arun K. Somani & Akhilesh Tyagi — Scribe: Hue-Sung Kim

1 Implementation of functional units using LUTs

1.1 Look-Up Table

1.1.1 Single-bit output LUT

If there is a Look-Up Table (LUT) which has 'a' number of inputs, we can have 2^{2^a} possible functions. For example, 4-LUT(4-input LUT) can represent 2^{2^4} different functions. In addition, we can implement even $(2^{2^a})^n$ functions if there are n such LUTs. In this case, we assume that each memory produces only one output. A LUT with one output consists of a memory and multiplexer (Figure1). The multiplexer can be replaced by a decoder with appropriate addressing.

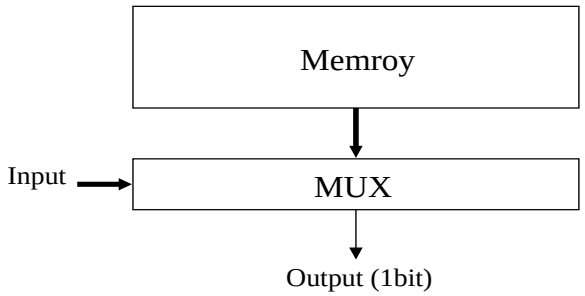

Figure 1: A basic structure of LUT

4-LUT is commonly used in most FPGAs. The reason for that is that the logic block including routing is area-efficient in FPGAs when it is implemented by 4-LUTs instead of other different input LUTs [1]. So, it is worth to find a way to implement a function with any number of inputs using 4-LUTs. If $(2^{2^4})^n \geq 2^{2^a}$, all the possible functions for a-input LUT can be realized. In other words, we can implement a-input LUT with n number of 4-LUTs where $n \geq 2^{a-4}$. Figure 2 shows an example of implementing 7-LUT using a number of 4-LUTs. The number of 4-LUTs are determined by $n \geq 2^{7-4} = 2^3 = 8$. In the figure, the multiplexers are also implemented by LUTs. So, the total number of LUTs needed is $2^{a-4}+2^{a-4} - 1 = 2^{a-3} - 1 = 2^{7-3} - 1 = 15$. By extending this concept, any function with any number of inputs can be implemented by single-bit output 4-LUTs.

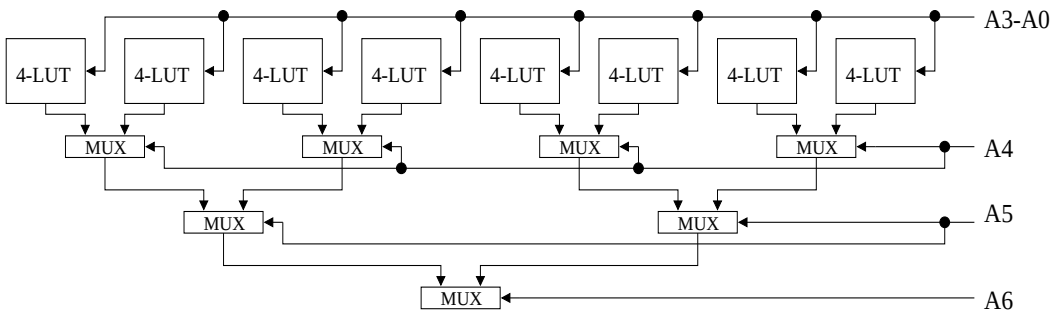

Figure 2: 7 input function implemented by 8 4-LUTs and 7 multiplexers

1.1.2 One large LUT with Multiple outputs

We can also implement any function using a large memory with multiple bits in the width. $(2^{2^a})^w$ functions can be represented in the large LUT as shown in Figure 3. The multi-bit output LUT can have only a set of inputs while each single-bit output LUT has its own set of inputs. This means that there is less flexibility in implementing functions in a multi-bit output LUT than in a single-bit output LUT.

In a large LUT with multiple outputs, no interconnection and carry propagation are required because all possible outputs can be stored into the big memory. In addition, unlike single-bit output LUTs, one large LUT does have only one big decoder or multipl exers with a inputs. The area of LUT reduces due to these factors. However, it may not be area-efficient when the number of possible inputs increases as the area increases by power of 2. The required memory size of one large LUT is much larger than that of small LUTs when the number of inputs increases. The computing time in this case may not be reduced much due to the complex decoder logic and the long bit line for reading the large memory. The following example shows how a large LUT is inefficient. Suppose we have a 64K *18bit memory, where w=18 and a=16(2 8bit inputs) in Figure 3. In the memory, we can have an adders and a subtracter. We compare the peak capacity and real capacity of the large LUT.

$$C_{peak} = \frac{w \times 2^{a-4}}{Area \times Time} = \frac{18 \times 2^{12}}{165M \times 40 \times 10^{-9}} \approx 11200 ge/\lambda^2 - s$$

$$C_{real} = \frac{N_{FAs} \times N_{ADD}}{Area \times Time} = \frac{16 \times 2}{165M \times 40 \times 10^{-9}} \approx 5 ge/\lambda^2 - s$$

where w=9×2(Each unit has 9bit outputs.) 2^{a-4}=# of 4-LUTs to represent a input-LUT with single output, N_{FAs}=# of FAs(LUTs) for one 8bit adder, and N_{ADD}=# of 8bit adders in the LUT.

In the example, we are not using all the capacity of the LUT at all, even less than 1 %. Therefore, mapping a function to a large LUT with multiple outputs makes it inefficient.

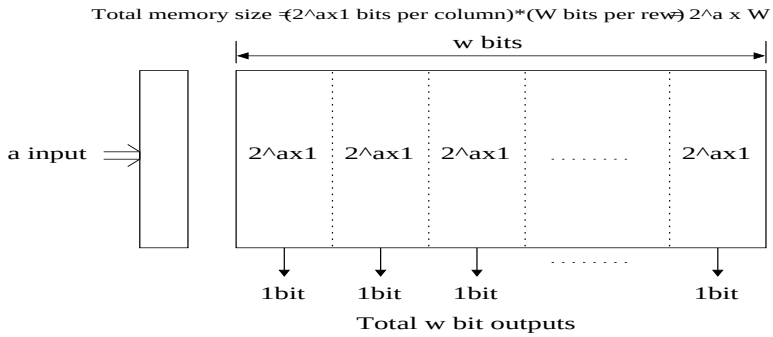

Figure 3: Multi-output LUT

1.2 Addition & Multiplication using multi-function LUT

1.2.1 Adders using LUTs

- **An adder using small Multi-output LUTs**
  We noted that one large multi-bit output LUT is not utilized efficiently to implement a function in section 1.1.2. Instead of using one large LUT, we show implementations of an 8bit adder with a number of small multi-bit output LUTs. First, look at an 8bit adder consisting of two 9-LUT. Each 9-LUT has two 4bit plus one 1bit-carry inputs and 5-bit outputs for a 4bit addition. The carry is propagated to the next 9-LUT only after the previous 4bit addition in one LUT is done (i.e. ripple carry). Since each LUT should be read one by one, this adder will take long time to finish an addition (Figure 4(a)). By employing the concept of carry select adder we can implement a much faster adder with 8-LUTs because reading the next LUT does not depend on the previous carry. The detail of the implementation is depicted in Figure 4(b). The total time of the modified adder is the read time for one 8-LUT and the time for two multiplexers. If we have more bits in inputs, the modified adder is much faster. However, this adder still does not utilize the area efficiencies (See Table 1). To make a better adder, a 4-LUT with 6bit outputs can be exploited (Figure 4(c)). The same scheme can be applied to the 4-LUTs to implement 8bit adder, but four such 4-LUTs are needed. The total time of the adder with the 4-LUTs might be larger than that with the 8-LUTs because it has twice the number of multiplexers to be propagated. However, the read time for a 4-LUT is faster than for an 8-LUT since it has a smaller decoder and shorter lines for reading memory. Table 1 shows the comparison of area and time for the implementations described above. We can see that the 4-LUT with 6bit outputs is area-efficient and has comparable time to others. For example, the ratio of real capacity and peak capacity for 8bit adder using 4-LUTs is $\frac{C_{real}}{C_{peak}} = \frac{8\times 2/A\times T}{4\times w\times 2^{a-4}/A\times T} = \frac{4}{6}$, where w=6 and a=4. For the sake of simplicity, the additional multiplexers are not considered. If we consider the additional multiplexers, the adder with 5-LUTs has the smallest area. However, the adder using 4-LUT with muxes has an advantage of computing time.

- **Adder/multiplier reconfigurable LUT**
  The 4-LUT described above can be reconfigured as a 2bit multiplier. Since we need 4bit output in

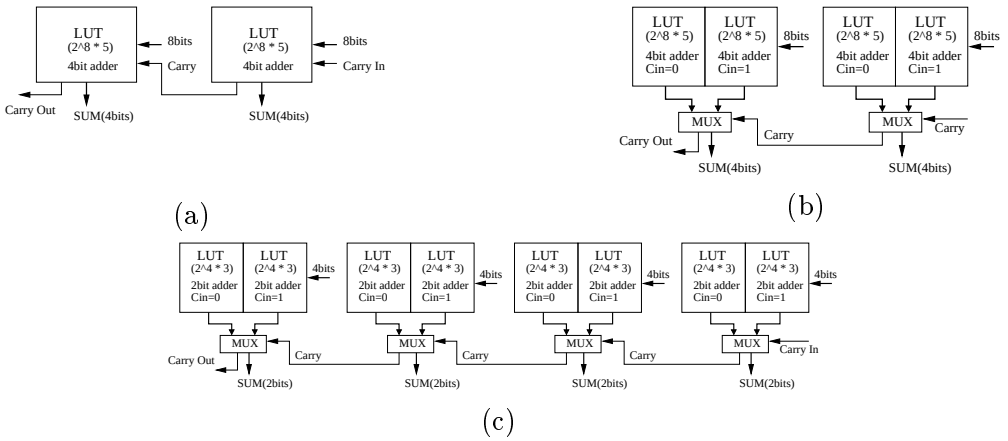

Figure 4: Scan Chain : 8bit adder using (a) two 9-LUTs ; (b) two 8-LUTs; (c) four 4-LUTs

Table 1: Comparison of area and time for 8bit adder using various LUTs

Type of LUT	Area utilization $(\frac{C_real}{C_peak})$	Total time
One 16-LUT	w=9, a=16 $\frac{8\times 2/A\times T}{w\times 2^{a-4}/A\times T} = \frac{16}{9\times 2^{12}}$	read time for one 16-LUT
Two 9-LUT	w=5, a=9 $\frac{8\times 2/A\times T}{2\times w\times 2^{a-4}/A\times T} = \frac{16}{10\times 2^{5}}$	read time for two 9-LUT
Two 8-LUT	w=10,a=8 $\frac{8\times 2/A\times T}{2\times w\times 2^{a-4}/A\times T} = \frac{16}{10\times 2^{5}}$	read time for one 8-LUT + 2 muxes
Four 5-LUT	w=3, a=5 $\frac{8\times 2/A\times T}{4\times w\times 2^{a-4}/A\times T} = \frac{4}{6}$	read time for two 5-LUT
Four 4-LUT	w=6, a=4 $\frac{8\times 2/A\times T}{4\times w\times 2^{a-4}/A\times T} = \frac{4}{6}$	read time for one 4-LUT + 4 muxes

a 2bit multiplier, we waste some portions of the 4-LUT. However, the 4-LUT for a 2bit multiplier is still area-efficient because a 2bit multiplier consists of a partial product generator and 2bit adder, which means the gate evaluation(ge) is higher than that of 2bit adder. Therefore, we are improving the area utilization for the multiplier. ($\frac{C_{real}}{C_{peak}} > \frac{4}{6}$)

To make a reconfiguration between a 2bit adder and multiplier, we need some additional logics. The detail of the logic is shown in Figure 5. The reconfigurable LUT can have D-FFs for pipelining. One question that arises for the reconfigurable LUT is why not combine the column decoder and carry selection logic into 3-to-8 decoder. It looks like an efficient way to reduce the area. However, it takes more area and time. The area of a 3bit decoder is roughly 2 times more than that of 2bit decoder including routings while the area of three multiplexers including interconnections is less than that of 2bit decoder. From the computation point of view, the entire 3bit decoder depends upon the previous LUT while only the multipliers do in the separated case. So, the time taken increases by combining the two. The total time for the 8bit adder in both cases is as follows.

$T_{separated} = T_{mem} + T_{2decoder} + 4T_{mux}$
$T_{combined} = T_{mem} + 4T_{3decoder}$
usually, $4T_{3decoder} > T_{2decoder} + 4T_{mux}$.

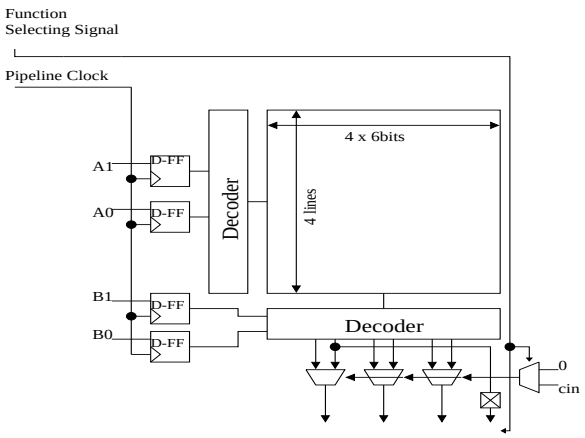

Figure 5: 2bit Adder/multiplier reconfigurable LUT

1.2.2 Multipliers using LUTs

- **Parallel Add multiplier**
  Figure 6 shows a common technique using parallel additions for the multiplication of two binary numbers on FPGAs [2]. Since there is no combinational logic on FPGAs, all the blocks in the figure are LUTs. By adding registers into each level of addition, this multiplier can be pipelined to produce higher throughput. However, it requires a large number of LUTs to implement all the logics.

- **Constant Coefficient Multiplier**

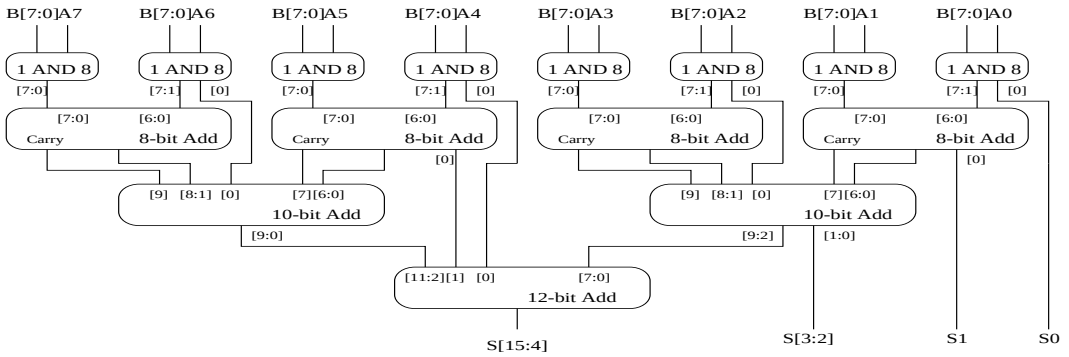

Figure 6: 8bit parallel add multiplier

Digital Signal Processing (DSP) and Matrix applications commonly calculate constant coefficient multiplication. If one of the input values is constant to the multiplier, each multiplication result of the constant value can be stored into n-LUT initially. So, a multiplication of $n \times C$ can be performed as one LUT read. If the coefficient is changed, we have to reconfigure the contents in the LUT. It takes 2^n write cycles to finish reconfiguration of the LUT. If the coefficient is not changed freque ntly in an application, then it is very useful. Figure 7(a) shows a 8bit constant coefficient multiplier with two 4×8 constant multipliers. The 4×8 constant multiplier consists of 12 4-LUT containing 16 partial products for an 8bit multiplication. Two partial products from two 4×8 constant multipliers are added in a bit-arranged 12bit adder. Unlike the parallel add multiplier, it just takes 2 steps for a multiplication. In addition, less area is required to implement the multiplier. Since it has reconfiguration time whenever the coefficient is changed as mentioned before, total computation time in both cases will be close.

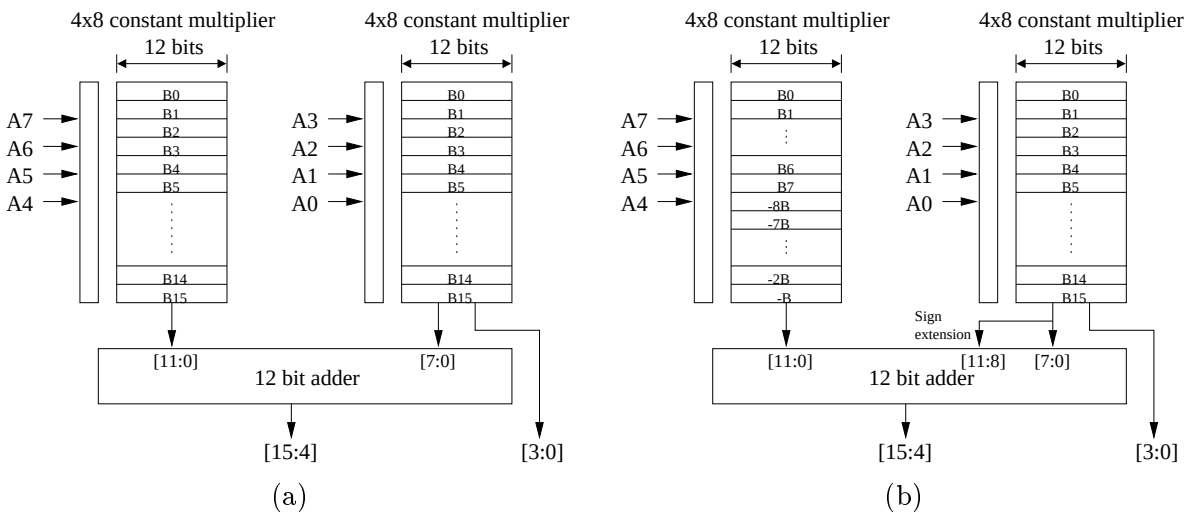

Figure 7: 8bit constant multiplier ($A \times B$) with (a) positive numbers ; (b) negative numbers (signed numbers)

- **Constant Multiplier with self reconfiguration**
  When a coefficient is changed, the pre-computed results of a multiplication should be rewritten into the LUT. These contents can be loaded from off or on chip configuration memory. However, this increases the memory bandwidth and is not an efficient way when the coefficient is determined by variable input values. Therefore, instead of loading configuration data from the memory, run-time reconfiguration of LUTs for a constant coefficient multiplication is proposed [2]. Accumulating sum of partial products for the coefficient, the contents for the LUT are produced one every cycle. This technique requires additional hardware such as a counter and accumulator. The total number of cycles for the reconfiguration is basically 2^n with an n-bit multiplicand. To reduce the number of reconfiguration cycles, we can split the LUTs to allow multiple writes to the LUTs. Depending upon the number of splited LUTs, the reconfiguration cycles can be reduced to half or quarter of 2^n cycles. In this case, we need more hardware like adders to support multiple writes. An example of an 8bit self-reconfigurable constant multiplier is given in [2]. In the example, we can have 16, 8, and 4 reconfiguration cycles when the LUTs are splited into 0, 2, and 4 LUTs, respectively. The result in [2] shows that the self-configuring constant coefficient multiplier with a number of reconfiguration cycles has less area than and comparable computation time of the parallel add multiplier.

- **Multiplication of negative numbers using LUTs**
  In the previous section, we consider only a multiplication of positive numbers. It is also possible to compute a constant coefficient multiplication with negative numbers using the same LUTs. Figure 7(b) shows the contents of LUTs for 8bit multiplier with signed numbers. Note that the sign bits should be extended according to 12th bit output of the 2nd multiplier.

References

[1] Stephen D. Brown, Robert J. Francis, Jonathan Rose, and Zvonko G. Vranesic, "Field-Programmable Gate Arrays", Kluwer Academic Publisjers, 1992, Chapter 4. Logic Block Architecture.

[2] Mathew Wojko and Hossam ElGindy, "Self Configuring Binary Multipliers for LUT addressable FPGAs", the 5th Australasian Conference on Parallel and Real-Time Systems, 1998.

[3] Andre' DeHon, "Reconfigurable Architectures for General-Purpose Computing", AI Technical Report 1586, Oct. 1986.