컴퓨터구조 프로젝트 보고서

Project 0

수업 명: 컴퓨터구조

담당 교수: 이성원

학과: 컴퓨터정보공학부

학번: 2023202070

이름: 최현진

제출일: 2025.04.

Introduction

이번 Proxy #1-1 과제는 Ubuntu 환경에서의 시스템 프로그래밍 실습에서 기본적인 파일 생성과 관련된 함수들의 동작을 이해하고 이를 활용하여 proxy 캐시 기능을 구현하는 것을 포함한다. 과제는 크게 두 가지 주제로 나뉘는데, 먼저 mkdir()을 사용한 소스 코드에서 함수에 주어진 인자대로 권한이 설정되지 않는 문제를 점검한다. 해결하기 위해 mkdir()과 umask() 함수의 동작을 이해하는 과정이 포함된다. 최종적으로는 문제가 발생한 이유와 해결된 소스 코드를 첨부한다. 다음으로 입력 받은 URL들을 SHA1 해시값으로 변환하고, 이를 기반으로 캐시 디렉토리 및 파일을 생성하는 프로그램을 구현한다. 이를 위해 정의한 sha1\_hash() 함수는 URL을 SHA1 해시값으로 변환하기 위해 사용한다. getHomedir() 함수는 사용자 홈 디렉토리의 경로를 반환한다. mkdir() 함수 등에서 인자로 ~를 입력하여 홈에 접근하는 것이 불가능하기에 사용한다.

2.1 Practice for digital logics with Logisim-E

|  |
| --- |
| 1. What is the Transmission gate?    Transmission Gate (TG)는 PMOS와 NMOS를 병렬적으로 연결한 스위치와 같은 회로이다.  NMOS와 PMOS는 각각 신호 전달 시에 손실이 발생한다. NMOS는 low 값 전달 시에는 손실이 없지만, week 1을 전달한다. 반대로 PMOS는 high 신호는 잘 통과하지만 week 0를 전달한다.  따라서 PMOS와 NMOS를 병렬로 연결하면, high 신호는 PMOS를 통해 손실 없이 전달되고 low 신호는 NMOS를 통해 손실 없이 전달된다. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2. Implement a MUX using Transmission gates    두 개의 TG를 사용해 구성하였다. 선택선 S가 1일 때, A와 연결된 TG가 열리고 B와 연결된 TG는 닫히므로 출력 Y는 A를 전달한다. 반대로 S가 0일 때는 A 쪽 TG가 닫히고 B 쪽 TG가 열리면서 Y는 B값을 출력하게 된다. TG는 S와 S’를 각각 제어선으로 사용한다.  S = 0 -> Y = B, S = 1 -> Y = A   |  |  |  |  | | --- | --- | --- | --- | | A | B | S | Y | | 0 | 0 | 0 | 0 | | 0 | 1 | 0 | 1 | | 1 | 0 | 0 | 0 | | 1 | 1 | 0 | 1 | | 0 | 0 | 1 | 0 | | 0 | 1 | 1 | 0 | | 1 | 0 | 1 | 1 | | 1 | 1 | 1 | 1 | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3. Implement a Latch using MUXs.    En = 0 -> Q에 이전 D 값 유지  En = 1 -> Q에 D 값 저장  MUX의 입력 0은 출력 Q의 피드백, 입력 1은 외부 입력 D로 연결했다. 선택선으로는 Enable 신호를 사용한다. logisim 내장 mux는 2번에서 구현한 회로와 다르게 S값이 입력과 같다. (S=0 input 0, S=1 input 1)   |  |  |  |  | | --- | --- | --- | --- | | En | D | 이전 Q | Q | | 1 | 0 | 0 | 0 | | 1 | 0 | 1 | 0 | | 1 | 1 | 0 | 1 | | 1 | 1 | 1 | 1 | | 0 | X | 이전 Q | 이전 Q | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 4. Implement D-FF using Latches.      3번에서 만든 latch 두 개 사용, master 래치에는 선택선에 CLK 를 연결하고, slave 래치 선택선에는 CLK’를 연결했다.  master의 출력이 slave의 입력으로 들어가 동작하게 된다.  따라서 CLK=1이면 마스터가 D값을 저장하고, CLK=0: 슬레이브가 마스터의 출력을 받아 Q에 D 값이 저장된다.   |  |  |  | | --- | --- | --- | | CLK | D | Q | | ↑ | 0 | 0 | | ↑ | 1 | 1 | | ↓ | X | 이전 Q | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5. Implement Synchronous Up/Down Counter using D-FFs and adders.  <components description>  CLK: 상승 엣지에서만 출력 A가 증가하거나 감소한다.(동기식)  reset: 초기값을 설정할 수 있다.  mux: 선택선(up\_down)을 통해 0일 때 up(0001을 adder의 입력으로 사용), 1일 때 down(1111을 adder의 입력으로 사용) 카운터로 동작하게 한다. A0(lsb)과 연결된 adder의 B 입력은 항상 1이기 때문에 mux는 3개를 사용한다. mux의 입력으로 상수를 사용할 수 있다.  adder: Q인 A 입력과 증감 모드에 따른 B 입력의 가산 결과로 출력은 d-ff의 d로 들어간다.  A3~A0: 카운터 출력값으로 lsb가 A0이다.  <truth table>   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | CLK | reset | up\_down | 이전 Q | 다음 Q | 설명 | | ↑ | 1 | X | XXXX | 0000 | 리셋 | | ↑ | 0 | 0 | 0000 | 0001 | up | | ↑ | 0 | 0 | 1111 | 0000 | up(overflow) | | ↑ | 0 | 1 | 0001 | 0000 | down | | ↑ | 0 | 1 | 0000 | 1111 | Down(underflow) |   <diagram example>    Logisim을 이용하여 구성한 4비트 Synchronous Up/Down Counter에서, up\_down 스위치를 토글하며 A의 출력이 0000 → 0001 → 0010으로 증가함을 확인하였다. DOWN으로 전환 시, 반대로 감소하는 것을 관찰할 수 있었고, 1111에서 증가 시 0000으로 되돌아가는 overflow 동작도 정상적으로 확인되었다.    -up\_down: 0 (증가 모드)  0000~1111까지 1비트씩 A가 증가하는 up counter 동작을 수행한다. 1111에서 overflow시 0000으로 돌아온다.    -up\_down: 1 (감소 모드)  1111~0000까지 A가 1비트씩 감소하는 down counter 동작을 수행한다. 0000에서 underflow 시 1111으로 출력된다.  현재 회로에서는 메모리 없이 FF로만 상태를 저장하고 있다. 프로그램 가능한 카운터로 확장하기 위해서, ROM이나 RAM 블록을 활용하여 초기 상태를 지정하거나 조건에 따라 상태를 바꿀 수 있도록 개선할 수 있을 것이다. |

|  |
| --- |
| 6. Compare Synchronous counter and Asynchronous counter such as a ripple counter.  공통점:   * 상향(up), 하향(down) 카운터가 있다.   Synchronous counter(동기식 카운터):   * 모든 ff에 동일한 클럭 신호를 사용하기 때문에 고속 처리가 가능하다. * 프로세서, 메모리 주소 생성 등 정밀한 제어가 필요할 때 응용된다.   Asynchronous counter(비동기식 카운터, 리플 카운터)   * 회로가 간단하다. * 첫번째 ff에만 클럭 신호가 입력된다. * 앞쪽의 ff 출력이 뒤쪽의 ff의 클럭으로 사용된다. * ff가 많아짐에 따라 delay가 커진다. * 오차 가능성이 높다. * 간단한 타이머, 빈도 측정에 사용된다. |

|  |
| --- |
| 7. What makes that Asynchronous counter is rarely used?  비동기식 카운터는 6번에서 설명했듯, 이전 ff의 출력이 다음 ff의 클럭 신호로 연결되는 방식이기 때문에 전파 지연(propagation delay)이 누적된다. 이로 인해 카운터의 속도가 느려지고, 실제 출력에 오차가 생긴다. 따라서 빠르고 정밀한 동작이 요구되는 실제 시스템에서는 비동기식 카운터가 거의 사용되지 않는다. |