

KNN für XOR-Funktion

G.Döben-Henisch
Fachbereich Informatik und Ingenieurwissenschaften
FH Frankfurt am Main
University of Applied Sciences
D-60318 Frankfurt am Main
Germany
Email: doeben_at_fb2.fh-frankfurt.de

6. April 2009

Zusammenfassung

This is a simple example of an artificial neural network which implements the XOR function. To read more about this have a look to the online text www.uffmm.org/anns.html.

1 Anforderungen

Das Modell soll das Verhalten eines einfachen neuronalen Netzes darstellen, das dem Verhalten der XOR-Funktion entspricht.

2 Systembeschreibung

2.1 Input-Output

Für die XOR-Funktion gilt die folgende Wahrheitstabelle:

IN_X	IN_Y	OUT_{XOR}
1	1	0
1	0	1
0	1	1
0	0	0

Wenn also z.B. an den Eingängen IN_X und IN_Y die Werte '1' und '1' anliegen sollten, dann müßte der Output entsprechend der XOR-Funktion den Wert '0' zeigen. Entsprechend in den anderen Fällen.

2.2 Systemfunktion

Das System besteht in diesem Beispiel aus einem *Netzwerk* von Neuronen. Solch ein Netzwerk kann man verstehen als eine mathematische *Struktur*, die eine *Menge* von Neuronen N , Gewichten W sowie Beziehungen CON zwischen diesen Neuronen und Gewichten umfaßt sowie eine *Operation dyn* über diesen Mengen. Als Formeln:

$$ANN = \langle N, W, CON, dyn \rangle \quad (1)$$

$$CON_{ANN} \subseteq N_{ANN} \times W_{ANN} \times N_{ANN} \quad (2)$$

$$dyn_{ANN} : CON_{ANN} \mapsto CON_{ANN} \quad (3)$$

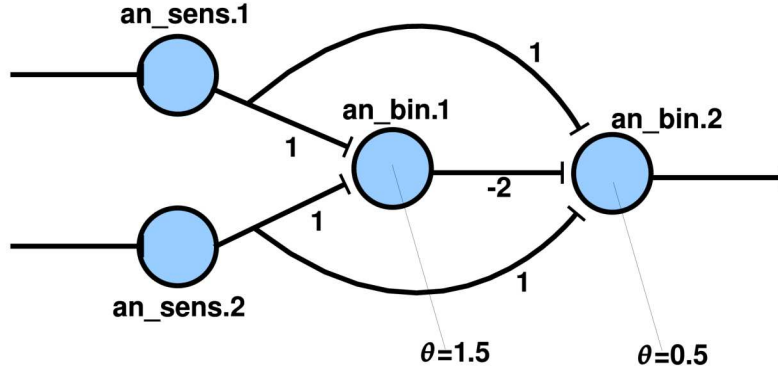


Abbildung 1: Beispiel eines künstlichen neuronalen Netzwerkes mit 2 sensorischen und 2 binären Neuronen

Dieses Schema soll nun zu einem konkreten Netzwerk spezialisiert (instantiiert) werden, wie es im Bild 1 angezeigt wird.

Für die Menge der Neuronen N_{ANN} nehmen wir vier Neuronen an, zwei sensorische Neuronen $an_{sens.i}$ und zwei binäre Neuronen $an_{bin.i}$:

$$N_{ANN} = \{an_{sens.1}, an_{sens.2}, an_{bin.1}, an_{bin.2}\} \quad (4)$$

Dabei soll gelten, $\theta_{bin.1} = 1.5, \theta_{bin.2} = 0.5$. Diese Neuronen verknüpfen wir mittels der Verknüpfungsrelation CON_{ANN} wie folgt:

$$CON_{ANN} = \left\{ \begin{array}{l} \langle an_{sens.1}, 1, an_{bin.1} \rangle \\ \langle an_{sens.1}, 1, an_{bin.2} \rangle \\ \langle an_{sens.2}, 1, an_{bin.1} \rangle \\ \langle an_{sens.2}, 1, an_{bin.2} \rangle \\ \langle an_{bin.1}, -2, an_{bin.2} \rangle \end{array} \right\}$$

Jetzt fehlt nur noch die dynamische Abarbeitungsfunktion dyn_{ANN} . Hier lassen sich viele Varianten denken. Ein einfaches Beispiel wird durch den folgenden Algorithmus dargestellt (vgl. die Formel 6). Die Grundidee besteht darin, daß die Werte des neuronalen Netzes in diesem Fall von *links* nach *rechts* berechnet werden. Liegt ein Ausgangswert vor, wird die Berechnung wieder bei den Eingängen wiederholt. Ein Durchgang entspricht dann einem *Zyklus* innerhalb eines Prozesses bzw. einer Einheit auf einer abstrakten Zeitlinie.

Die Abarbeitungsfunktion dyn_{ANN} wird zusammengesetzt aus mehreren

Teilfunktionen. Die Teilfunktion *take_inputs* beginnt bei den sensorischen Neuronen und bildet daraus eine erste Nachfolgermenge Q . Für jedes Element der Menge Q wird dann jeweils die Neuronfunktion mit der Teilfunktion *compute_each* berechnet. Ausgehend von Q werden mit der Funktion *successors* wieder alle Nachfolger berechnet, also $Q = succ(Q)$ und für diese mit *compute_each* wieder alle Neuronfunktionen $neuron_{TYPE.i}$. Dann wird mit der Teilfunktion *if_out* geprüft, ob die letzten Neuronen schon berechnet worden sind. Mit der Annahme, daß alle Ausgangsneuronen von den Inputneuronen aus alle die gleiche minimale Pfadlänge haben, kann die Berechnung der Menge Q bei minimaler Pfadlänge +1 dann wieder von vorne beginnen, andernfalls muß sie bei der Teilfunktion *successors* fortgesetzt werden.

$$f_{xor} = take_inputs \otimes compute_each \otimes successors \otimes if_out \quad (5)$$

$$take_inputs : INP \mapsto Q = INP \quad (6)$$

$$compute_each : neuron_{type.i} \in Q \quad (7)$$

$$successors : Q \mapsto succ(Q) \quad (8)$$

$$compute_each : neuron_{type.i} \in Q \quad (9)$$

$$if_out : YES : take_inputs; NO : successors \quad (10)$$

Für die Berechnung der einzelnen Neuronen benötigt man die Struktur dieser Neuronen. Für Eingangsneuronen –auch *sensorische* Neuronen genannt– gilt:

$$AN_{sens} = \langle INP, OUTP, \emptyset, \emptyset, neuron_{sens} \rangle \quad (11)$$

Der Aktivierungszustand und der Schwellwert werden hier –stark vereinfachend– als *leere Mengen* \emptyset angenommen, da der Ausgabewert ausschließlich vom Eingabewert abgängen soll:

$$neuron_{sens} : INP \mapsto OUTP \quad (12)$$

dabei wird angenommen, daß die Neuronfunktion $neuron_{sens}$ des sensorischen Neurons im Beispiel eine *Identitätsfunktion* ist, d.h.

$$neuron_{sens}(INP) = OUTP \quad (13)$$

Im allgemeinen Fall kann die Funktion eines sensorischen Neurons natürlich erheblich komplexer strukturiert sein. Für die beiden binären Neuronen in der Zwischenschicht und in der Ausgangsschicht gelten folgende Zusammenhänge (siehe auch das Beispiel über binäre Neuronen):

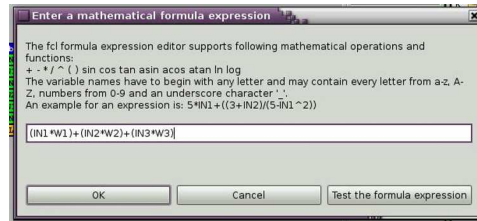


Abbildung 2: Eingabe einer Formel mit dem Formeleditor

$$AN_{bin} = \langle INP, OUP, \{\theta\}, neuron_{bin} \rangle \quad (14)$$

$$neuron_{bin} = net_{bin} \otimes act_{bin} \quad (15)$$

$$net_{bin} = \sum_{i=1}^2 W_i IN_i \quad (16)$$

$$act_{bin} = \begin{cases} 1 & : net_j \geq THETA \\ 0 & : sonst \end{cases} \quad (17)$$

3 Beispiel

Bei der Modellierung des XOR-Netzwerkes wird schrittweise vorgegangen, und zwar *bottom up*, d.h. zuerst werden die *Grundelemente* bereitgestellt, und dann werden aus diesen komplexere Einheiten zusammengebaut.

Im Fall des binären Neurons gibt es zwei Grundelemente: die Summenfunktion *net_bin* –einmal mit zwei, einmal mit 3 Eingängen– und die Aktivierungsfunktion *act_bin*. Alle werden so konstruiert, daß man ein neues Modell eröffnet und dieses unter dem neuen Thema *netxor* abspeichert. Das Bild 2 zeigt, wie man ein kleines Modell auch durch direkte Eingabe einer Formel erzeugen kann. In diesem Fall die Formel für eine Netzsummenfunktion mit drei Eingängen. Das Ergebnis kann man dann im Bild 3 sehen. Allerdings muß man noch die Eingangsparameter *IN1–3*, *W1–2* sowie den Ausgangsparameter *NET_SUM* zusätzlich einfügen.

Das Bild 4 zeigt das FCL-Modell der binären Aktivierungsfunktion. Sie hat als einen Input den Output der Summenfunktion *net_bin* und vergleicht diesen Werte mittels der Funktion *if_geq* (wenn größer oder gleich). Ist der Wert der Summenfunktion größer oder gleich dem Schwellwert, dann wird eine '1' ausgegeben, sonst eine '0'.

Dann kann man das eigentliche Modell des binären Neurons bauen, indem man ein neues Modell beginnt und die vorher gebauten Modelle *net_bin* und *act_bin* als Elemente aus der Bibliothek hereinzieht. Fügt man die entsprechen-

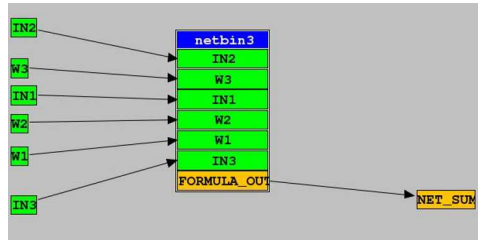


Abbildung 3: Das Ergebnis nach der Eingabe plus Eingangs- und Ausgangsparametern

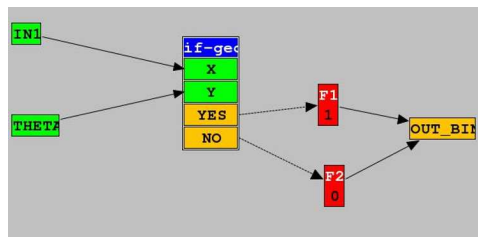


Abbildung 4: Binäre Aktivierungsfunktion

den Input und Outputparameter noch hinzu und verbindet alle Ein- und Ausgänge, dann erhält man ein binäres Neuron mit zwei Eingängen (vgl. das Bild 6 bzw. eines mit 3 Eingängen (vgl. Bild 6)

Die Konstruktion des vollständigen XOR-Netzwerkes ist dann sehr einfach. Man eröffnet ein neues Modell und zieht die beiden binären Neuronen mit den 2 bzw. 3 Eingängen in das Konstruktionsfeld. Dann fügt man die entsprechenden Input- und Outputparameter hinzu und die Konstruktion ist abgeschlossen (siehe Bild 7. Die Gewichte wurden in diesem Fall mittels Konstanten (:= roten Kästchen im Modell) fest verdrahtet. Diese Konstanten könnte man natürlich auch nach Bedarf –z.B. innerhalb von Lernalgorithmen– durch frei programmier-

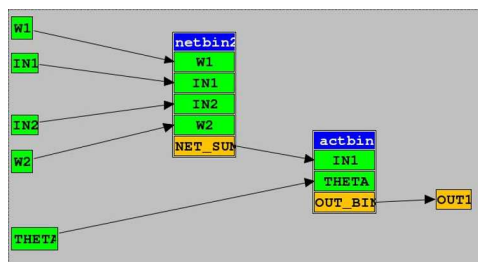


Abbildung 5: Binäres Neuron mit Summen- und Aktivierungsfunktion; 2 Eingänge, 1 Ausgang

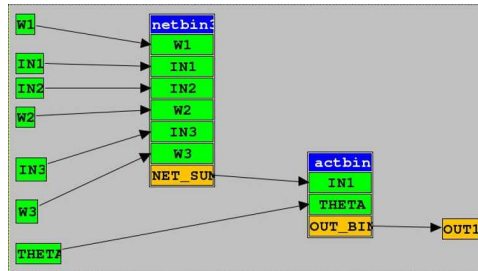


Abbildung 6: Binäres Neuron mit Summen- und Aktivierungsfunktion; 3 Eingänge, 1 Ausgang

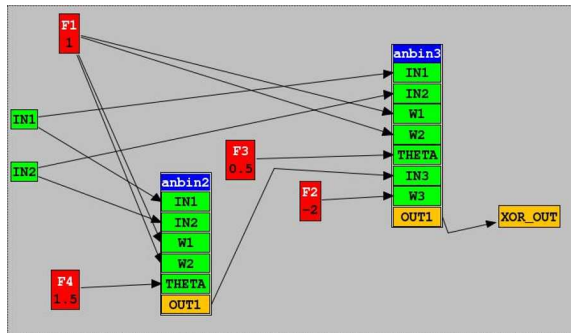


Abbildung 7: Vollständiges XOR-Netzwerk

bare Parameter ersetzen. Dann kann man dieses Netzwerk wieder ganz einfach in die schon bekannte Testumgebung einsetzen (vgl. Bild 8).

Im simulationsmodus kann man sich dann die Konsolenausgabe anzeigen lassen:

```
Sat Mar 21 16:35:11 CET 2009 Editor: 0.8.3 Simulator: ForSYnt V 0.4.2d beta - FCL Message
Start Simulation with id: expnetxor_1
Simulator version: ForSYnt V 0.4.2d beta
Start Time: Saturday, March 21, 2009 4:35:11 PM
```

```
*** ForSYnt V 0.4.2d beta
```

```
$S1_IN1,$CLCK_IN2,$S2_IN3,$OUT_OUT7,$S2_CTRL_OUT6,$S1_CTRL_OUT5,$CLCK_OUT4
0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 0,0,0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0,
1 Output: OUT, contains 1 INT: 0
1 Output: S2_CTRL, contains 1 INT: 0
1 Output: S1_CTRL, contains 1 INT: 0
```

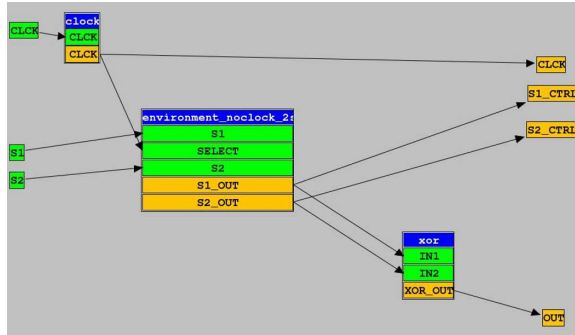


Abbildung 8: Vollständiges XOR-Netzwerk innerhalb einer Testumgebung

```

1 Output:  CLCK, contains 1 INT: 1
2 Input :  S1, contains 20 INT: 0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 0
2 Input :  CLCK, contains 1 INT: 1
2 Input :  S2, contains 20 INT: 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0
2 Output:  OUT, contains 1 INT: 0
2 Output:  S2, contains 1 INT: 0
2 Output:  S1, contains 1 INT: 0
2 Output:  CLCK, contains 1 INT: 2
3 Input :  S1, contains 20 INT: 0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 0
3 Input :  CLCK, contains 1 INT: 2
3 Input :  S2, contains 20 INT: 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0
3 Output:  OUT, contains 1 INT: 0
3 Output:  S2, contains 1 INT: 1
3 Output:  S1, contains 1 INT: 1
3 Output:  CLCK, contains 1 INT: 3
4 Input :  S1, contains 20 INT: 0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 0
4 Input :  CLCK, contains 1 INT: 3
4 Input :  S2, contains 20 INT: 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0
4 Output:  OUT, contains 1 INT: 1
4 Output:  S2, contains 1 INT: 0
4 Output:  S1, contains 1 INT: 1
4 Output:  CLCK, contains 1 INT: 4
5 Input :  S1, contains 20 INT: 0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 0
5 Input :  CLCK, contains 1 INT: 4
5 Input :  S2, contains 20 INT: 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0
5 Output:  OUT, contains 1 INT: 1
5 Output:  S2, contains 1 INT: 1
5 Output:  S1, contains 1 INT: 0
5 Output:  CLCK, contains 1 INT: 5
6 Input :  S1, contains 20 INT: 0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 0
6 Input :  CLCK, contains 1 INT: 5

```

```

6 Input : S2, contains 20 INT: 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0
6 Output: OUT, contains 1 INT: 0
6 Output: S2, contains 1 INT: 0
6 Output: S1, contains 1 INT: 0
6 Output: CLCK, contains 1 INT: 6
7 Input : S1, contains 20 INT: 0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 0
7 Input : CLCK, contains 1 INT: 6
7 Input : S2, contains 20 INT: 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0
7 Output: OUT, contains 1 INT: 0
7 Output: S2, contains 1 INT: 1
7 Output: S1, contains 1 INT: 1
7 Output: CLCK, contains 1 INT: 7
8 Input : S1, contains 20 INT: 0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 0
8 Input : CLCK, contains 1 INT: 7
8 Input : S2, contains 20 INT: 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0
8 Output: OUT, contains 1 INT: 1
8 Output: S2, contains 1 INT: 0
8 Output: S1, contains 1 INT: 1
8 Output: CLCK, contains 1 INT: 8
9 Input : S1, contains 20 INT: 0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 0
9 Input : CLCK, contains 1 INT: 8
9 Input : S2, contains 20 INT: 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0
9 Output: OUT, contains 1 INT: 1
9 Output: S2, contains 1 INT: 1
9 Output: S1, contains 1 INT: 0
9 Output: CLCK, contains 1 INT: 9
10 Input : S1, contains 20 INT: 0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 0
10 Input : CLCK, contains 1 INT: 9
10 Input : S2, contains 20 INT: 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0
10 Output: OUT, contains 1 INT: 0
10 Output: S2, contains 1 INT: 0
10 Output: S1, contains 1 INT: 0
10 Output: CLCK, contains 1 INT: 10
11 Input : S1, contains 20 INT: 0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 0
11 Input : CLCK, contains 1 INT: 10
11 Input : S2, contains 20 INT: 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0
11 Output: OUT, contains 1 INT: 0
11 Output: S2, contains 1 INT: 1
11 Output: S1, contains 1 INT: 1
11 Output: CLCK, contains 1 INT: 11
12 Input : S1, contains 20 INT: 0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 0
12 Input : CLCK, contains 1 INT: 11
12 Input : S2, contains 20 INT: 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0
12 Output: OUT, contains 1 INT: 1
12 Output: S2, contains 1 INT: 0
12 Output: S1, contains 1 INT: 1

```

```

12 Output:  CLCK, contains 1 INT: 12
13 Input :  S1, contains 20 INT: 0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 0
13 Input :  CLCK, contains 1 INT: 12
13 Input :  S2, contains 20 INT: 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0
13 Output:  OUT, contains 1 INT: 1
13 Output:  S2, contains 1 INT: 1
13 Output:  S1, contains 1 INT: 0
13 Output:  CLCK, contains 1 INT: 13
14 Input :  S1, contains 20 INT: 0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 0
14 Input :  CLCK, contains 1 INT: 13
14 Input :  S2, contains 20 INT: 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0
14 Output:  OUT, contains 1 INT: 0
14 Output:  S2, contains 1 INT: 0
14 Output:  S1, contains 1 INT: 0
14 Output:  CLCK, contains 1 INT: 14
15 Input :  S1, contains 20 INT: 0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 0
15 Input :  CLCK, contains 1 INT: 14
15 Input :  S2, contains 20 INT: 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0
15 Output:  OUT, contains 1 INT: 0
15 Output:  S2, contains 1 INT: 1
15 Output:  S1, contains 1 INT: 1
15 Output:  CLCK, contains 1 INT: 15
16 Input :  S1, contains 20 INT: 0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 0
16 Input :  CLCK, contains 1 INT: 15
16 Input :  S2, contains 20 INT: 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0
16 Output:  OUT, contains 1 INT: 1
16 Output:  S2, contains 1 INT: 0
16 Output:  S1, contains 1 INT: 1
16 Output:  CLCK, contains 1 INT: 16
17 Input :  S1, contains 20 INT: 0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 0
17 Input :  CLCK, contains 1 INT: 16
17 Input :  S2, contains 20 INT: 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0
17 Output:  OUT, contains 1 INT: 1
17 Output:  S2, contains 1 INT: 1
17 Output:  S1, contains 1 INT: 0
17 Output:  CLCK, contains 1 INT: 17
18 Input :  S1, contains 20 INT: 0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 0
18 Input :  CLCK, contains 1 INT: 17
18 Input :  S2, contains 20 INT: 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0
18 Output:  OUT, contains 1 INT: 0
18 Output:  S2, contains 1 INT: 0
18 Output:  S1, contains 1 INT: 0
18 Output:  CLCK, contains 1 INT: 18
19 Input :  S1, contains 20 INT: 0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 0
19 Input :  CLCK, contains 1 INT: 18
19 Input :  S2, contains 20 INT: 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0

```

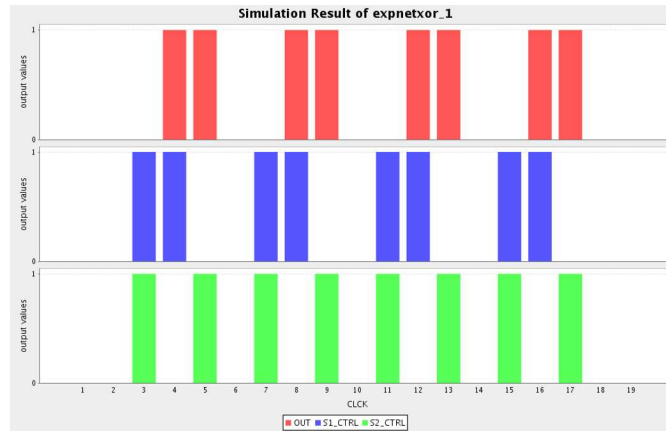


Abbildung 9: Testlauf mit vollständigem XOR-Netzwerk in Umgebung

```

19 Output:  OUT, contains 1 INT: 0
19 Output:  S2, contains 1 INT: 0
19 Output:  S1, contains 1 INT: 0
19 Output:  CLCK, contains 1 INT: 19
20 Input :  S1, contains 20 INT: 0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 0
20 Input :  CLCK, contains 1 INT: 19
20 Input :  S2, contains 20 INT: 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0

..... done

```

```

Sat Mar 21 16:35:11 CET 2009 Editor: 0.8.3 Simulator: ForSYnt V 0.4.2d beta - FCL Message
Simulation with id: expnetxor_1 was finished.
Simulation Time: 0 h 0 m 0 s 70 ms

```

Den gleichen Testlauf kann man sich auch grafisch anzeigen lassen (siehe Bild 9).

4 Quellen

Siehe das Skript unter www.uffmm.org/anns.html.