Bit-Operationen

Informatik Grundlagen
Kantonsschule am Burggraben

Ivo Blöchliger

Web-Entwicklung

- Accounts
- Verbindung mit dem Server

Verknüpfungen von 2 Bits

- Eingabe: 2 Bits
- Ausgabe: 1 Bit
- Wie viele Eingaben sind möglich?

- Wie viele solche Verknüpfungen gibt es insgesamt?

Verknüpfungen von 2 Bits

- Eingabe: 2 Bits
- Ausgabe: 1 Bit
- Wie viele Eingaben sind Möglich?
  2 Bits → 4 Zustände, also 4 mögliche Eingaben
- Wie viele solche Verknüpfungen gibt es insgesamt?
  2 Mögliche Ausgaben pro Eingabe frei wählbar,
  also 2^4 = 16 mögliche Verknüpfungen

AND, OR, NOT

A	B	**A and B**
0	0	**0**
1	0	**0**
0	1	**0**
1	1	**1**

A	B	**A or B**
0	0	**0**
1	0	**1**
0	1	**1**
1	1	**1**

A	**Not A**
0	**1**
1	**0**

NAND: universelle Verknüpfung

- Technisch einfach zu realisieren
  nur 4 Transistoren
- Alle Verknüpfungen können damit aufgebaut werden.
- not A = A nand A
- A and B =
- A or B =

A	B	**A nand B**
0	0	**1**
1	0	**1**
0	1	**1**
1	1	**0**

NAND: universelle Verknüpfung

- Technisch einfach zu realisieren
  nur 4 Transistoren
- Alle Verknüpfungen können damit aufgebaut werden.
- not A = A nand A
- A and B = not (A nand B) = (A nand B) nand (A nand B)
- A or B = not ((not A) and (not B)) = (not A) nand (not B) = (A nand A) nand (B nand B)

A	B	**A nand B**
0	0	**1**
1	0	**1**
0	1	**1**
1	1	**0**

Halbaddierer (Summe 2er Bits)

- INPUT: 2 Bits A,B
- OUTPUT 2 Bits S und C (Sum and Carry)

- C = A and B
- S =

A	B	**Sum S**	**Carry C**
0	0	**0**	**0**
1	0	**1**	**0**
0	1	**1**	**0**
1	1	**0**	**1**

Halbaddierer (Summe 2er Bits)

- INPUT: 2 Bits A,B
- OUTPUT 2 Bits S und C

A	B	**Sum S**	**Carry C**
0	0	**0**	**0**
1	0	**1**	**0**
0	1	**1**	**0**
1	1	**0**	**1**

- C = A and B
- S = (A or B) and (not (A and B)) = A xor B

XOR (exklusiv oder)

- A oder B, aber nicht beides
- Interessante Operation für Verschlüsselung, weil sie sich selbst rückgängig macht:
- (A xor B) xor B = A
  (A xor B) xor A = B

A	B	**A xor B**
0	0	**0**
1	0	**1**
0	1	**1**
1	1	**0**

Volladdierer, Summe dreier Bits

- Input 3 Bits A, B, C_{in}
- Output Summe S, Carry C_{out}

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

- 2 Halbaddierer plus ein ‘or’ oder
- S = A xor B xor C_{in}
- C_{out} = (A and B) or (A and C) or (B and C)
  = ((A xor B) and C) or (A and B)

Technische Herausforderungen

- Timing
  - Warten, bis das Resultat stabil ist
    - Trotz Terahertz Schaltfrequenz (billionstel Sekunde)
  - Clock (Taktsignal), typischerweise 2GHz
- Adder: Carry Ripple
  - Jeder Volladdierer wartet auf den Übertrag des Vorgängers
  - Gibt raffinierte Lösungen dafür.

Logisim

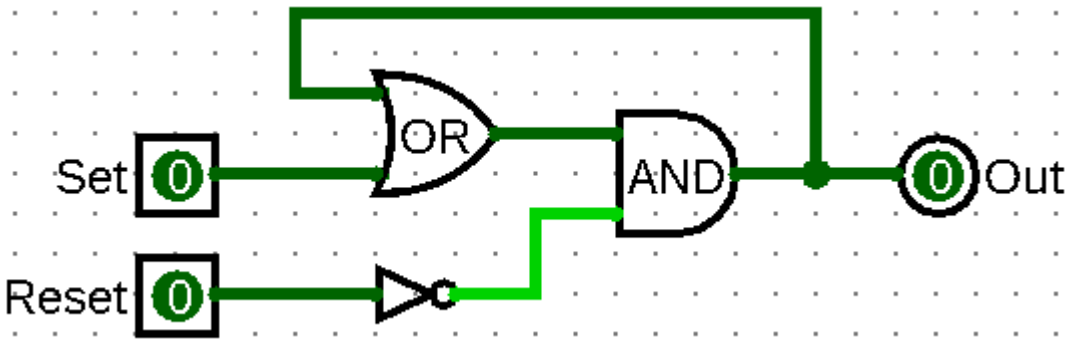
- Software für logische Schaltungen
  - Bau
  - Simulation
- Grundkonzepte verstehen:
  - Adder
  - Speicher
  - Plexer
  - Clock

Vorgehen

- Download von Logisim
- Bedienung und Grundkonzepte von Logisim verstehen.
- Bau eines Halbaddierers
- Bau eines Volladdierers
- Bau eines 4-Bit Addierers

Speicher: SR-Flip-Flop

- Speichern von einzelnen Bits
- Mit Feedback-Loop:



SR-Flip-Flop (Set/Reset)

- Zwei Leitungen, um ein Bit zu setzen.
- Einfacher:
  - Eine Leitung **Data** (was)
  - Eine Leitung **Latch** (wann)
- Wenn Latch 1 ist, wird der Wert von Data gespeichert.

Aufgabe: D-Flip-Flop

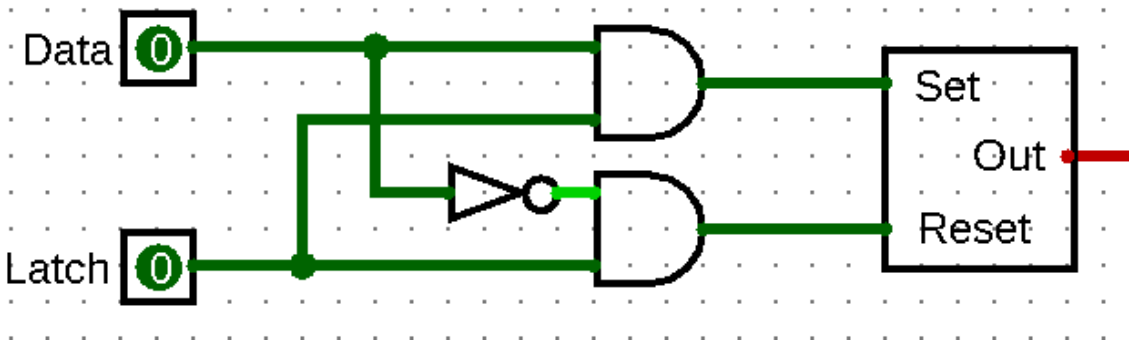
- 2 Eingänge
  - Data D
  - Latch L
- 2 Ausgänge: Set S, Reset R
  - So dass damit ein SR-Flip-Flop gesteuert werden kann.
- Erstellen Sie eine Wahrheitstabelle
- Bauen Sie diese Schaltung in Logisim.

D	L	**S**	**R**
0	0		
1	0		
0	1		
1	1		

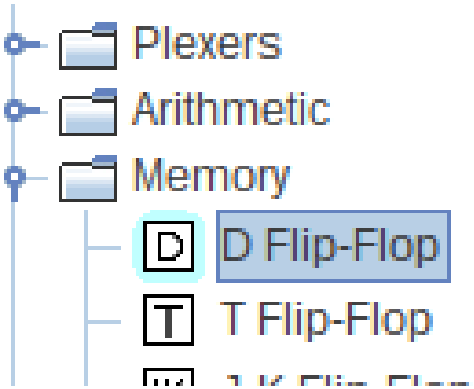
Aufgabe: D-Flip-Flop

- 2 Eingänge
  - Data
  - Latch
- 2 Ausgänge: Set, Reset
  - So dass damit ein SR-Flip-Flop gesteuert werden kann.

D	L	S	R
0	0	**0**	**0**
1	0	**0**	**0**
0	1	**0**	**1**
1	1	**1**	**0**



D-Flip-Flop von Logisim

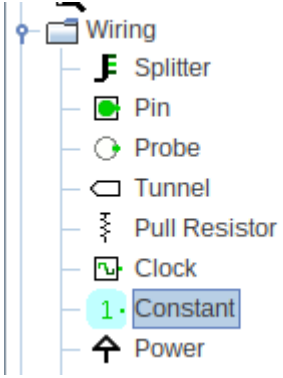


Latch / Clock

Output

Data

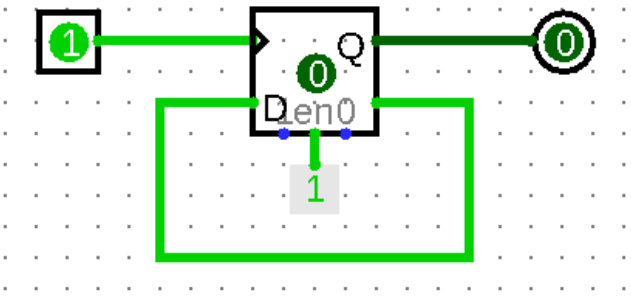
Invertierter Output



Enable (muss 1 sein)

Zähler (Counter)

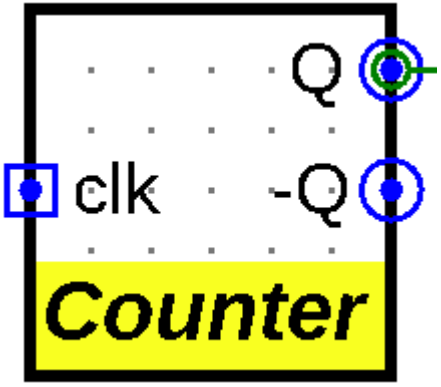
- Immer wenn der Input von 0 auf 1 wechselt
  - Soll der Zähler seinen Zustand ändern.
- Hängt vom Zustand ab.
  - Feedback
  - Problem: Oszillation vermeiden
- Lösung:
  - 2 D-Flip-Flops, die einander den Zustand weiter geben, aber einer wenn der Input von 0 auf 1 wechselt, der andere von 1 auf 0.



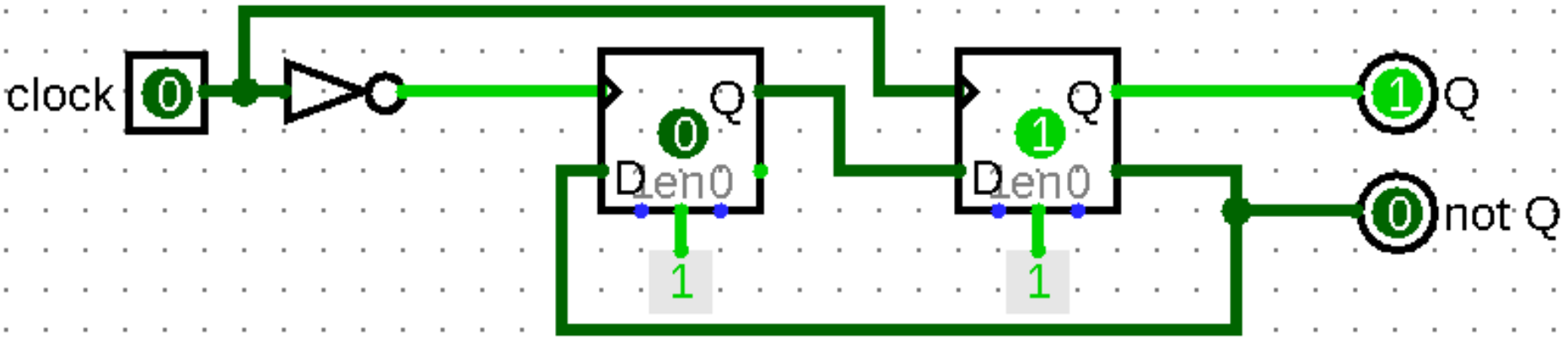
Aufgabe: 1-Bit-Zähler

- Zähler: Eingang Clock, Ausgänge Q, und “not Q”

  Bauen Sie einen 1-Bit Zähler mit 2 D-Flip-Flops und verpacken Sie diesen dann hübsch:



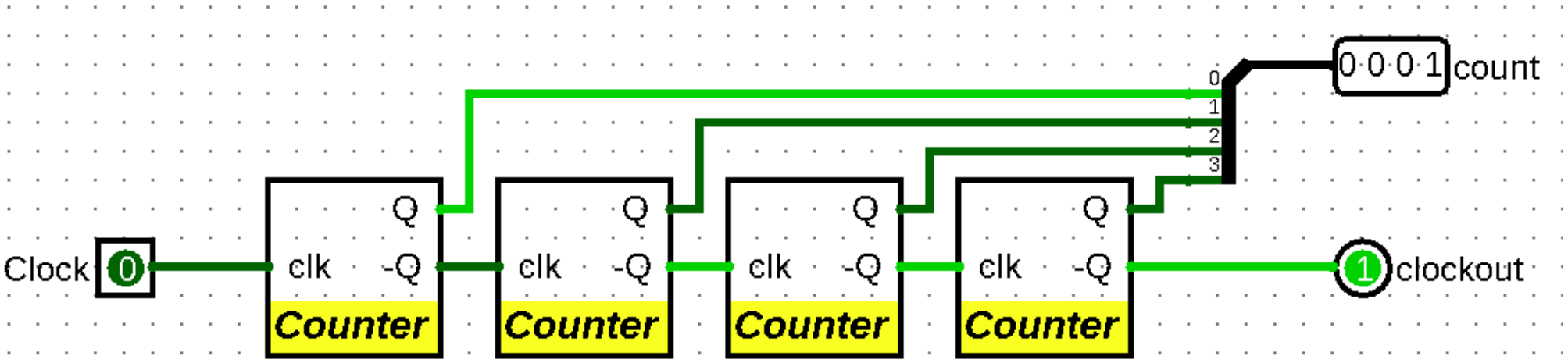
1-Bit Zähler



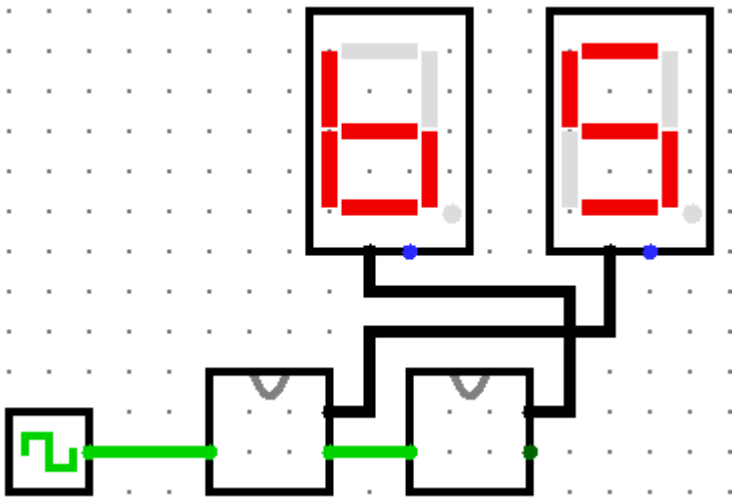
Aufgabe: 4-Bit-Counter

- Bauen Sie einen 4-Bit Counter aus 1-Bit Countern.
- Verwenden Sie ein Clock-Signal als Eingang.
- Vereinigen Sie die 4 Ausgänge mit einem Splitter auf eine einzelne 4-Bit Leitung.
- Schliessen Sie ein Hex-Display daran an.
- Das “not Q” des letzten Counters soll ebenfalls ein Ausgang des 4-Bit Counters sein.
- Bauen Sie daraus ein 8-Bit Counter

4-Bit Counter



8-Bit Counter (mit zwei 4-Bit C.)



Multiplexer, Demultiplexer

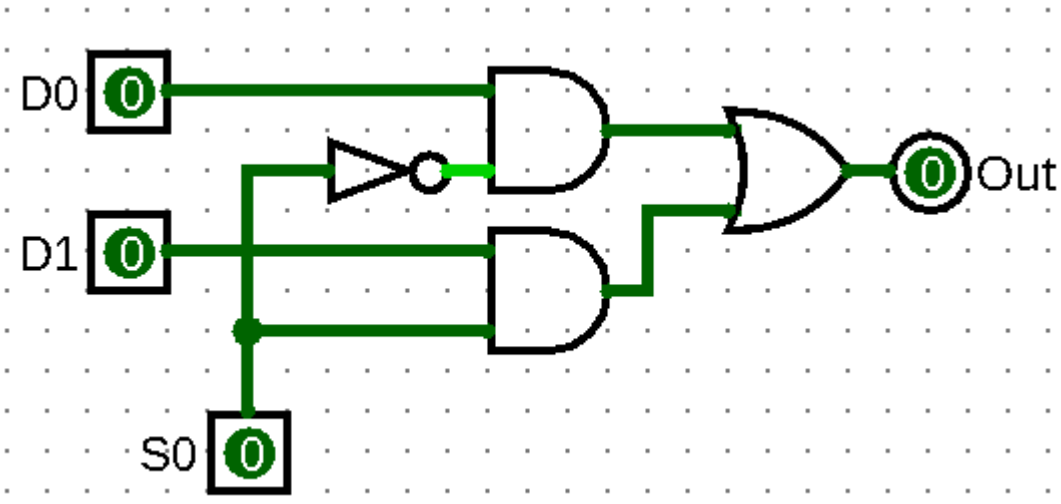
- Multiplexer:
  - 2^n Daten-Eingänge
  - n Selektoren-Eingänge, (Binärzahl s für die Auswahl)
  - 1 Datenausgang (Kopie des Dateneingangs Nummer s)
- Einfachste Variante: n=1
  - Auswahl aus zwei Eingängen

Aufgabe 1-Bit Multiplexer

- 3 Eingänge
  - D0, D1 (Dateneingänge)
  - S0 (Selektor)
- 1 Ausgang O
  - Liefert D0 wenn S0=0
  - Liefert D1 wenn S0=1

Aufgabe 1-Bit Multiplexer

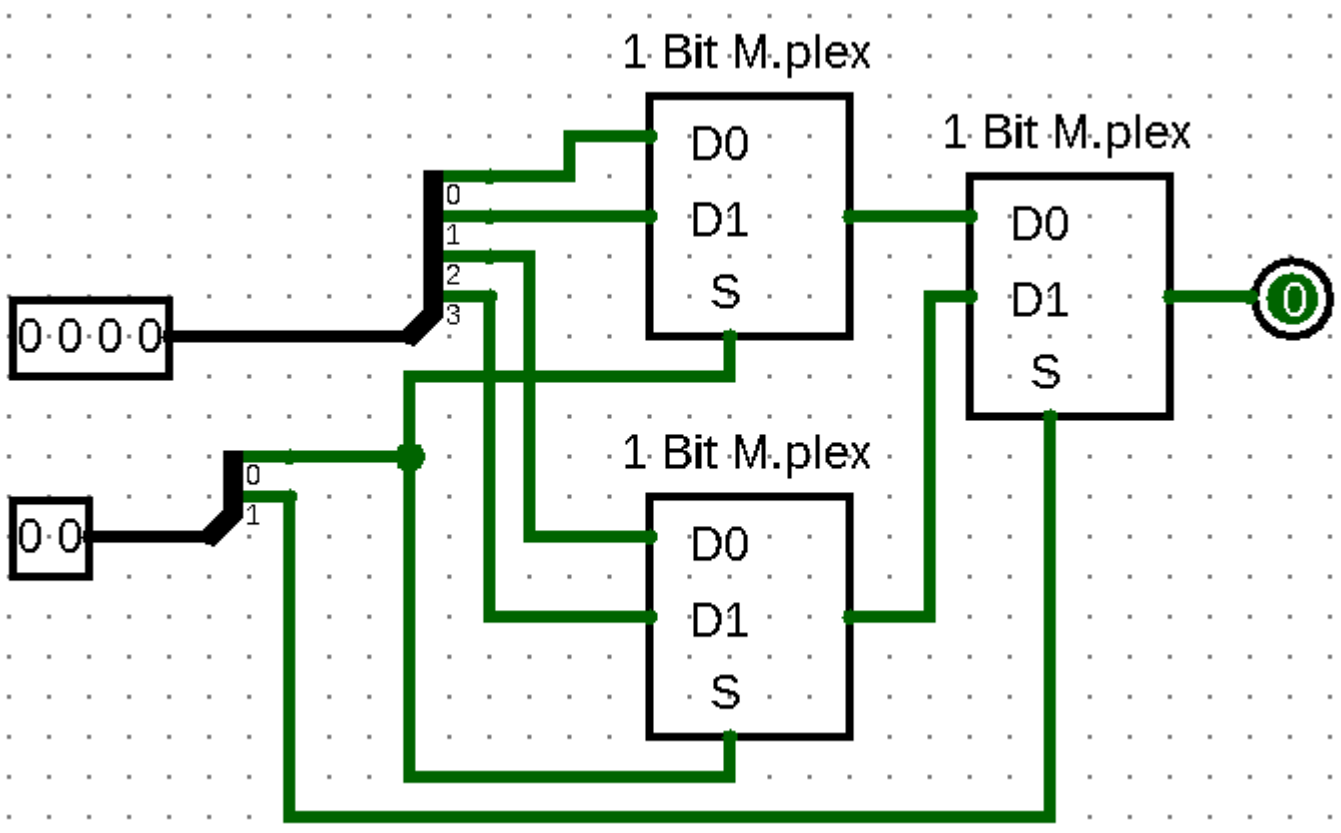
- 3 Eingänge
  - D0, D1 (Dateneingänge)
  - S0 (Selektor)
- 1 Ausgang O
  - Liefert D0 wenn S0=0
  - Liefert D1 wenn S0=1



Zusatzaufgabe 2bit-Multiplexer

- Packen Sie den 1-Bit Multiplexer in einen Subcircuit
- Add circuit…
- Bauen Sie mit drei 1-Bit Multiplexern
  einen 2-Bit Multiplexer.
- Sie können einen Input verwenden, der 4 Bits breit ist.

2-Bit Multiplexer



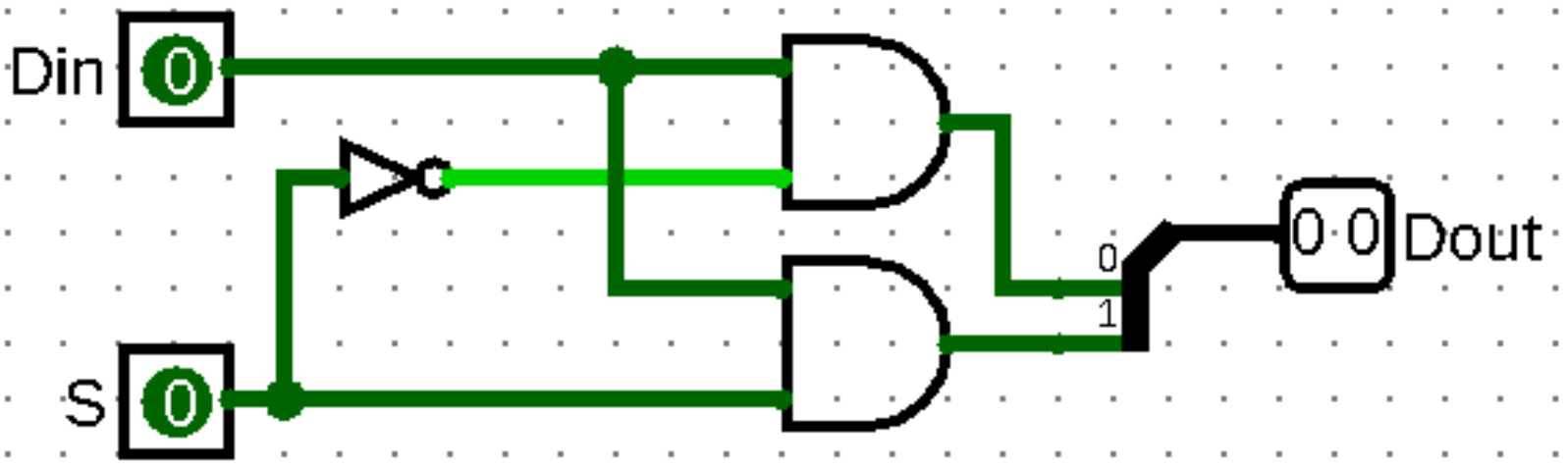
Demultiplexer

- Ein Dateneingang
- 2^n Datenausgänge
- n Selektoreneingänge

Aufgabe: 1-Bit Demultiplexer

- 1 Dateneingang D
- 1 Selektor-Eingang S
- 2 Ausgänge D0, D1, wobei
  - D0 eine Kopie von D wenn S=0, sonst 0
  - D1 eine Kopie von D wenn S=1, sonst 0

Aufgabe: 1-Bit Demultiplexer



Speicher

- Viele einzelne Speicherzellen
- Datenbus (parallele Leitungen, z.B. 64 Bit breit)
- Adressbus (auswahl der Speicherzelle)
- Bytes sind nummeriert
  - via Demultiplexer geschrieben
    - alle Zellen hängen am Datenbus, Adresse wählt Zelle aus
  - via Multiplexer ausgelesen

ALU

- Arithmetic and Logic Unit
- Werte als Input in Registern (“CPU-Speicher”)
- Diverse Operationen (arithmetisch, logisch)
  - “Alle” werden ausgeführt
  - Resultat wird durch OP-Code bestimmt (Multiplexer)
  - OP-Code bestimmt, wohin das Resultat geschrieben wird
    - RAM oder Register

Programmcode im Speicher

- PC (Programm Counter)
  - Register mit Speicheradresse der aktuellen Anweisung
- Fetch-Execute-Cycle
  - OP-Code und Daten aus dem Speicher holen
  - Decodieren, ausführen
  - Resultate evtl. Zurückschreiben
  - PC-Register anpassen