

친절한 임베디드 시스템 개발자 되기 강좌 : 논리회로로의 확장

논리회로로의 확장

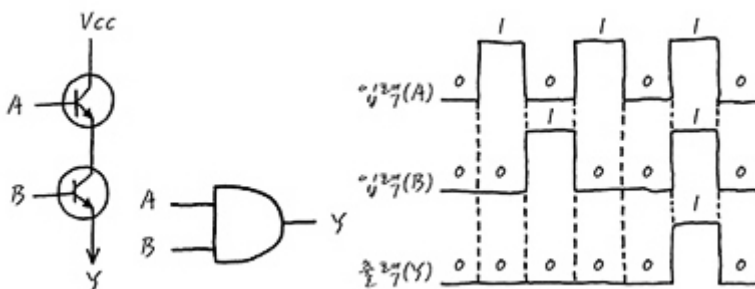
학교다닐 때, 위낙에 논리에 약해서 "논리야 놀자"라는 책을 탐독한 적이 있었는데, 그 논리의 정의로는 인간의 지식을 분석하여 논리적으로 명제화 하는 분야라는 문구를 본 적이 있는데, 이는 바로 이런 Computing Architecture에도 적용된다고 봅니다. 왜냐, 인간의 지식을 0과 1로 표현 하여, 이를 감쪽하게 요리질 한 후, 다시 0과 1의 명제로 정의 하여,저장하거나, 일을 시키거나 하니까, 결국은 Computer Architecture에서의 논리도 마찬가지경우라고 해야 할 것 같습니다. 우리가 흔히 Logic 1, Logic 0 으로 부르는 논리도 뭐 매 한가지라고 볼 수 있는데, 상당히 철학적인 내용이고, 더 깊게 다루자니 너무 무겁고 해서 일단 실전으로 들어 가야 하겠습니다.

TR과 R,L,C등의 기본 사용법을 익혔으니, 논리회로로의 확장이 필요합니다. 너무 깊숙히 들어가는 것 같지만, 알아두면 유용한 것들이니, 그냥 심심풀이로 생각하고 읽어보면 좋을 것 같습니다.

우선 이런 Computer Architecture를 이해 하려면, TR과 R,L,C등이 어떻게 엉켜 있는지알아볼 필요가 있습니다. 디지털 신호를 input으로 넣었을 때, 우리가 뭔가 원하는 output을 만들어 내려면, 논리적인 순서에 의해서 data를 manipulation할 필요가 있습니다.

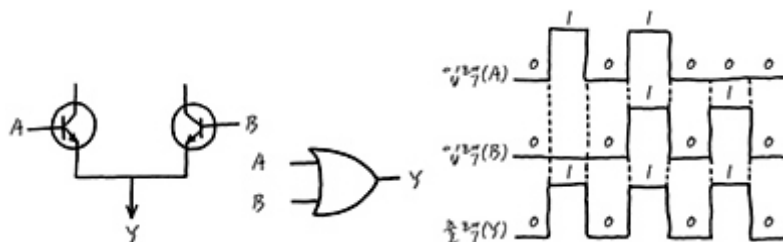
이런것을 논리 회로라고 부르며, 논리회로로 확장하기 이전에 논리회로는 어떻게 구현되는가를들여다 보겠습니다.

첫번째로 AND입니다. AND란, 논리곱이라고도 부르며, "if (A && B)" 와도 같은 concept입니다. 두개가 모두 TRUE (Logic 1)이어야 output이 TRUE 즉 Logic 1이 되는 것을 의미하며, 다음과 같이 표시하고 구현합니다. - TR을 이용한 회로는 여러가지 구현 방법이 있겠으나, 가장 간단한 예로 구현한 것이니까 참고하세요 -



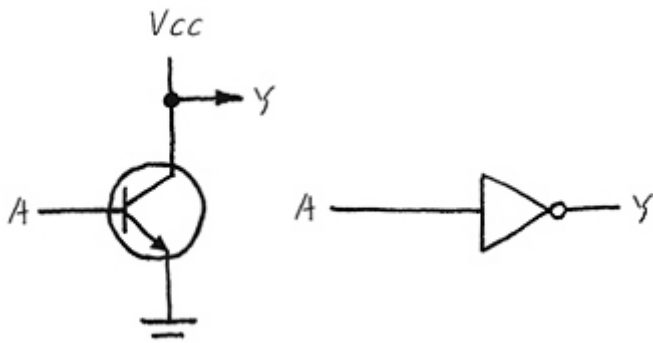
Transistor로의 구현은 TR두개를 직렬로 연결하고, A와 B가 모두 입력되어야 TR이 모두 ON이 되어Y output으로 Vcc가 출력됩니다. 이런 TR을 모두 그리기 번거로우니까 AND symbol로 뽕뽕그려서나타내며, 이 symbol 단위를 gate라고 부르며 결국에는 AND gate라고 부릅니다.

두번째로는 OR이며, OR란 논리합이라고도 부르고 "if (A||B)"와 같은 의미 입니다. 두개중 아무거나 하나 TRUE 가 되면 (Logic 1), output이 TURE, 즉, Logic 1이 되는 것을 의미하며, 다음과 같이 표시하고 구현합니다.








Transistor로 구현은 TR두개를 병렬로 연결하고 A와 B중 어느 것이라도 1이라면 TR이 ON이 되어Input Vcc가 output Y로 나가는 형태 입니다. 이 역시 OR gate라고 부릅니다.

회로를 구성하다 보면, inverter가 필요할 때도 있습니다 inverter는 input을 반전시키는 일을 하며, 표시와 구현은 다음과같이 합니다.



이 그림 보시면 생각나는 것 없나요?우리 switch로서의 TR의 역할에서 pull up할 때 봤잖아요.보시는 바와 같이 A가 Logic 1일 때는 Y로 0이 출력되고, 0일 때는 Y로 1이 출력됩니다.이런식으로 TR을 잘 이용하면, Logic gate들을 만들어 낼 수 있는데, 이런 Logic gate들을 이용해서 Digital Logic 회로를 만들어 내고, Logic회로들을 잘 조합하여 CPU를 만들어 낼 수도 있습니다. 이런 Digital Logic Gate는 AND, OR, Inverter만 있는 것은 아니고, 여러가지가 더 있습니다.

논리 회로	논리 회로 기호	A, B, Y	불대수 표현												
NOT		<table><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	0	0	1	1	$Y = A$								
0	0														
1	1														
NAND		<table><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	0	0	1	0	1	1	1	0	1	1	1	0	$Y = \overline{A \cdot B}$
0	0	1													
0	1	1													
1	0	1													
1	1	0													
NOR		<table><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	0	0	1	0	1	0	1	0	0	1	1	0	$Y = \overline{A + B}$
0	0	1													
0	1	0													
1	0	0													
1	1	0													
XOR		<table><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	0	0	0	0	1	1	1	0	1	1	1	0	$Y = A \oplus B$
0	0	0													
0	1	1													
1	0	1													
1	1	0													
NXOR		<table><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	0	0	1	0	1	0	1	0	0	1	1	1	$Y = \overline{A \oplus B}$
0	0	1													
0	1	0													
1	0	0													
1	1	1													

그러니까, 우리가 Chip vendor에서 사다가 쓰는 chip들의 내부는 이런 Logic Gate들을integration해서 특정 목적을 수행하려는 Logic 회로이라니까요. 재미 있는 것은 근래에는 이런 gate모임들도 VHDL이나 verilog등을 이용해서 Programming 언어로 표현하여, chip을 만들어 낼 수도 있습니다. 마치 C언어를 다루 듯이 input을 넣으면 어떤 Logic을 통해서 어떤 output이 나오도록 C의 함수처럼 chip을 구현한 후, 마치 CD 굽듯이 특정 chip burner에 넣으면 그런 Chip을 만들어 내는 기술이지요. 세상이 여간 편한게 아닙니다. (이런 chip들을 PLD/ PGA/ FPGA라고 부릅니다.)

이런 Logic gate를 이용해서 뭐하면 좋을지 간단한 예를 하나 짚고 넘어가겠습니다. 가장 간단한 예로 1bit Binary Adder를 하나 만들어 보겠습니다. Binary Adder의 진리표는 다음과 같습니다.

+	0	1
0	00 01	
1	01 10	

보시다시피, 1 bit + 1 bit는 2bit가 되죠. 가만히 보면 사실은 가장 낮은 LSB끼리더하고 LSB가 둘다 1일 때만 carry 1 이 발생 합니다. 고로, Binary Adder의 진리표는 다음과 같이 나눌 수 있겠습니다.

+	0	1		carry	0	1
0	0	1	과	0	0	0
1	1	0		1	0	1

<LSB>

<MSB>

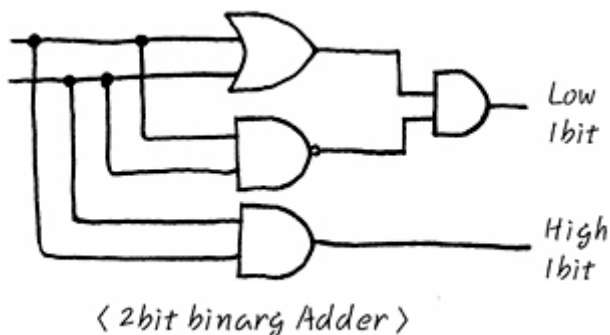
오호라, 이렇게 그려놓고 보니, 사실 왼쪽 표는 OR랑 비슷하고, 오른쪽은 AND그대로네요. 그러면, carry는 AND를 이용하고, 왼쪽의 OR는 약간 손봐야 겠는걸요. OR는,

OR	0	1
0	0	1
1	1	1

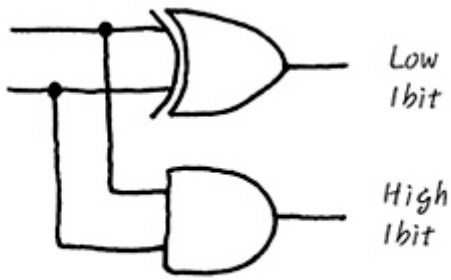
마지막 LSB진리표에 (2,2)자리인 1+1 자리에 1만 들어가면 엉악없는 OR인데 말이에요. 그러면, 제일 비스무리한게 뭐가 있을까 생각해 보면, NAND가 있네요. NAND의 진리표는

NAND	0	1
0	1	1
1	1	0

(2,2) 에 1+1자리만 0이네요. 그러면~ 두개를 AND해 버리면? 오호! 우리가 원하는 1의 자리만 더하는 <LSB>의 진리표가 나오게 됩니다. 고로 이를 모두 합하면, 다음과 같은 논리회로를 만들어 낼 수가 있습니다.



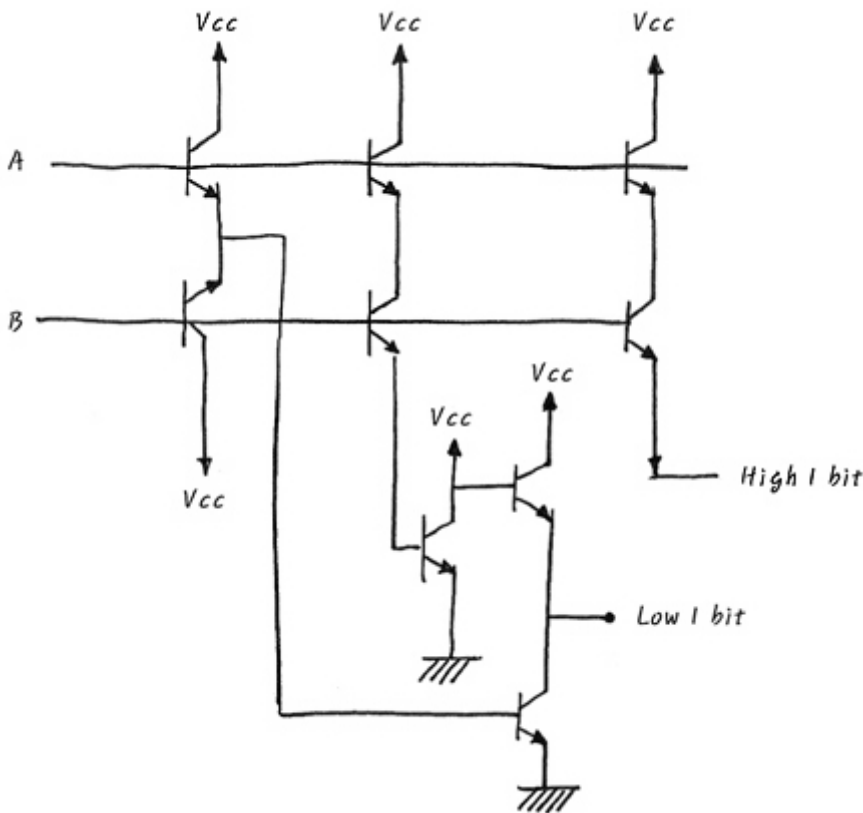
간단하게 OR하고 NAND를 AND하여, Low LSB 1bit를 만들어내고, 나머지 High MSB 1bit는AND를 해서 만들어 낼 수 있습니다. 오, 뿌듯하다. 그런데, 가만히 보면 OR하고 NAND를 AND해서 만들어 냈는데 보기 보다는 너무 복잡하네요. 이걸 이렇쿵 저렇쿵 해보면, OR과 NAND를 AND한 것은XOR과 같다는 걸 알 수 있습니다. (XOR는 두 input이 틀려야 TRUE가 되는 논리회로죠)- 가만히 생각해 보시면, XOR의 의미는 carry를 뺀 2진 합이라는 사실.이걸 기억해 두시면 훨씬 계산하기 편합니다. $0 \text{ xor } 0 = 0$, $0 \text{ xor } 1 = 1$, $1 \text{ xor } 1 = 0$ - 그래서, 조금더 간단하게 다시 그려보면,



< Simplified 2bit Adder >

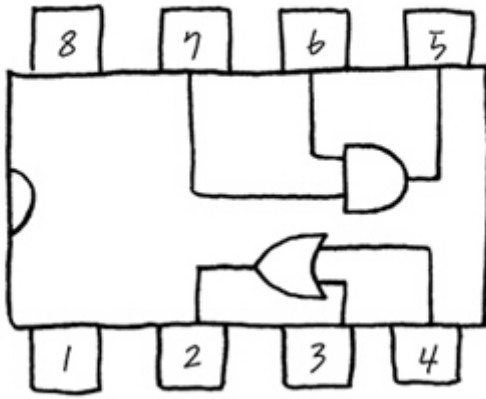
요령코름 간단하게 2bit Adder를 만들어 내었습니다. 짜잔. 이런 input과 원하는 output을 만들어 내기 위하여, Logic 회로를 구성하는

방법중 하나가 흔히들 말하는 카르노맵이고요, 이런 카르노맵을 구사하면 간단하게 input에 대한 원하는 output을 만들어 내는 논리 조합을 만들어 낼 수 있습니다. 물론 Macro하게 보면 이렇게 symbol로 간단하게 보고 있지만, 실제로는 저안에 많은 TR들이 들어 있겠죠, 그걸 잊지 말아야 합니다. 2bit Adder를 Transistor들의 논리 회로들을 실제로 그림을 그리면 다음과 같이 그릴 수 있습니다. 여간 복잡한게 아니죠?

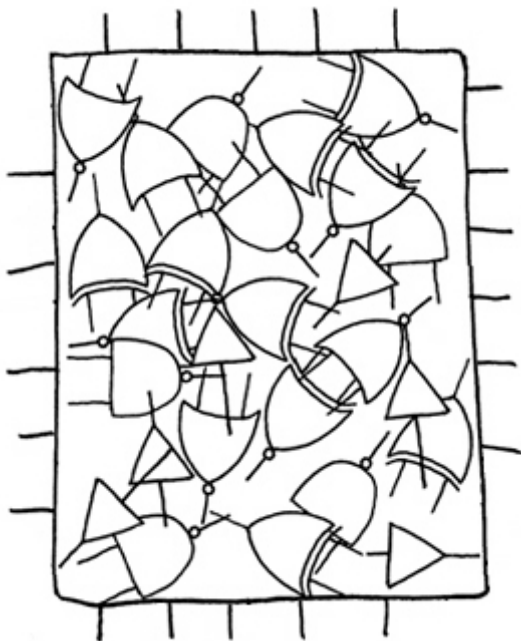


이런 논리 회로들은 다음과 같은 모양새로 IC라는 집적회로에 집적되기도 합니다. 예를 들면 다음과 같은 그림이죠. 7번 pin과 6번~ pin에 입력을 넣으면 5번 pin으로 AND output이 나온다. 또는 3번 pin과 4번 pin에 입력을 넣으면 2번 pin으로 OR output이 나온다 뭐 이런 식 입니다. - 엄청 간단한 예를 들려고 하니 민망하긴 합니다. -

U301



이런 논리조합들을 잘 조절 하면 우리가 원하는 output을 만들어내는 회로를 만들어 낼 수 있을 것입니다. Digital Chip set은 논리회로가 가득차 있는 모양이라니까요.



회로 이야기를 계속 하다보니, 한가지 notation을 짚고 넘어가면 좋을 것 같습니다. 우리 전압을 얘기 할 때, V_{cc} 라든가 V_{ee} 등의 notation을 많이 사용하였는데 그 의미를 되짚어 보면, 다음의 표와 같습니다.

BJT	FET	' V_{xx} ' 의미
V_{cc}	V_{dd}	Positive supply voltage
V_{ee}	V_{ss}	Negative supply, ground

BJT (Transistor)와 FET사이의 사용하는 notation이 다릅니다. V_{cc} 는 Collector 전원, V_{ee} 는 Emitter전원, V_{dd} 는 Drain 전원, V_{ss} 는 Source전원입니다. (FET는 BJT와 약간 다르게 Drain, Source, Gate로 다리가 구성되어 있습니다.) 요약하면, V_{cc} , V_{dd} 는 전원으로 연결되고, V_{ee} , V_{ss} 는 보통 Gound 단에 연결 됩니다. 보통 전자회로 책들을 보면 이런 식으로 구분되어 있습니다만, 근래에는 FET와 BJT가 혼합되어 마구 사용되어 그 부분이 모호해져서 V_{cc} 와 GND로 전원과 GND를 그냥 표시하는 일이 비일 비재 해 졌습니다.

통상 BJT를 이용하여 꾸민 회로를 TTL (transistor - transistor Logic)이라고 부르며, FET를 이용하여 꾸민 회로를 CMOS 회로라고 부르는데, BJT나 FET나 하는 역할은 비슷합니다. BJT와 마찬가지로 입력 전압으로 출력 전류를 제어 하는 역할을하는데, 결론론 적으로는 그렇고 원리는 많이 틀립니다. FET는 높은 input impedance를 가지고 있어 잔력 사용 효율이 좋으며, 단점으로는 속도가 느려 단순 스위치로는 적합하나, Linear Amplifier로서는 부적합 합니다.

이런 얘기는 다시 풀어 얘기하면, FET는 Logic쪽에 BJT는 Amplifier쪽에 어울린다는 의미에서 시작하자면, 역시나 Logic 쪽에서는 VDD - VSS, Linear Operation쪽에서는 VCC - GND의 notation이 많이 사용되겠습니다.

by 히언 | [2009/05/25 22:51](#) | [하드웨어콜라주](#) | 트랙백 | 핑백(2) | 덧글(4)

