



Test und Verlässlichkeit

Grosse Übung zu Foliensatz 5

Prof. G. Kemnitz

Institut für Informatik, TU Clausthal (TV_GUeF5)
2. Februar 2017



Software



Aufgabe 5.1: Kontrollflussgraph

C-Programm zur Berechnung des größten gemeinsamen Teilers.

```
int ggt(int x, int y) {
    int c;
n1:   if ( x < 0 )
n2:     x = -x;
n3:   if ( y < 0 )
n4:     y = -y;
n5:   while ( y != 0 ) { // solange y != 0
        // ersetze x durch y und y durch den Rest
        // von x modulo y
n6:     c = x % y; x = y; y = c;
n7:   } // Test Abbruchbedingung
n8:   return x;
}
```

Zeichnen Sie den Kontrollflussgraph.

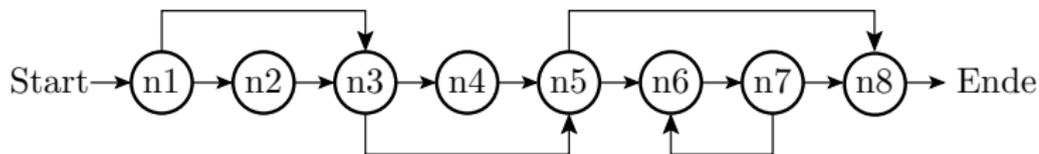


Zur Kontrolle

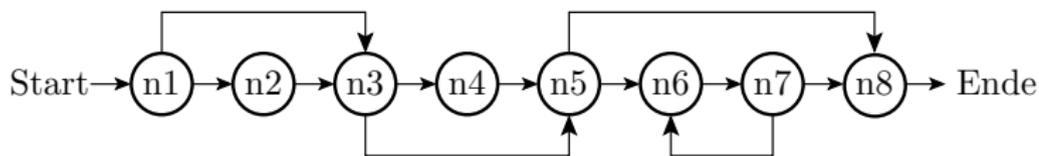
```
int ggt(int x, int y) {  
    int c;  
n1:   if (x < 0)  
n2:     x = -x;  
n3:   if (y < 0)  
n4:     y = -y;  
n5:   while (y != 0){  
n6:     c = x % y; x = y; y = c;}  
n7:   } //Test Abbruchbedingung  
n8:   return x;}  

```

Kontrollflussgraph:



Aufgabe 5.2: Kontrollflussgraph 2



- 1 Bestimmen Sie den Kontrollflussablauf für das Testbeispiel $x = 4, y = 6$. Füllen Sie dazu nachfolgende Tabelle aus ($*$ – von der Anweisung zugewiesenen Werts, sonst frei lassen.).

Anw.	$x^{(*)}$	$y^{(*)}$	Anw.	$x^{(*)}$	$y^{(*)}$
	4	6			
n1					
n3					

- 2 Welche Anweisungs- und Kantenüberdeckung hat der Test?



1. Software

```
int ggt(int x, int y) {
    int c;
n1:  if (x < 0
n2:    x = -x;
n3:  if (y < 0)
n4:    y = -y;
n5:  while (y != 0){
n6:    c = x % y; x = y; y = c;}
n7:  } //Test Abbruchbedingung
n8:  return x;}
```

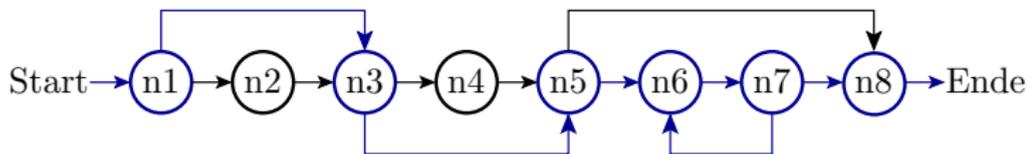
Anw.	$x^{(*)}$	$y^{(*)}$	Anw.	$x^{(*)}$	$y^{(*)}$
	4	6			
n1					
n3					



Zur Kontrolle

Anw.	$x^{(*)}$	$y^{(*)}$	Anw.	$x^{(*)}$	$y^{(*)}$
	4	6	n6	4	2
n1			n7		
n3			n6	2	0
n5			n7		
n6	6	4	n8		
n7			Ende		

- Abgearbeitete Anweisungen und Kanten:

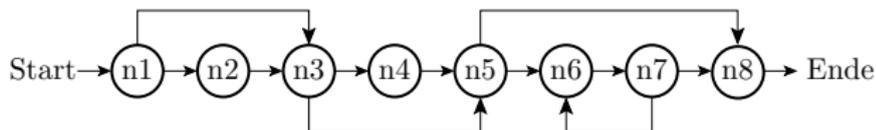


- Anweisungsüberdeckung: 75% (6 von 8 Anweisungen).
- Kantenüberdeckung: $\approx 55\%$ (6 von 11 ohne Start und Ende).



Aufgabe 5.3: Kontrollflussgraph 3

```
int ggt(int x, int y) {int c;  
n1:  if (x < 0  
n2:    x = -x;  
n3:  if (y < 0)  
n4:    y = -y;  
n5:  while (y != 0){  
n6:    c = x % y; x = y; y = c;}  
n7:  } //Test Abbruchbedingung  
n8:  return x;}
```



- 1 Konstruieren Sie ein Testbeispiel, dass jede Anweisung in der Abfolge n1-n2-n3-n4-n5-n6-n7 genau einmal abarbeitet.
- 2 Was ist der größte gemeinsame Teiler von $x = 6$, $y = 0$ und was berechnet das Programm dafür.



Zur Kontrolle

- 1 Für ein Testbeispiel, dass jede Anweisung in der Abfolge n1-n2-n3-n4-n5-n6-n7 genau einmal abarbeitet, muss gelten $x < 0$, $y < 0$ und $x \% y \Rightarrow 0$, z.B. $x = -4$ und $y = -2$ (Programmablauf siehe Tabelle).

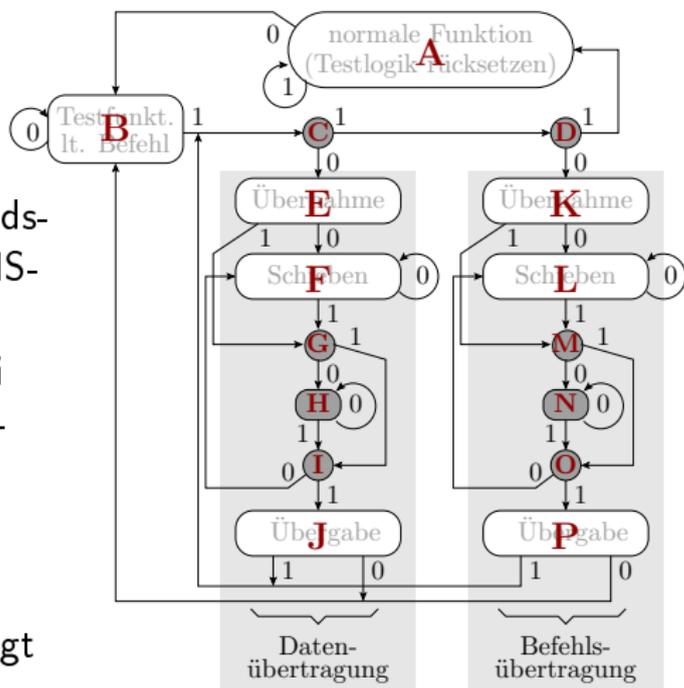
Anw.	Wert x	Wert y	Anw.	Wert x	Wert y
	-4	-2	n5		
n1			n6	2	0
n2	4		n7		
n3			n8		
n4		2	Ende		

- 2 Es gibt keinen größten gemeinsame Teiler von $x = 6$, $y = 0$ und das Programm berechnet den Wert 0. Aus dem Kontrollfluss ist nicht ablesbar, dass die Kontrolle der Wertebereiche für die Eingabe unvollständig ist.

Aufgabe 5.5: Automatentest TAP-Controlle

Spezifizieren Sie eine Zustandsfolge und die zugehörige TMS-Folge (TMS – Übergangsbedingung an den Kanten), bei der alle 32 Kanten des Automaten mindestens einmal durchlaufen werden.

Die Initialisierung aus einem beliebigen Startzustand erfolgt mit $6 \times TMS = 1$.





Zur Kontrolle

Z	?	?	?	?	?	?	A	A
TMS	1	1	1	1	1	1	1	0

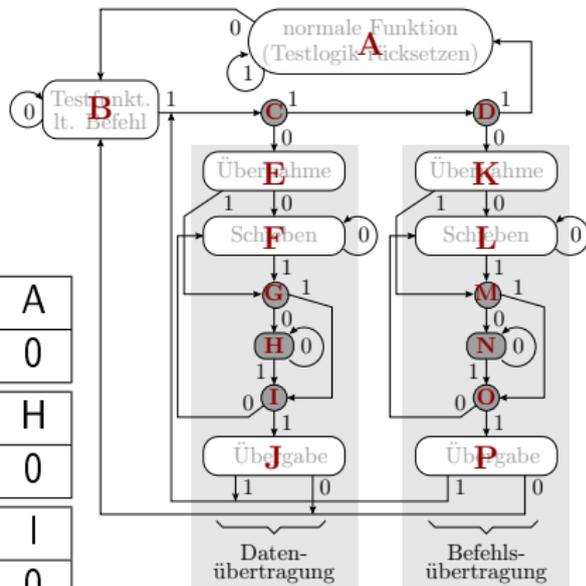
Z	B	B	C	E	F	F	G	H
TMS	0	1	0	0	0	1	0	0

Z	H	I	J	B	C	E	G	I
TMS	1	1	0	1	0	1	1	0

Z	E	G	I	J	C	D	K	L	L	M	N	N	O	P
TMS	1	1	1	0	1	0	0	0	1	0	0	1	1	0

Z	B	C	D	K	M	O	L	M	O	P	C	D	A	
TMS	1	1	0	1	1	0	1	1	1	1	1	1		

(farbig – wiederholt durchlaufene Kanten).





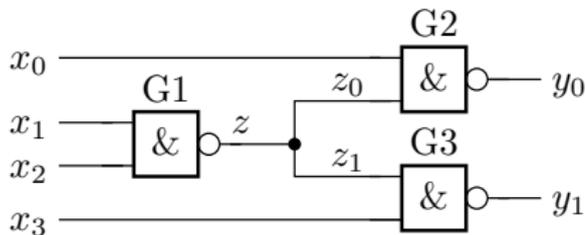
Schaltkreistest



Haftfehler

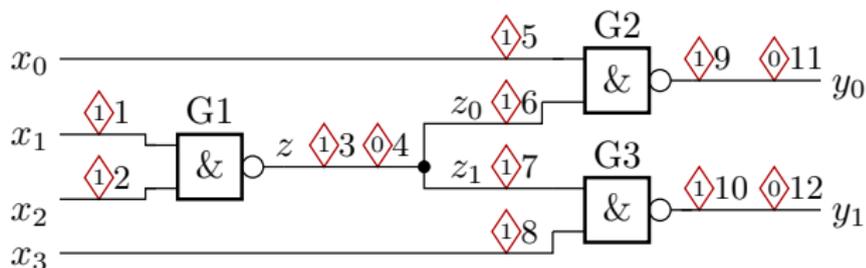
Aufgabe 5.6: Haftfehlermenge

Stellen Sie für die nachfolgende Schaltung die Menge aller Haftfehler auf. Bei mehreren identisch nachweisbaren Haftfehlern ist jeweils nur einer in die Fehlermenge aufzunehmen.



Hinweis: Notation der Haftfehler $sa0(\langle \text{Signalname} \rangle)$ bzw. $sa1(\langle \text{Signalname} \rangle)$.

Zur Kontrolle

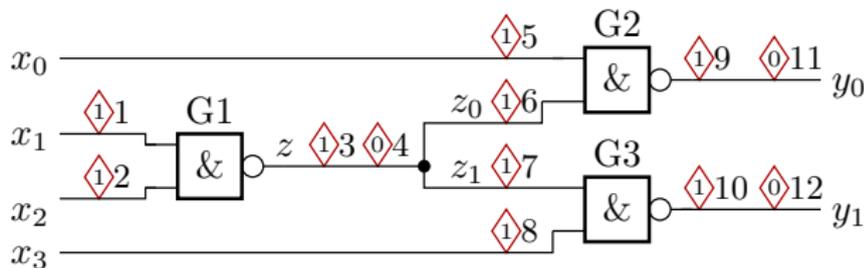


Liste der Modellfehler:

Nr.	0	1	2	3	4	5
Fehler	sa1(x_0)	sa1(x_1)	sa1(x_2)	sa1(x_3)	sa0(z)	sa1(z)
Nr.	6	7	8	9	10	11
Fehler	sa1(z_0)	sa1(z_1)	sa0(y_0)	sa1(y_0)	sa0(y_1)	sa1(y_1)

Aufgabe 5.7: Fehlersimulation

Schreiben Sie ein C-Programm zur fehlerparallelen Simulation der Schaltung aus der Aufgabe zuvor. Gutsimulation in Bit 0, Simulation der Fehler in den den Fehlern zugeordneten Bits 1 bis 12:

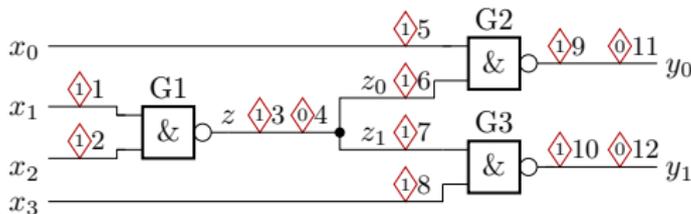


Programmrahmen:

```
uint16_t x0, x1, x2, x3, z, z0, z1, y0, y1;
<wiederhole für alle 16 Eingabemöglichkeiten>{
  <Simulation der Gatter und Fehler>
}
```



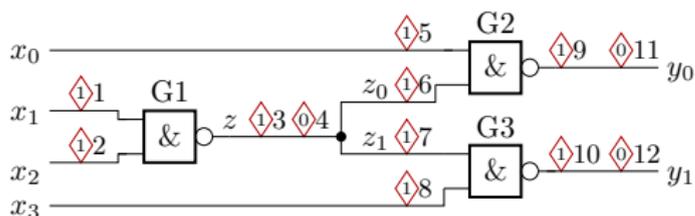
Zur Kontrolle



```

uint16_t x0, x1, x2, x3, z, z0, z1, y0, y1;
for (x3=0; x3==0xFF; ~x3){
  for (x2=0; x2==0xFF; ~x2){
    for (x1=0; x1==0xFF; ~x1){
      for (x0=0; x0==0xFF; ~x0){
        x0 = x0 | 1<<5;      // F5: sa1(x0)
        x1 = x1 | 1<<1;      // F1: sa1(x1)
        x2 = x2 | 1<<2;      // F2: sa1(x2)
        x3 = x3 | 1<<8;      // F8: sa1(x3)
        z  = ~(x1 & x2);     // Gatter G1
        z  = z | 1<<3;       // F3: sa1(z)
        z  = z & ~(1<<4);    // F4: sa0(z)
        z0 = z | 1<<6;       // F6: sa1(z0)
        z1 = z | 1<<7;       // F8: sa1(z1)
      }
    }
  }
}

```



```

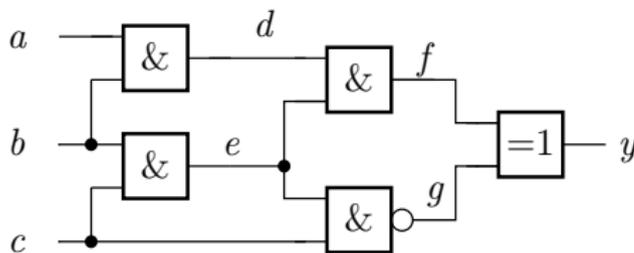
for (x3=0; x3==0xFF; ~x3){
  for (x2=0; x2==0xFF; ~x2){
    for (x1=0; x1==0xFF; ~x1){
      for (x0=0; x0==0xFF; ~x0){
        ...
        z0 = z | 1<<6;      // F6: sa1(z0)
        z1 = z | 1<<7;      // F8: sa1(z1)
        y0 = ~(x0 & z0);    // Gatter G2
        y0 = y0 | 1<<9;     // F9: sa1(y0)
        y0 = y0 & ~(1<<11); // F11: sa0(y0)
        y1 = ~(z1 & x3);    // Gatter G3
        y1 = y1 | 1<<10;    // F10: sa1(y1)
        y1 = y1 & ~(1<<12); // F12: sa0(y1)
      }
    }
  }
}

```



Testberechnung

Aufgabe 5.8: D-Algorithmus¹

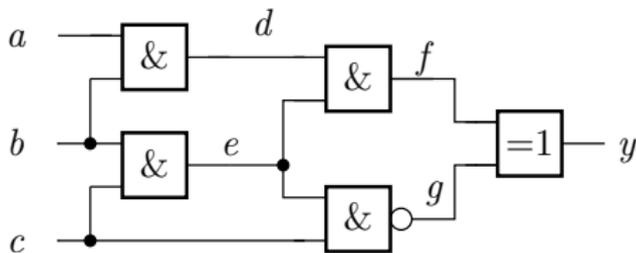


- 1 Geben Sie alle Möglichkeiten für die Sensibilisierung eines D-Pfads von Eingang c zum Ausgang y an.
- 2 Suchen Sie für den Haftfehler $sa1(a)$ einen Test mit dem D-Pfad $a \rightarrow d \rightarrow f \rightarrow y$.

Kennzeichnung der Wertefestlegungen: F – lokale Fehlernachweisbedingung; I – implizite Festlegung; E – Entscheidung; \bar{E} – invertierte Entscheidung; W – Widerspruch.

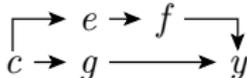
¹Aus http://www.eda.ei.tum.de/fileadmin/tueieda/www/EI-BSc/EDS/tutorium/Tutorial_Dalgorithmus.pdf

Zur Kontrolle

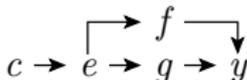


Aufgabenteil 1: alle Pfade

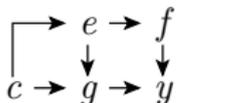
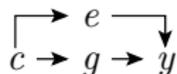
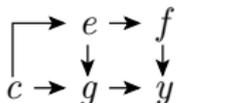
$c \rightarrow g \rightarrow y$



$c \rightarrow e \rightarrow g \rightarrow y$



$c \rightarrow e \rightarrow f \rightarrow y$

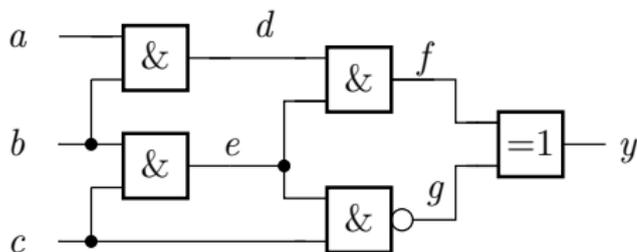


Aufgabenteil 2:
 Testsuche für
 sa1(a), D-Pfad
 $a - d - f - y$

$a = D$	F
$b = 1$	I
$d = D$	I
$e = 1$	I
$f = D$	I
$c = 1$	I
$g = 0$	I
$d = D$	I

$(a, b, c) = (1, 1, 1)$

Aufgabe 5.9: D-Algorithmus Fortsetzung

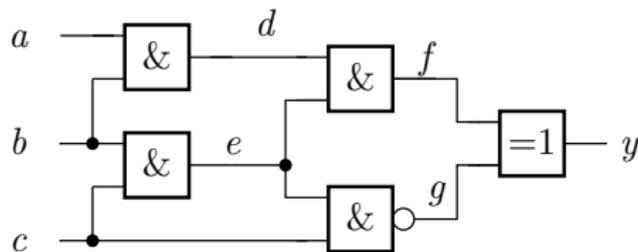


- 3 Suchen Sie für den Haftfehler $sa0(e)$ einen Test mit dem D-Pfad $e \rightarrow g \rightarrow y$.
- 4 Suchen Sie für den Haftfehler $sa0(c)$ einen Test mit dem D-Pfad $c \rightarrow e \rightarrow f \rightarrow y$.

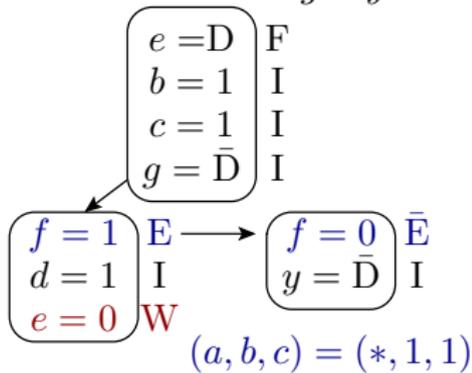
Kennzeichnung der Wertefestlegungen: F – lokale Fehlernachweisbedingung; I – implizite Festlegung; E – Entscheidung; \bar{E} – invertierte Entscheidung; W – Widerspruch.



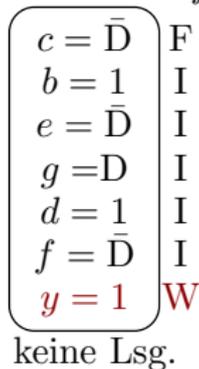
Zur Kontrolle



Aufgabenteil 3: sa1(e)
über D-Pfad $e - g - y$

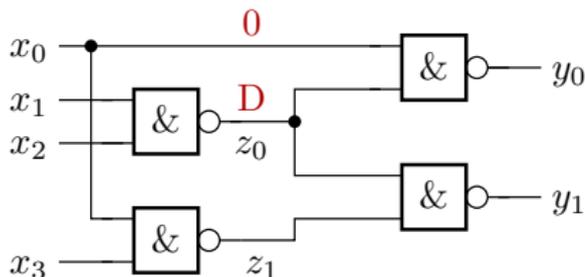


Aufgabenteil 4: sa0(c)
über D-Pfad $c - e - f - y$

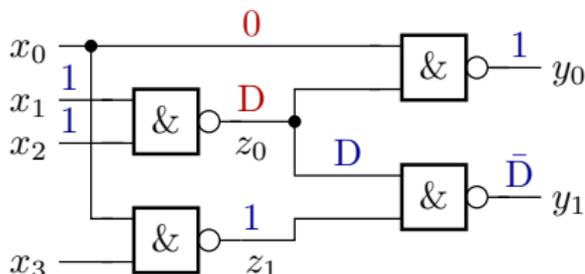


Aufgabe 5.10: Implikationstest

Bestimmen Sie für die nachfolgende Schaltung mit den beiden Signalfestlegungen (einmal »0« und einmal »D«) alle damit implizit festgelegten Signalwerte.



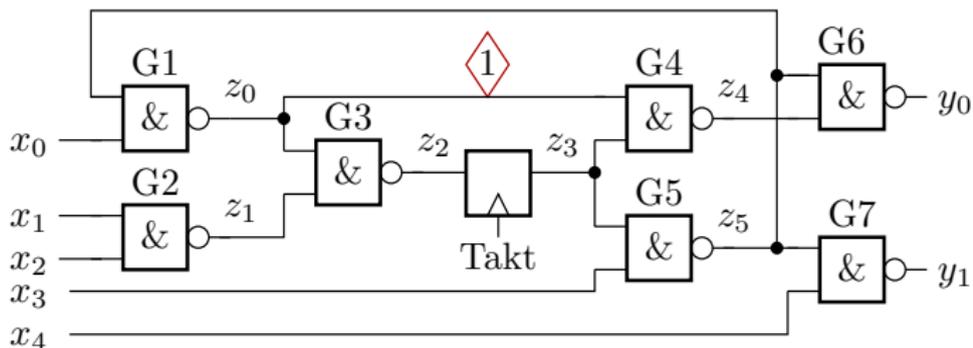
Zur Kontrolle



- Die vorgegebene »0« verbietet Weiterführung des D-Pfads nach y_0 .
- Eindeutiger D-Pfad nach y_1 .
- $z_0 = D$ impliziert $x_1 = x_2 = 1$.
- $x_0 = 0$ impliziert $y_0 = 1$ und $z_1 = 1$

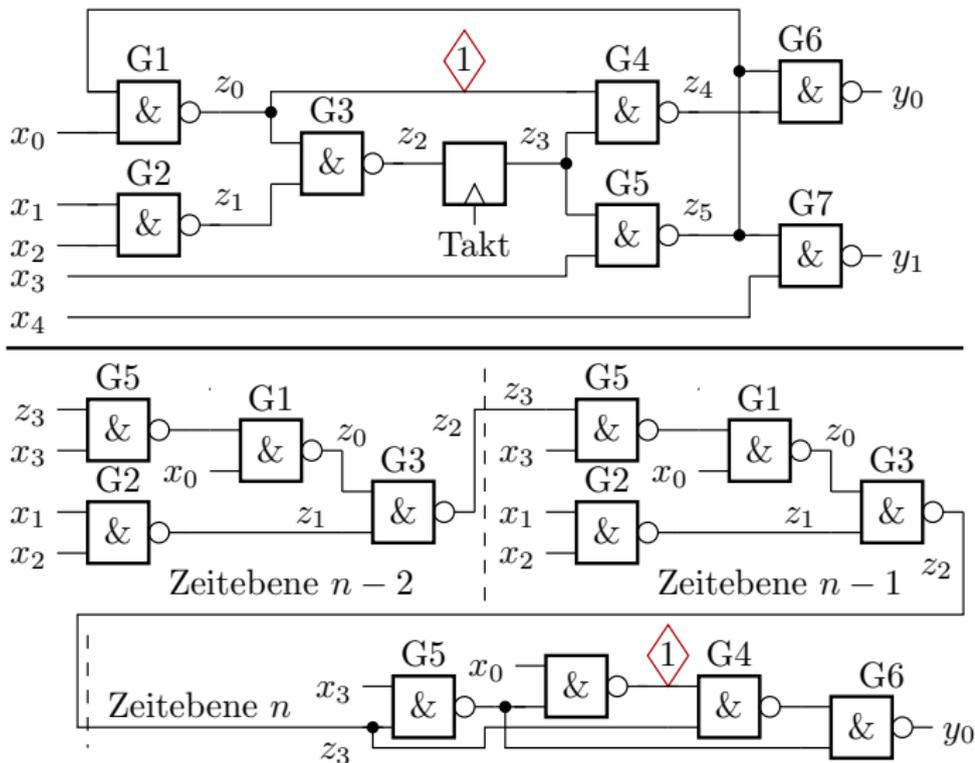
Aufgabe 5.11: Kombinatorische Ersatzschaltung

Rollen Sie die nachfolgende Schaltung zu einer kombinatorischen Ersatzschaltung für die Testberechnung des eingezeichneten Haftfehlers auf mit einer Begrenzung der Länge der Steuerspfade auf max. drei Zeitebenen (max. 3 Schaltungskopien).





Zur Kontrolle

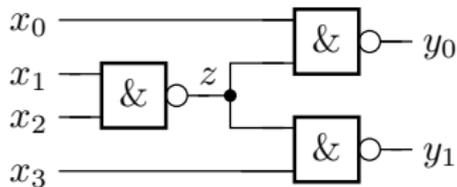




Andere Fehlermodelle

Aufgabe 5.12: Toogle-Überdeckung

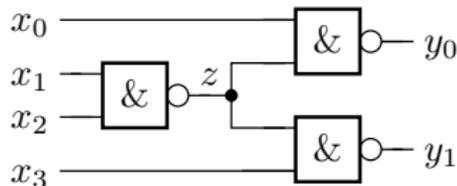
- 1 Kontrollieren Sie für die nachfolgende Schaltung und den angegebenen Testsatz, welche Signale noch nicht mindestens einmal den Wert null und einmal den Wert eins annehmen.
- 2 Erweitern Sie den Testsatz um zusätzliche Eingaben so, dass eine 100%ige Toggle-Test-Überdeckung erzielt wird.



x_3	x_2	x_1	x_0	z	y_1	y_0
0	1	1	0			
1	1	0	0			
1	1	1	0			

Hinweis: Verwenden Sie als Hilfsmittel die Wertetabelle rechts.

Zur Kontrolle



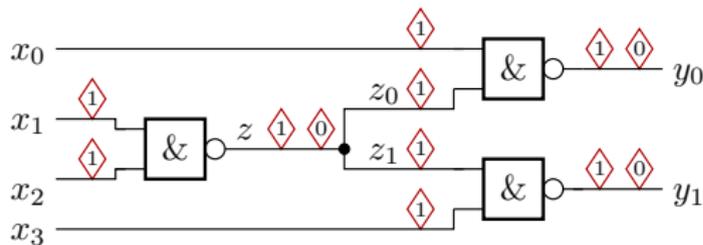
x_3	x_2	x_1	x_0	z	y_1	y_0
0	1	1	0	0	1	1
1	1	0	0	1	0	1
1	1	1	0	1	0	1
1	0	0	1	1	0	0

- 1 Mit den ersten drei Testbeispielen sind nicht abgedeckt: $x_0 = 2$, $x_2 = 0$ und $y_0 = 0$
- 2 Für 100% Toggle-Überdeckung genügt ein weiterer Test:

$$x_3x_2x_1x_0 = 1001$$

Aufgabe 5.13: Haftfehlerüberdeckung Toggle-Test

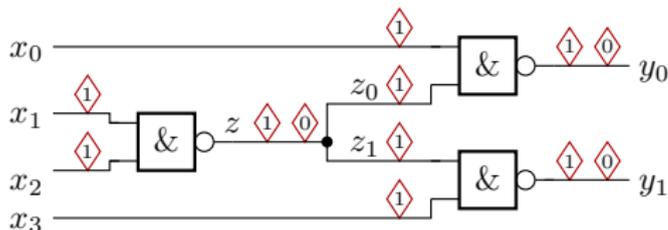
- 1 Bestimmen Sie für die Eingaben des Toggle-Tests aus der Aufgabe zuvor, welche der eingezeichneten Haftfehler nachgewiesen werden.
- 2 Wie groß ist die Haftfehlerüberdeckung des Toggle-Tests?



x_3	x_2	x_1	x_0	z	y_1	y_0	sa1(x_0)	sa1(x_1)	sa1(x_2)	sa1(x_3)	sa1(z)	sa0(z)	sa1(z_0)	sa1(z_1)	sa1(y_0)	sa0(y_0)	sa1(y_1)	sa0(y_1)
0	1	1	0															
1	1	0	0															
1	1	1	0															

Zur Kontrolle

- 1 Kennzeichnung der nachweisbaren Haftfehler:



x_3	x_2	x_1	x_0	z	y_1	y_0	sal(x_0)	sal(x_1)	sal(x_2)	sal(x_3)	sal(z)	sa0(z)	sal(z_0)	sal(z_1)	sal(y_0)	sa0(y_0)	sal(y_1)	sa0(y_1)
0	1	1	0	0	1	1										×		×
1	1	0	0	1	0	1	×	×		×		×				×	×	
1	1	1	0	0	1	1				×	×			×		×	×	
1	0	0	1	1	0	0	×	×	×	×		×	×	×	×		×	

- 2 Haftfehlerüberdeckung: 100%



Aufgabe 5.15: Gatterverzögerungsfehler

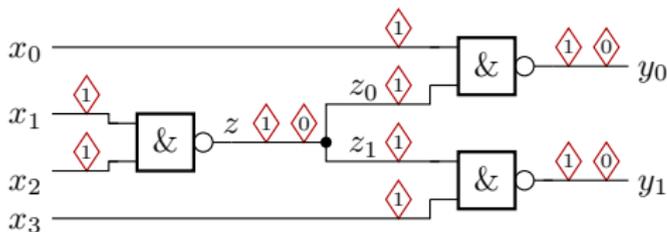
- 1 Für welche der Haftfehler in der Aufgabe zuvor wird der korrespondierende Gatterverzögerungsfehler nachgewiesen.
- 2 Wie groß ist die Gatterverzögerungsfehlerüberdeckung des betrachteten Toggle-Tests?

Hinweis: Der korrespondierende Gatterverzögerungsfehler zu »stuck-at-0« ist »slow-to-rise« und zu »stuck-at-1« »slow-to-fall«.
Zusätzliche Nachweisbedingung ist ein Signalwechsel am Fehlerort.

Zur Kontrolle

- 1 Nachweisbare Verzögerungsfehler farbiger unterlegt:

zusätzliche Änderung des Wertes am Fehlerort



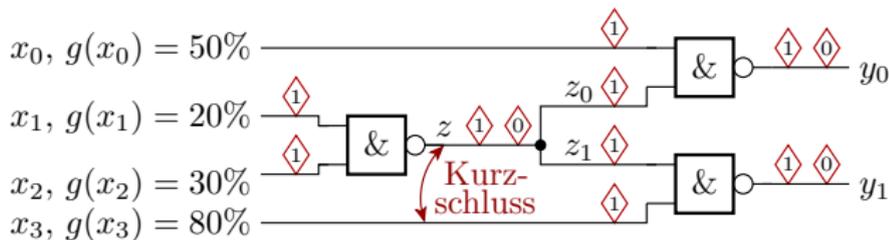
x_3	x_2	x_1	x_0	z	y_1	y_0	sa1(x_0)	sa1(x_1)	sa1(x_2)	sa1(x_3)	sa1(z)	sa0(z)	sa1(z_0)	sa1(z_1)	sa1(y_0)	sa0(y_0)	sa1(y_1)	sa0(y_1)	
0	1	1	0	0	1	1										×			×
1	1	0	0	1	0	1	×	×		×		×				×	×	×	
1	1	1	0	0	1	1				×	×			×		×	×	×	
1	0	0	1	1	0	0	×	×	×	×		×	×	×	×		×		

(sa0 \Rightarrow »slow-to-rise«; sa1 \Rightarrow »slow-to-fall«)

- 2 Überdeckung für Verzögerungsfehler: $\frac{10}{12} \approx 83\%$

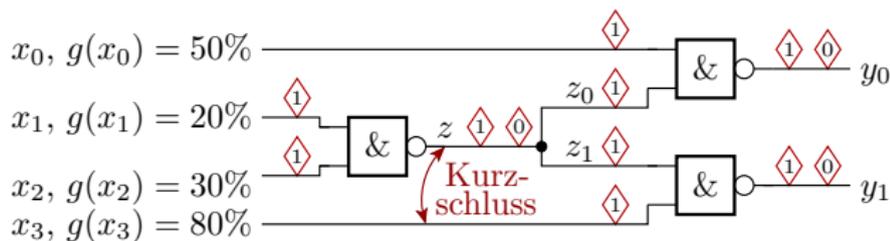
Aufgabe 5.16: Kurzschlussnachweis

Bei dem eingezeichneten Kurzschluss soll sich null durchsetzen.
 Welche der eingezeichneten Haftfehler teilen sich Nachweisbedingungen mit dem Kurzschluss und wie groß ist für jeden dieser Haftfehler die Wahrscheinlichkeit, dass wenn ein Test ihn nachweist, auch der Kurzschluss nachgewiesen wird?



Hinweis: $g(\dots)$ – Signalwichtungen, Auftretshäufigkeit einer Eins. Die bedingte Wahrscheinlichkeit, dass ein Kurzschluss von einem Haftfehlertest nachgewiesen wird, ist die Wichtung $g(\dots)$ der anderen beteiligten Leitung oder deren Gegenwahrscheinlichkeit.

Zur Kontrolle



Der Kurzschluss »0 setzt sich durch« teilt sich mit den Haftfehlern $sa_0(x_3)$ und $sa_0(z)$ Anregungs- und Beobachtungsbedingungen.
 Bedingte Wahrscheinlichkeiten für den Kurzschlussnachweis, wenn einer dieser Haftfehler nachgewiesen wird:

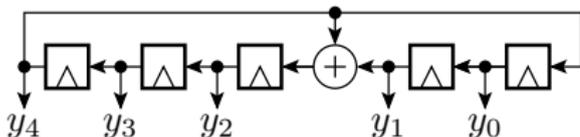
- $sa_0(x_3)$, wenn $z = 0$. $P(z = 0) = g(x_1) \cdot g(x_2) = 6\%$
- $sa_0(z)$, wenn $x_3 = 0$. $P(x_3 = 0) = 1 - g(x_3) = 20\%$



Selbsttest

Aufgabe 5.17: LFSR-Zykluslänge

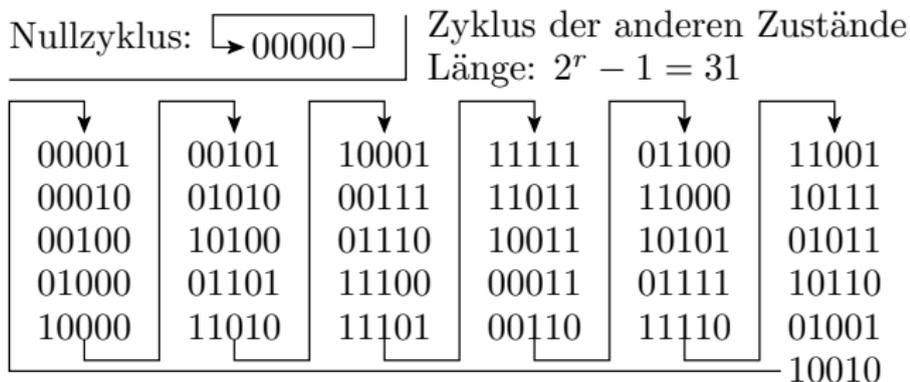
Untersuchen Sie für das nachfolgende 5-Bit linear rückgekoppelte Schieberegister die Zyklusstruktur.



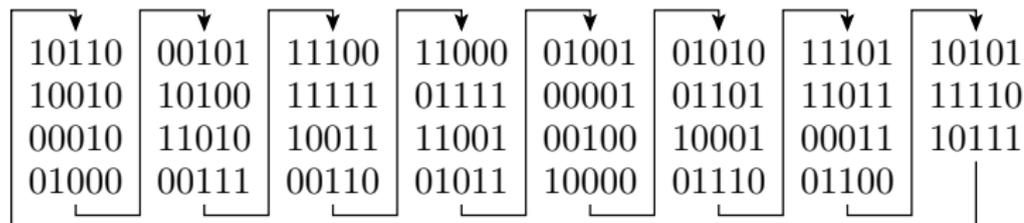
- 1 Bestimmen Sie für jeden der 32 möglichen Zustände alle erreichbaren Folgezustände.
- 2 Wie viele unterschiedliche Testeingaben lassen sich maximal hintereinander erzeugen?
- 3 Wie lautet die zyklisch generierte Testeingabefolge, wenn der Generator mit $y_4y_3y_2y_1y_0 = 01011$ initialisiert und jeder zweite Zustand als Testeingabe verwendet wird?



Zur Kontrolle

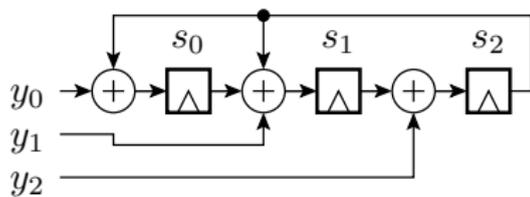


Jede zweite Zustand ab 01011:



Aufgabe 5.18: Signaturregister

Gegeben ist das nachfolgende Signaturregister, eine Sollausgabefolge von einem fehlerfreien Testobjekt und die verfälschte Ausgabefolge eines fehlerhaften Testobjekts:



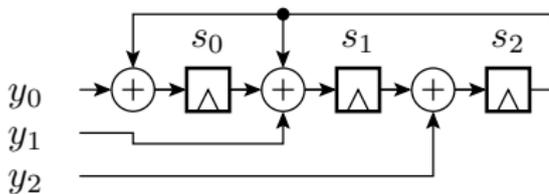
korrekte Folge			fehlerhafte F.		
y_0	y_1	y_2	y_0	y_1	y_2
1	0	1	1	1	1
0	0	1	0	0	1
1	1	0	0	1	0
0	0	0	0	0	0

- Wie groß ist die Wahrscheinlichkeit, dass das Signaturregister für eine verfälschte Datenfolge ein anderes Prüfkenzeichen als für die korrekte Datenfolge berechnet?
- Welche Signatur (Prüfkenzeichen) hat die Sollausgabefolge und welche die fehlerhafte Ausgabefolge?
- Kontrollieren Sie am Beispiel, dass die Signatur der Differenzfolge gleich der Differenz der Signaturen ist (Überlag.-Satz).

1 Maskierungswahrscheinlichkeit:

$$p_M =$$

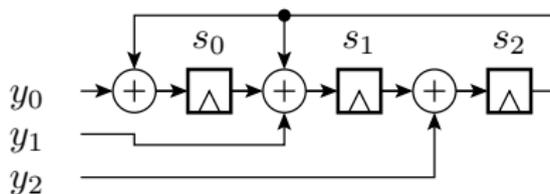
2 Signatur der Soll- und der Fehlerfolge:



	korrekte Folge			Sollsignatur			fehlerhafte F.			Fehlersign.		
Schritt	y_0	y_1	y_2	s_0	s_1	s_2	y_0	y_1	y_2	s_0	s_1	s_2
0	1	0	1	0	0	0	1	1	1	0	0	0
1	0	0	1				0	0	1			
2	1	1	0				0	1	0			
3	0	0	0				0	0	0			
Endw.												

Zur Kontrolle

- 1 Maskierungswahrscheinlichkeit: $p_M = 2^{-3} = 0,125$
- 2 Signatur der Soll- und der Fehlerfolge:

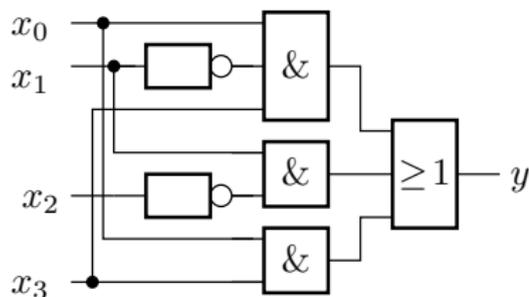


Schritt	korrekte Folge			Sollsignatur			fehlerhafte F.			Fehlersign.		
	y_0	y_1	y_2	s_0	s_1	s_2	y_0	y_1	y_2	s_0	s_1	s_2
0	1	0	1	0	0	0	1	1	1	0	0	0
1	0	0	1	1	0	1	0	0	1	1	1	1
2	1	1	0	0	1	1	0	1	0	0	1	0
3	0	0	0	0	0	1	0	0	0	0	1	1
Endw.				1	1	0				1	1	1

- 3 Differenzen rot unterlegt.

Aufgabe 5.19: Selbsttest

Testobjekt sei die nachfolgende Schaltung:



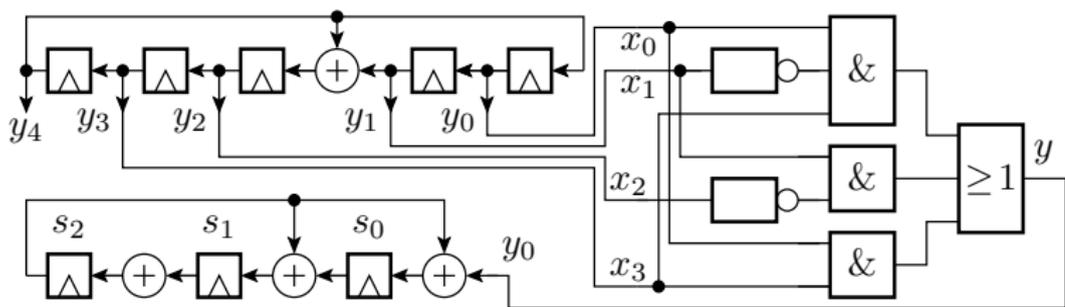
	$x_3x_2x_1x_0$	y	$s_2s_1s_0$
1	0 0 1 0 1		0 0 0
2			
3			
4			
5			
Signatur:			

- Schließen Sie am Eingang das 5-Bit LSFR von Folie 38 mit y_0 an x_0 , y_1 an x_1 bis y_3 an x_3 und an den Ausgang das 3-Bit-Signaturregister von Folie 40 an mit y an y_0 an.
- Bestimmen Sie die Sollsignatur für Generatorstartwert 00101, den Signaturregisterstartwert 000 und 5 Testschritte. Füllen Sie dazu rechts die Tabelle aus.

Zur Kontrolle

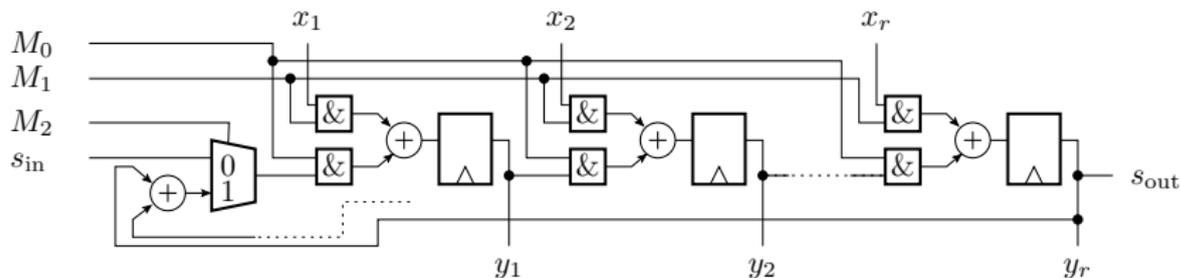
- 1 Selbsttestlösung mit Pseudo-Zufallsgenerator am Eingang und Signatoranalysator am Ausgang ↓
- 2 Wertetabelle Testablauf ⇒

	$x_3x_2x_1x_0$	y	$s_2s_1s_0$
1	0 0 1 0 1	0	0 0 0
2	0 1 0 1 0	1	0 0 0
3	1 0 1 0 0	0	0 0 1
4	0 1 1 0 1	1	0 1 0
5	1 1 0 1 0	1	1 0 1
Signatur:			0 0 1



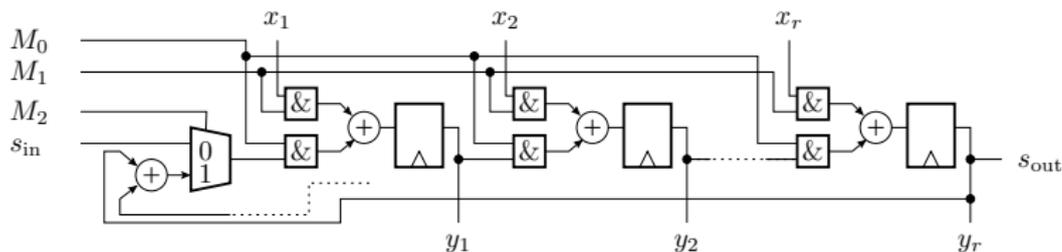
Aufgabe 5.20: Built-in Logic Block Observer

Die gezeigte Schaltung ist ein **Built-In Logic Block Observer** (BILBO) und führt in Abhängigkeit von den Steuersignalen M_0 bis M_2 eine der Funktionen aus: Intialisierung, normales Register, Schieberegister, Pseudo-Zufallsgenerator oder Signatureregister.



- 1 Welche Steuersignalbelegung steuert welche Funktion?
- 2 Vereinfachen Sie für jede dieser Steuersignalbelegungen die Schaltung durch Konstanteneliminierung (Vereinfachen bzw. Weglassen der logischen Verknüpfungen mit Konstanten.)

Zur Kontrolle



1 Steuersignalbelegungen und Funktion:

M_2	M_1	M_0	Funktion
-	0	0	Rücksetzen: $y_i = 0$
0	0	1	Schieben: $y_0 = s_{in}$, sonst $y_i = y_{i-1}$
-	1	0	Register: $y_i = x_i$
1	0	1	LFSR: $y_0 = y_r \oplus \dots$, sonst $y_i = y_{i-1}$
1	1	1	Signaturregister: $y_0 = x_0 \oplus y_r \oplus \dots$, sonst $y_i = x_i \oplus y_{i-1}$

2 Vereinfachte Schaltung für jede dieser Steuersignalbelegungen:

