논리회로 설계 및 실험

3주차

3주차 목표

목표

1. Encoder와 Decoder 및 가산기에 대한 이해
2. 반가산기와 전가산기를 이용하여 구조적 설계를 이해하고 이를 활용한 HDL 모듈의 확장 실습

Encoder와 Decoder

Encoder

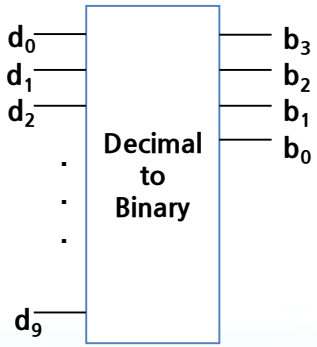
- 어떤 정보의 형태나 형식을 부호화(encoding)하여 다른 형태나 형식으로 변환하는 장치
- 처리속도 향상이나 데이터 압축 또는 데이터의 손실 방지를 위해서도 사용됨

Decoder

- Encoder로 변환한 정보를 그에 대응하는 원래의 정보로 복호화(decoding)하여 주는 장치

Encoder의 예

- 디지털 사진을 찍으면 실제로는 렌즈에 맺힌 상(analog)이 픽셀정보(digital)로 변환되어 저장됨
- 지난 2주차 실험에서는 십진수 정보를 이진수 형태(BCD code)로 변환하는 변환기를 구현함

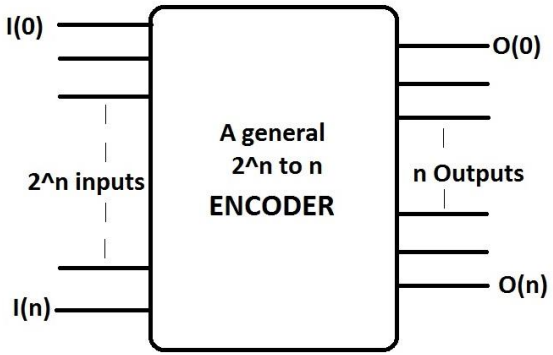


Decimal-to-Binary
변환기의 심벌

Encoder

Simple Encoder

- Simple Encoder는 one-hot code를 binary 정보로 변환함
- 이때 2^n개의 입력에 있어서 n개의 출력이 나옴



Priority Encoder

- 입력 bits의 MSB부터 출발하여 0이 아닌 첫 번째 bit의 index가 출력값이 됨
- 이때 해당 bit가 아닌 다른 bit 값들은 무시되며 압축(손실)이 일어남

4 to 2 Simple Encoder

I_3	I_2	I_1	I_0	O_1	O_0	V
0	0	0	0	x	x	0
0	0	0	1	0	0	1
0	0	1	0	0	1	1
0	1	0	0	1	0	1
1	0	0	0	1	1	1

4 to 2 Priority Encoder

I_3	I_2	I_1	I_0	O_1	O_0	V
0	0	0	0	x	x	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

올바른 one-hot code가 아니므로 O_1과 O_0는 don't care,
V(Valid)는 0을 출력

Binary	Gray code	One-hot
000	000	00000001
001	001	00000010
010	011	00000100
011	010	00001000
100	110	00010000
101	111	00100000
110	101	01000000
111	100	10000000

가산기 (Adder)

- 덧셈 연산을 수행하는 논리회로
- 한 자릿수 연산을 위해서는 Half adder, Full adder 등이 있음
- 멀티비트의 연산을 위해서는 Ripple carry adder, Carry look ahead adder 등이 있음

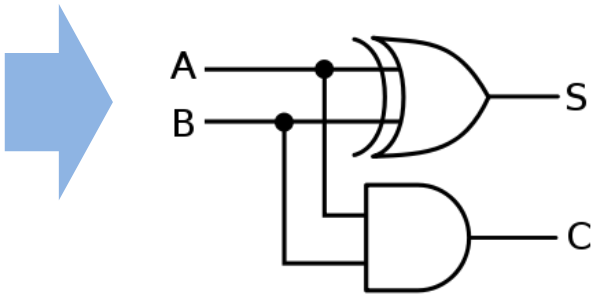
반가산기 (Half adder)

- 한 자릿수 덧셈을 수행하고 합(Sum)과 자리올림수(Carry)를 출력
- Carry는 AND gate, Sum은 XOR gate와 결과가 같음

	A	1
+	B	1
	S	0
	C	1

반가산기의 진리표

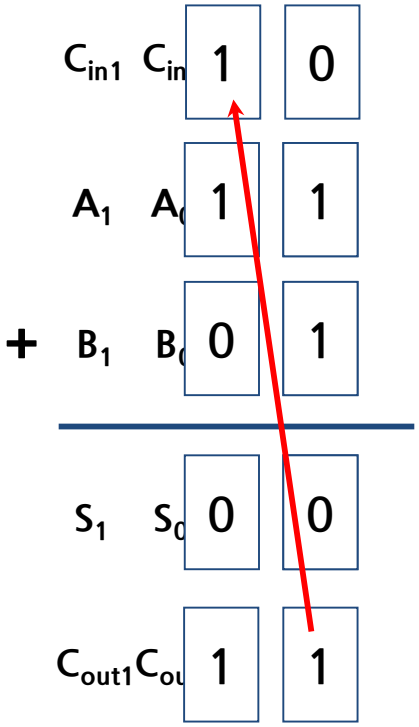
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



반가산기의 논리회로

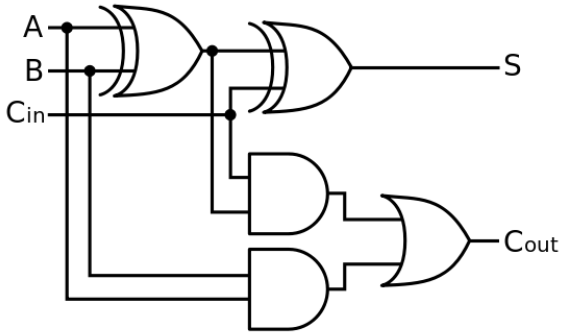
전가산기 (Full adder)

- 한 자릿수 덧셈을 수행할 때 이전 자리의 연산 결과로 받은 자리올림수(Carry)를 함께 연산하는 회로
- 두 개의 반가산기와 1개의 OR gate로 구성할 수 있음



전가산기의 진리표

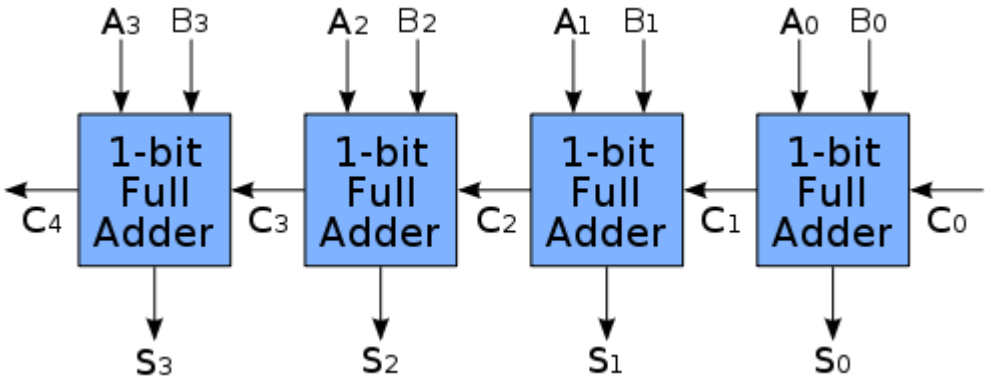
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



전가산기의 논리회로

리플 캐리 가산기 (Ripple carry adder)

- 복수의 전가산기를 이용하여 복수 비트의 덧셈 연산을 할 수 있는 가산기
- 간단한 구조이지만 전가산기의 입력이 이전 전가산기의 출력이므로 전달 지연이 발생함



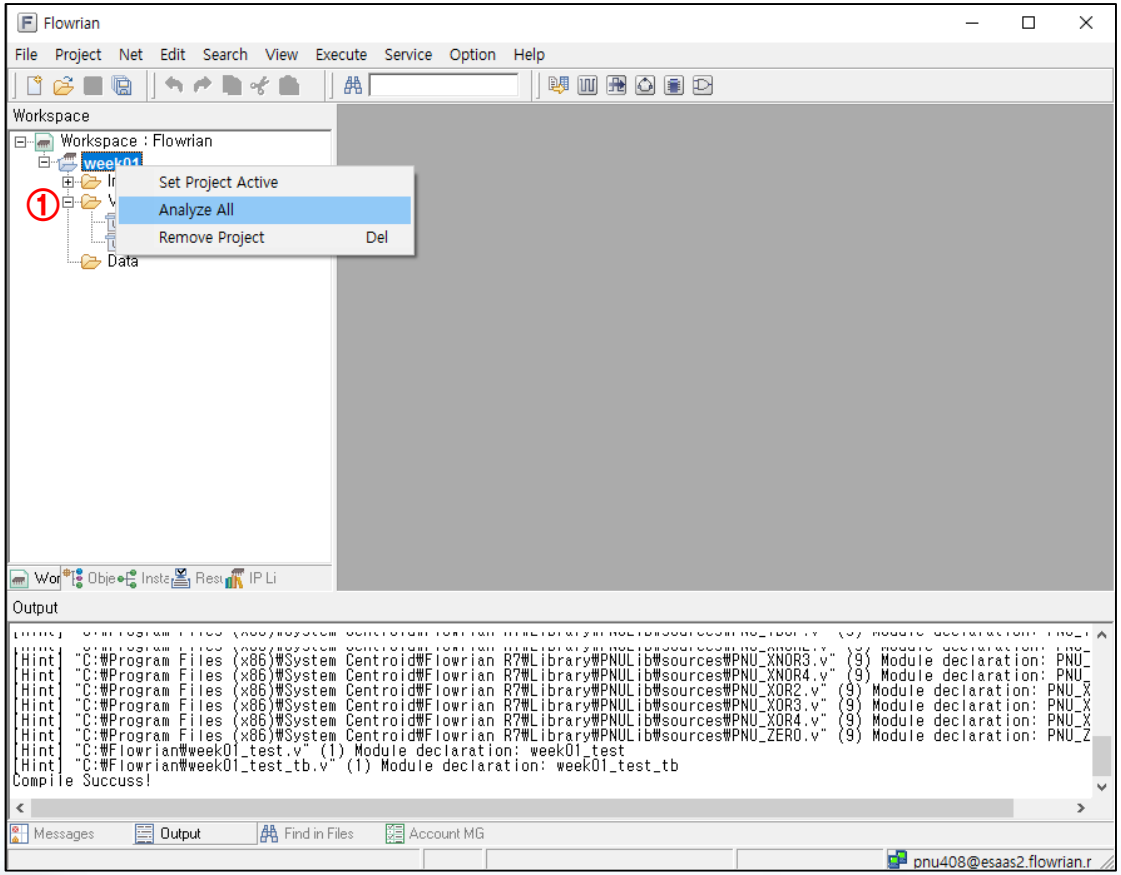
4bit Ripple carry adder의 구조

실습

Symbolian을 이용한 심벌 생성

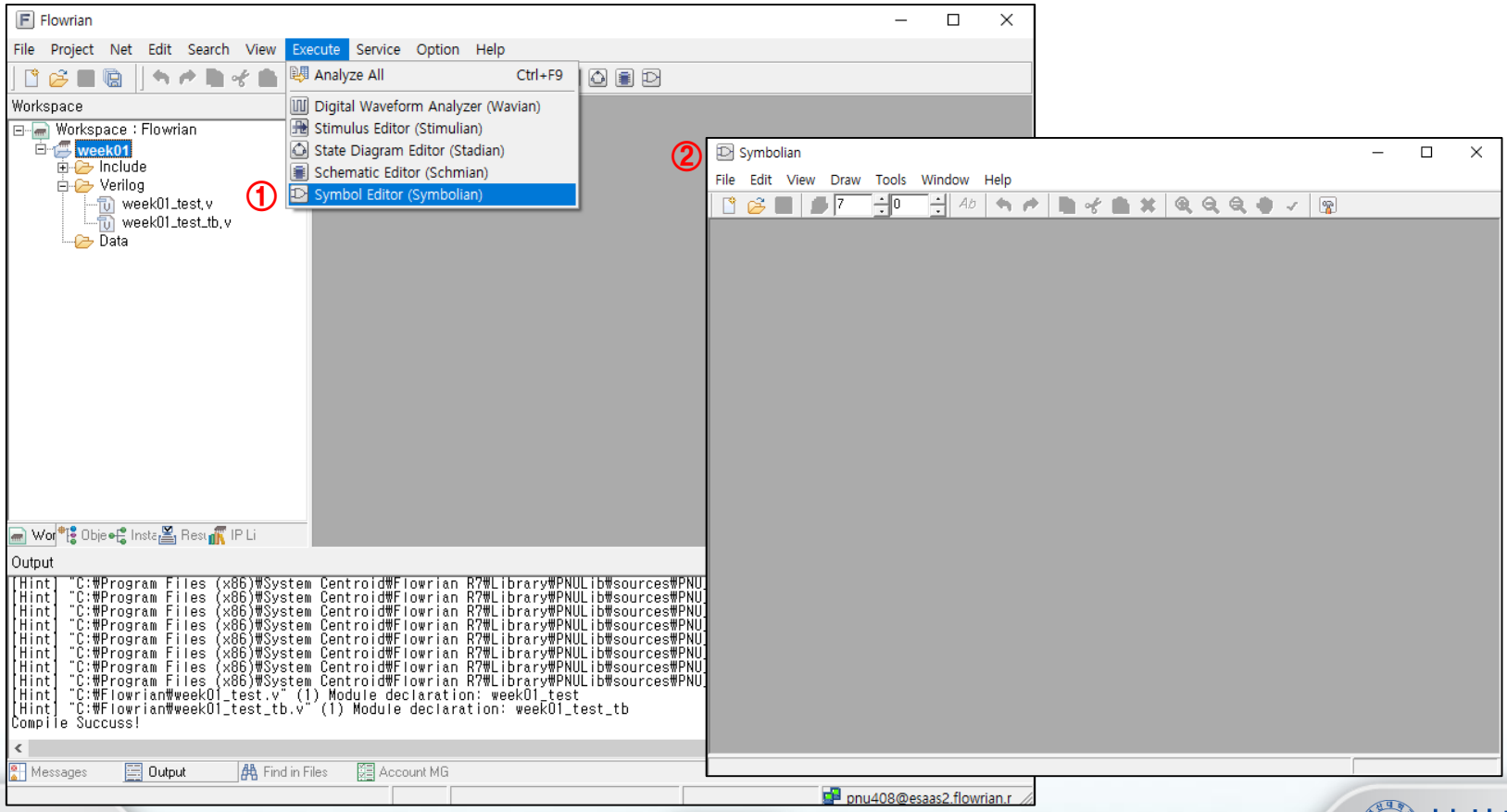
Symbolian 실행(1/2)

① [Analyze All]을 실행 (.v 파일을 추가한 상태에서)



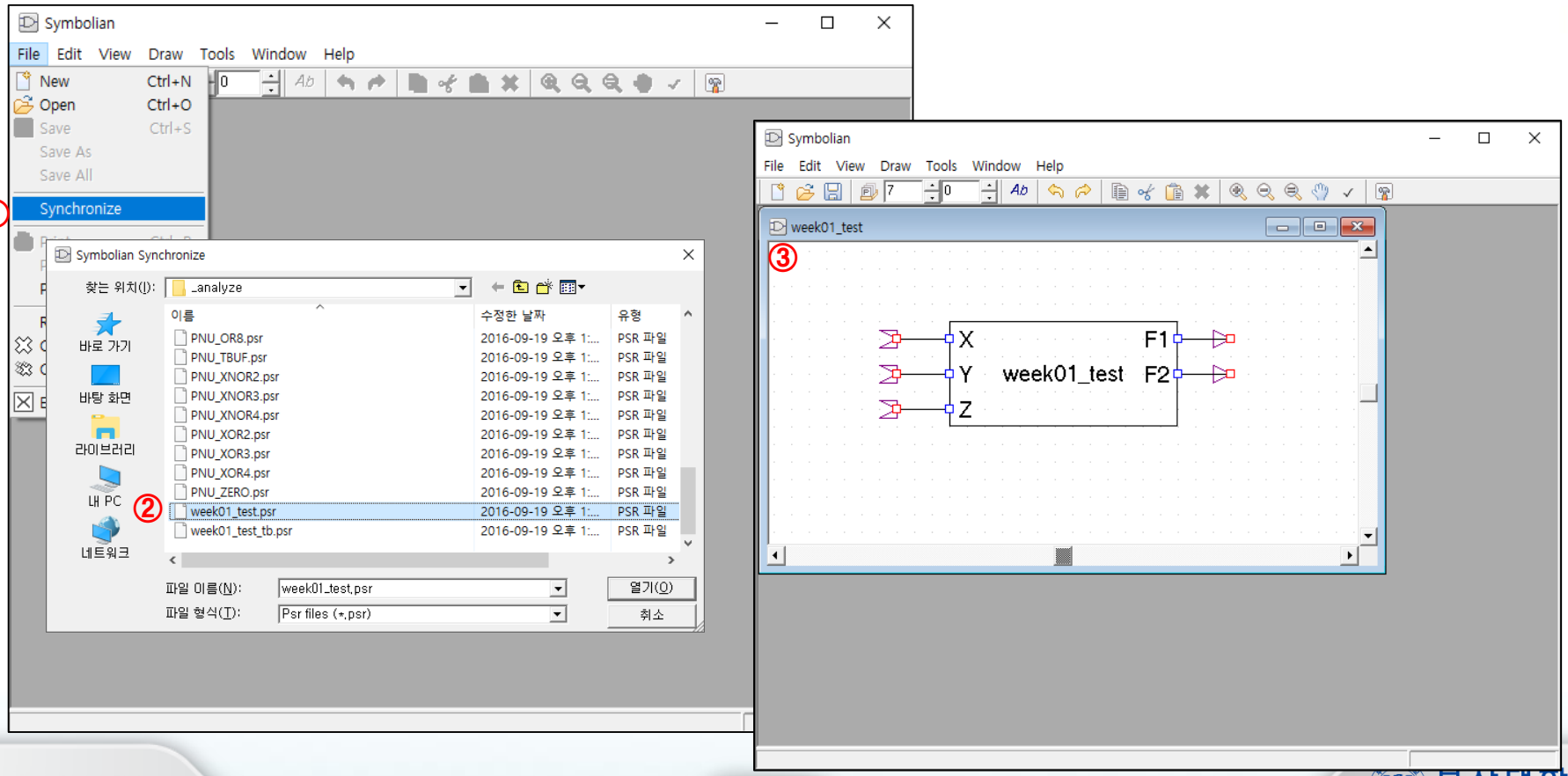
Symbolian 실행(2/2)

① [Execute] -> [Symbol Editor]클릭
② Symbolian 실행 화면



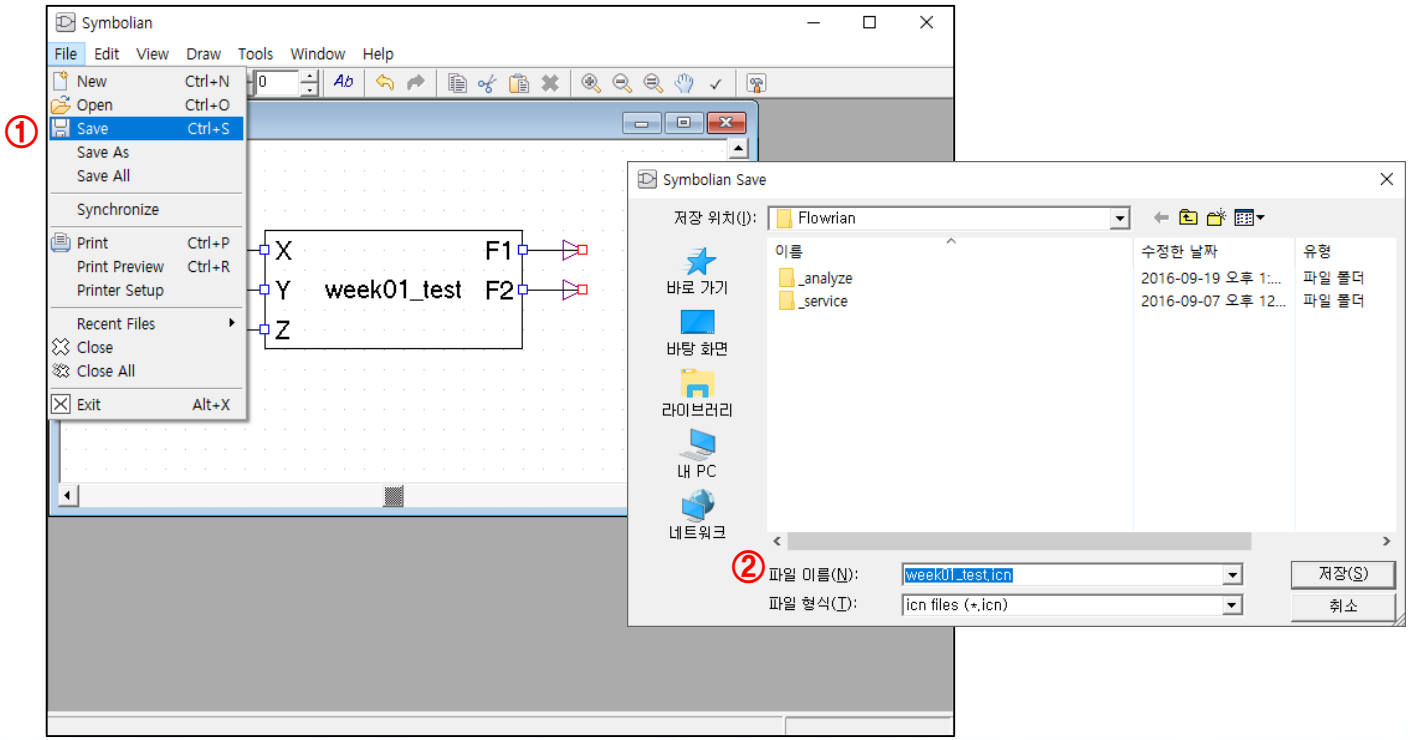
Symbol 생성

① [File] -> [Synchronize]클릭
② Symbol로 만들 .psr파일 선택
③ Symbol 생성



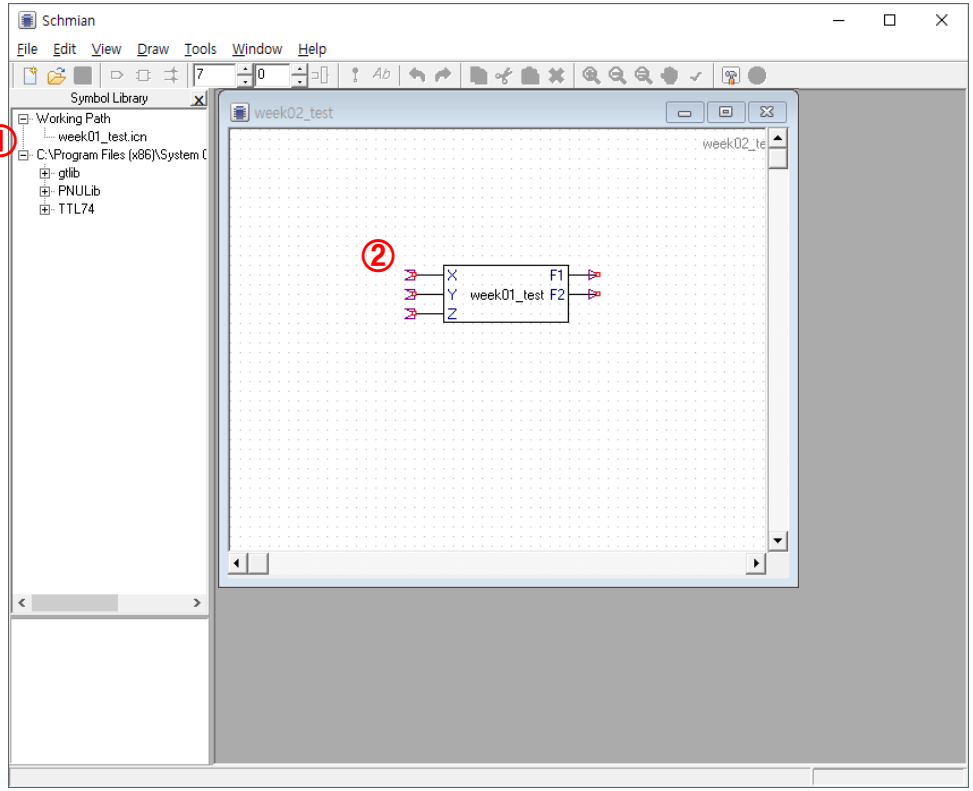
Symbol 저장

① [File] -> [Save]선택
② .icn파일 저장



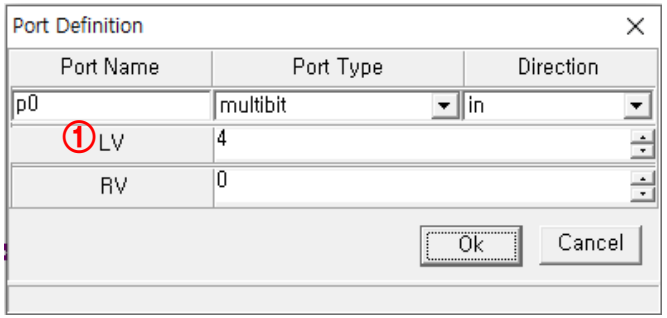
Symbol 사용

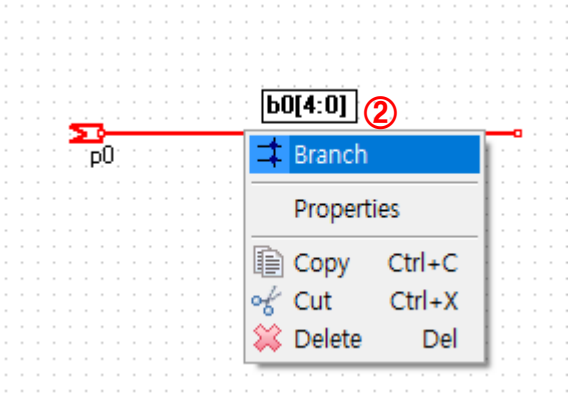
① Schmian에 자동으로 .icn파일이 추가됨
② 일반 게이트와 같은 방법으로 사용 가능
- 참고 : include에 .v 파일을 추가해줘야 analyze 가능



multibit 사용(1/2)

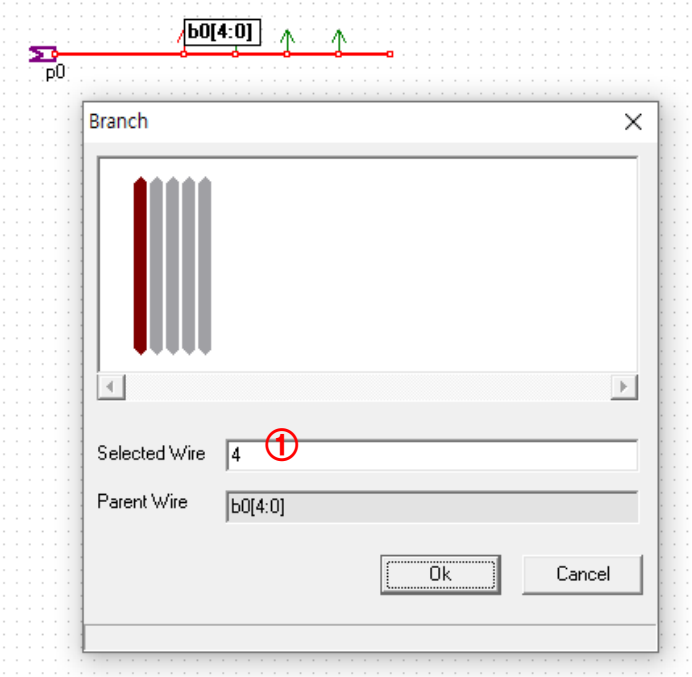
① LV, RV는 각각 몇번 비트에서 몇번 비트까지 사용할 것인지에 대한 지정
Ex) LV : 4, RV : 0 -> p0[4:0] (총 5비트)
② 각각의 singlebit를 사용하고 싶다면 wire 우클릭 -> Branch 클릭

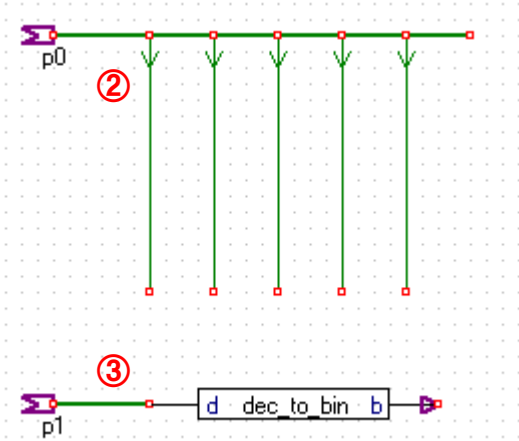




multibit 사용(2/2)

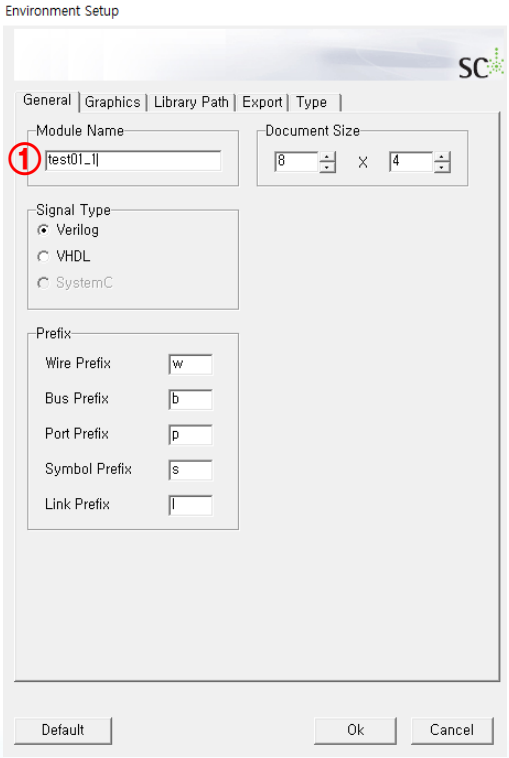
① Selected Wire에서 사용할 singlebit 선택
② multibit에 연결된 wire에서 각각의 singlebit를 Branch를 통해 뽑아내어 사용 가능
③ Symbolian에서 multibit를 사용한 .icn 파일을 만들었다면 multibit를 직접 연결 가능

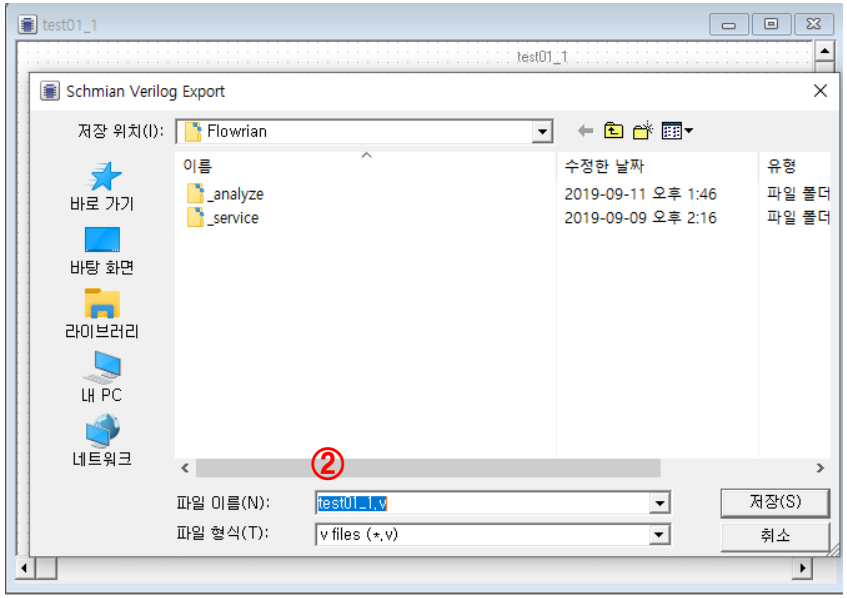




주의사항 – 해당 부분 지키지 않고 질문 시 감점

① Schmian에서 Module name은 꼭 알파벳으로 시작해야 하며,
이름에 특수문자 포함 x (예외: _ 는 가능함 (ex: test1_1))
② 설정한 Module name과 회로도 작성 후 Export 할 때 저장되는
.v 파일의 이름이 동일해야 함





실습 참고

주의사항

① 조별로 실습 검사 예정

② yongsu@islab.re.kr
위 메일로 실습 파일(ooo.sch, ooo.v, ooo.icn) 제출
- 조별 1명만 제출해도 점수 인정
- icn 파일 제출 시 해당하는 sch, v 파일 모두 제출

메일 제목 형식 : O주차_조_학번_이름
Ex) 3주차_1조_201612345_홍길동

파일명은 용도를 알아볼 수 있게 자유 형식으로 지정
Ex) full_adder.v, ripple_carry_adder.v