디지털 논리2 및

컴퓨터 공학 기초 실험2   
보고서

과제제목: Multiplier

실험일자: 2017년 11월 03일 (금)

제출일자: 2017년 11월 16일 (목)

학 과: 컴퓨터공학과

담당교수: 이준환 교수님

실습분반: 금요일 5, 6, 7

학 번: 2012722028

성 명: 장 한 별

1. 제목 및 목적
   1. 제목

Multiplier

* 1. 목적

Multiplier hardware의 기본개념에 대해서 이해한 뒤, 이를 구현하는 여러 가지 방법에 대해서 이해 할 수 있도록 한다. Binary Multiplication 의 개념에 대해서 먼저 이해 한 뒤, 나아가 radix-2, radix-4 의 개념에 대해서 이해할 수 있도록 한다. 후에 이를 구현할 수 있도록 하는 데에 목적이 있다.

1. 원리(배경지식)

**► Booth multiplication**

Radix-2

|  |  |  |  |
| --- | --- | --- | --- |
| <Rule of booth multiplication> | | | |
| xi | xi-1 | Operation | Description |
| 0 | 0 | Shift only | String of zeros |
| 0 | 1 | Add and shift | End of a string of ones |
| 1 | 0 | Subtract and shift | Beginning of a string of ones |
| 1 | 1 | Shift only | String of ones |
| 위 그림은 booth multiplication을 하기 위한 기본적인 규칙이다.  Multiplication은 Multiplicand(피승수) Multiplier(승수)를 곱하는 연산이다. 이때 booth multiplication의 방법은 두 개의 signed 2진 숫자를 2의보수의 원리를 이용하여 곱셈을 진행한다.  곱셈 방법은 Multiplier의 xi 와 xi-1을 1개씩 늘려가면서 비트 수만큼 비교를 하는 것이다.  자세히 말하자면, 일단 처음 Multiplier의 LSB(least significant bit)의 오른쪽에는 0비트가 있다고 가정한다. xi 와 xi-1 를 비교할 때(이때, xi와 xi-1을 비교하는 숫자는 multiplier이다), xi 가 0 이고 xi-1이 0 이면 오른쪽으로 1bit shift만한다. 이때, 모든 shift는 RSB이다. 그리고, xi 가 이고 xi-1이 1이면 add 후 오른쪽으로 1bit shift한다. 이때, add하는 숫자는 multiplicand이다. xi가 1이고, xi-1이 0이면, Subtract 후에 shift한다. 이때 subtract하는 숫자는 multiplicand이다.  subtract하는 방법은 2의 보수를 취한 뒤에 add를 하는 방식으로 하면 되겠다. 이렇게 진행하다 보면, 비트 수만큼 연산을 하게 되고 결과값을 도출할 수 있게 된다. Radix-2의 방법은 1이 연속적으로 많으면 좋은 곱셈 방식이지만, 0101 이런 식처럼 0과 1이 계속 교차되어 표현된 숫자의 곱셈연산에는 좋지 않을 수 있다. 따라서 이런 문제점을 보완하여 3bit씩 쪼개서 곱셈 연산을 진행하는 방식인 radix-4방식이 존재한다. | | | |

|  |  |  |
| --- | --- | --- |
| <Course of Radix-2 Booth Multiplication> | | |
| A  X  Y | 0101  0011(0)  0101’ | 5(Multiplicand) |
| 3(Multiplier) |
|  |
| Subtract A  Shift | 1011(inversion+1)  11011(RSB) |  |
|  |
| Shift | 111011(RSB) |  |
| ADD A  Shift | + 0101  001111  0001111 |  |
|  |
|  |
| Subtract A  Shift | 1011(inversion+1) |  |
| 1100111 |  |
| 11100111 |  |
| result | 11100111(2) | 15 |
| 위 과정은 booth multiplication의 radix-2 곱셈 과정을 나타낸 식이다.  위의 표에 나와있는 것처럼 xi 와 xi-1 를 비교할 때(이때, xi와 xi-1을 비교하는 숫자는 multiplier이다), xi 가 0 이고 xi-1이 0 이면 오른쪽으로 1bit shift만한다. 이때, 모든 shift는 RSB이다. 그리고, xi 가 이고 xi-1이 1이면 add 후 오른쪽으로 1bit shift한다. 이때, add하는 숫자는 multiplicand(5)이다. xi가 1이고, xi-1이 0이면, Subtract 후에 shift한다. 이때 subtract하는 숫자는 multiplicand(5)이다. subtract하는 방법은 2의보수를 취한 뒤에 add를 하는 하면 된다. | | |

1. 설계 세부사항

► state diagram

|  |
| --- |
|  |
| 위 그림은 Multiplier 의 state diagram 이다. 초기 상태인 IDLE, 연산 중인 EXEC, 연산이 끝난 DONE 총 3개의 state 로 나누어져 있고, 해당 조건에 해당 state로 넘어가거나 머무르도록 설계한다. |

► Pin description

|  |  |  |
| --- | --- | --- |
| Direction | Port name | Description |
| Input | clk | Clock |
| reset\_n | Active low reset |
| multiplier[31:0] | 승수 |
| Multiplicand[31:0] | 피승수 |
| op\_start | Start operation |
| op\_clear | Clear operation |
| Output | op\_done | Done operation |
| result[63:0] | Multiplier result |

위 표는 Multiplier 의 I/O description 이다. Input은 clk, reset\_n, op\_start signal, op\_clear signal 이 들어간다. 또한 32bits 의 multiplier 와 multiplicand 가 들어가게 된다. Output은 64bits 의 result 가 나오게 된다.

► block diagram

|  |
| --- |
| C:\Users\장한별 (Stanley)\Desktop\123.png |
| 위 그림은 Multiplier 의 block diagram 이다. input 으로는 op\_start, op\_clear, multiplicand, multiplier, reset\_n, clk이 들어가고, output으로는 result와 op\_done 이 나오게 됨을 확인할 수 있다. |

1. 설계 검증 및 실험 결과
   1. 시뮬레이션 결과

이번 시뮬레이션(testbench waveform)에서는 Multiplier를 계산하는 경우들을 여러가지로 나누었는데 첫째로 양수 X 양수 인 경우, 2. 음수 X 음수 인 경우, 3. 양수 X 음수 인 경우, 4. 음수 X 양수 인 경우 5. 양수 X 0 인 경우, 6. 0 X 양수 인 경우, 7. 음수 X 0 인 경우, 8. 0 X 음수 인 경우로 총 8가지로 나누었다.

|  |
| --- |
| 1. 양수 X 양수 |
| C:\Users\장한별 (Stanley)\Desktop\1.양양.JPG |
| 위 그림은 양수 X 양수(3 X 5)인 경우, Multiplier 의 waveform 이다. op\_start 가 1이 되어 연산 시작 signal을 입력시켰고, 연산 결과 15 가 정확하게 출력됨을 확인할 수 있다. |

|  |
| --- |
| * 1. 예외 처리1 |
| C:\Users\장한별 (Stanley)\Desktop\시간 오류.JPG |
| 위 그림은 1 번 testbench 에서 연산 시간을 충분히 넣어주지 않았을 때 waveform 이다. 위에서 확인하는 결과와 같이 연산 시간을 충분히 넣어 주지 않을 경우 연산 중 결과를 도출하지 못하고, 바로 다음 연산으로 넘어가므로 충분한 시간을 넣어주는 것이 매우 중요하다.  이 후 모든 경우의 수에서 충분한 시간을 넣어주었다 가정한 후 시뮬레이션 결과를 확인해 보도록 한다. |

|  |
| --- |
| * 1. 예외 처리2 |
| C:\Users\장한별 (Stanley)\Desktop\예외처리1.JPG |
| 위 그림은 1번 testbench 에서 multiplicand 와 multiplier 에 32bits 보다 높은 bit 를 입력시켰을 때의 waveform 이다. 위에서 확인하는 결과와 같이 32bits 보다 높은 bit를 입력시키면 제대로 동작하지 않는 것을 확인할 수 있다.  이 후 모든 경우의 수에서 32bits 이내의 bits 를 입력시키는 것을 가정한 후 시뮬레이션 결과를 확인해 보도록 한다. |

|  |
| --- |
| 1. 음수 X 음수 |
| C:\Users\장한별 (Stanley)\Desktop\2.음음.JPG |
| 위 그림은 음수 X 음수( -5 X -7) 인 경우, Multiplier 의 waveform 이다. op\_clear 가 1이 되어 초기화, op\_start 가 1이 되어 연산 시작된 후, 연산 결과 35가 정확하게 출력됨을 확인할 수 있다. |

|  |
| --- |
| 1. 양수 X 음수 |
| C:\Users\장한별 (Stanley)\Desktop\3.양음.JPG |
| 위 그림은 양수 X 음수( 9 X -2) 인 경우, Multiplier 의 waveform 이다. op\_clear 가 1이 되어 초기화, op\_start 가 1이 되어 연산 시작된 후, 연산 결과 -18가 정확하게 출력됨을 확인할 수 있다. |

|  |
| --- |
| 1. 음수 X 양수 |
| C:\Users\장한별 (Stanley)\Desktop\4.음양.JPG |
| 위 그림은 음수 X 양수( -5 X 5) 인 경우, Multiplier 의 waveform 이다. op\_clear 가 1이 되어 초기화, op\_start 가 1이 되어 연산 시작된 후, 연산 결과 -25가 정확하게 출력됨을 확인할 수 있다. |

|  |
| --- |
| 1. 양수 X 0 |
| C:\Users\장한별 (Stanley)\Desktop\5.양영.JPG |
| 위 그림은 양수 X 0 인 경우, Multiplier 의 waveform 이다. . op\_clear 가 1이 되어 초기화, op\_start 가 1이 되어 연산 시작된 후, 연산 결과 0가 정확하게 출력됨을 확인할 수 있다. |

|  |
| --- |
| 1. 0 X 양수 |
| C:\Users\장한별 (Stanley)\Desktop\6.영양.JPG |
| 위 그림은 0 X 양수 인 경우, Multiplier 의 waveform 이다. . op\_clear 가 1이 되어 초기화, op\_start 가 1이 되어 연산 시작된 후, 연산 결과 0가 정확하게 출력됨을 확인할 수 있다. |

|  |
| --- |
| 1. 음수 X 0 |
| C:\Users\장한별 (Stanley)\Desktop\7.음영.JPG |
| 위 그림은 음수 X 0 인 경우, Multiplier 의 waveform 이다. . op\_clear 가 1이 되어 초기화, op\_start 가 1이 되어 연산 시작된 후, 연산 결과 0가 정확하게 출력됨을 확인할 수 있다. |

|  |
| --- |
| 1. 0 X 음수 |
| C:\Users\장한별 (Stanley)\Desktop\8.영음.JPG |
| 위 그림은 0 X 음수 인 경우, Multiplier 의 waveform 이다. . op\_clear 가 1이 되어 초기화, op\_start 가 1이 되어 연산 시작된 후, 연산 결과 0가 정확하게 출력됨을 확인할 수 있다.  Multiplicand 나 Multiplier 에 0이 포함 된 4가지 경우의 수도 문제없이 출력됨을 확인할 수 있었다. |

* 1. 합성(synthesis) 결과

|  |
| --- |
| C:\Users\장한별 (Stanley)\Desktop\RTL.JPG |
| 위 그림은 Multiplier 의 RTL Viewer 이다. |

|  |
| --- |
| C:\Users\장한별 (Stanley)\Desktop\mul.flow.JPG |
| 위 그림은 Flow Summary 이다. Total logic element 가 286, Total registers 가 105, Total pins 이 133 임을 확인할 수 있다. |

1. 고찰 및 결론
   1. 고찰

이번 실습시간에는 multiplier hardware를 구현하는 문제였다. Multiplier를 구현하는 데에는 여러 가지 방법이 있지만, 실습 시간에 배웠던 내용은 binary multiplication, booth multiplication이 있었다. Booth multiplication에는 radix-2와, radix-4의 방법을 배웠는데, 필자는 booth multiplication의 radix-2의 방법으로 multiplier를 구현하였다.

Radix-2 방법의 장점은 multiplier가 0기 연속되어 나와있는 숫자에서는 단지 Shift 연산 만을 진행하기 때문에 더 기본 binary multiplication보다 성능이 좋아진다. 하지만 0101 이런 식으로 숫자가 연속적인 multiplier에서는 큰 효율을 보기가 힘들다. 하지만 이러한 문제점은 radix-4 방법으로 구현하면 쉽게 해결할 수 있다.

처음 multiplier를 구현함에 있어서 어려웠던 부분은 output 계산하는 calculate부분이었다. 디지털논리회로 수업시간에 배웠던 radix-2의 이론적인 내용은 알고 있었지만 이를 Verilog 코드로 옮기는 데에는 조금 어려움이 있었다.

처음엔 간단한 곱셈이지만 예상하는 결과가 나오지않아 당황했다. 알고보니 연산하는 시간에 충분한 시간을 넣어주지 않아 결과값이 나오는 데 까지 시간이 부족해 다음 연산으로 넘어가 나오지 않았던 문제가 있었지만, 충분한 시간(640ns)을 넣어 준 후 가볍게 해결됐다. 가장 까다로웠던 부분은, radix-2의 처음 비교하는 과정이 0비트의전에 0이 있다고 가정을 하고 비교를 하는 부분이 있었는데, 이 부분을 처음 result 가 0일때로 설정을 해준 뒤 testbench를 돌려보니 잘 되어서 문제가 없는 줄 알았는데, 만약 첫 비교 연산이 00 이 되어 Shift 만 하게 된다면, result가 계속 0이되어서 연산이 되지 않는 결과가 나왔다. 따라서 고민 끝에 count가 0일때라고 조건을 바꿔주게 되었고 어떠한 조건에서도 올바른 연산 결과값을 도출할 수 있게 수정하였다.

* 1. 결론

multiplier의 방법에는 binary Multiplication 방법과 booth Multiplication 방법이 있다. binary Multiplication 방법은 multiplier의 LSB부터 한자리씩 multiplicand에 곱하고 shift 하는 방식으로써 32bit X 32bit 연산이면 32번 연산이 수행되는 방법이다. 하지만 0을 곱하는 부분이 있는 partial product에서의 필요 없는 연산을 해줌으로써 시간이 소비되는 부분을 Booth Multiplication에서 보완이 될 수 있다.

따라서 radix 방법 중 가장 간단한 radix-2 방법으로 구현 하였는데 즉, 2bit를 비교하여 곱셈 연산을 수행하는 multiplier이다. LSB에서는 하위에 0이 있다고 가정을 하고 2bit씩 비교해서 multiplicand를 오직 shift만 할 지, 덧셈을 한 후에 shift할 지 아니면 뺄셈을 한 후에 shift를 할 지 정해주도록 한다.

또 Radix-2는 multiplier의 sign bit이 negative일 때와 positive일 때를 고민을 해주어야 한다. Sign bit이 positive일 때는 따로 특별한 경우 없이 정해진 방식대로 2bit씩 묶어서 비교를 하면 된다. 하지만 sign bit이 negative일 때는 다른 방식으로 sign bit을 계산한다. 하지만 단점이 있는데 중복된 bit가 없을 경우는 오히려 binary multiplication이 빠를 수 있다. Radix-4 방식을 사용하면 이러한 문제를 쉽게 해결할 수 있다.

1. 참고문헌

공진흥 / 컴퓨터공학기초실험2 / 새빛관303호(광운대학교) / 2017년.

이준환 / 디지털논리회로2 / 참빛관B101호(광운대학교) / 2017년.