CT Praktikum: Memory – SRAM Testing

1 Einleitung

Memory Tests sind wichtige Werkzeuge, um einerseits die Hardware von neu entwickelten Boards zu verifizieren und andererseits, um in der Produktion Fehler bei individuellen Leiterplatten zu detektieren (Produktionstest).

In diesem Praktikum entwickeln wir ein Programm, um die korrekte Funktion eines extern an einen Microcontroller angeschlossenen 2K x 8bit SRAMs zu prüfen, siehe Abbildung 1. Auf den ersten Blick erscheint dies ein einfaches Unterfangen zu sein. Allerdings müssen bei genauerem Hinsehen diverse Schwierigkeiten bewältigt werden, um einen kompakten, aussagekräftigen Test zu erhalten. Wir schauen uns zuerst die möglichen Fehlermechanismen in Hardware an und bauen dann schrittweise gezielte Tests auf, um solche Fehler zu detektieren und zu diagnostizieren.

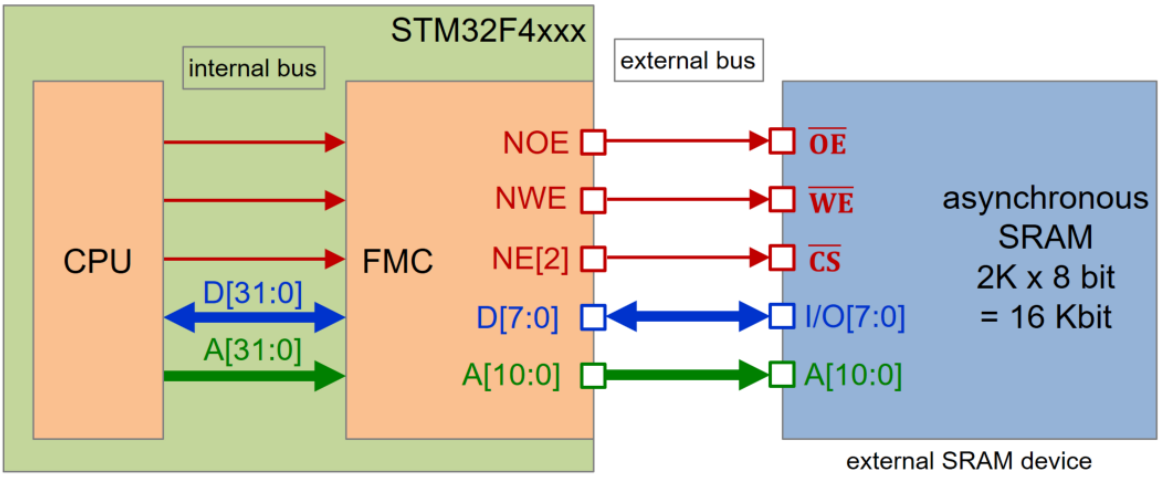

Abbildung 1: Anschluss SRAM an STM32 Microcontroller

2 Lernziele

- Sie kennen verschiedene Hardware Fehlermechanismen und verstehen ihre Auswirkungen auf die Software.
- Sie vertiefen und festigen Ihr Wissen im Bereich Anschluss von Speicherbausteinen.
- Sie können in Software kompakte und aussagekräftige Memory Tests für häufige Hardware Fehler entwickeln.

3 Aufbau

Abbildung 2 zeigt den Aufbau des Praktikums mit dem SRAM Zusatzboard.

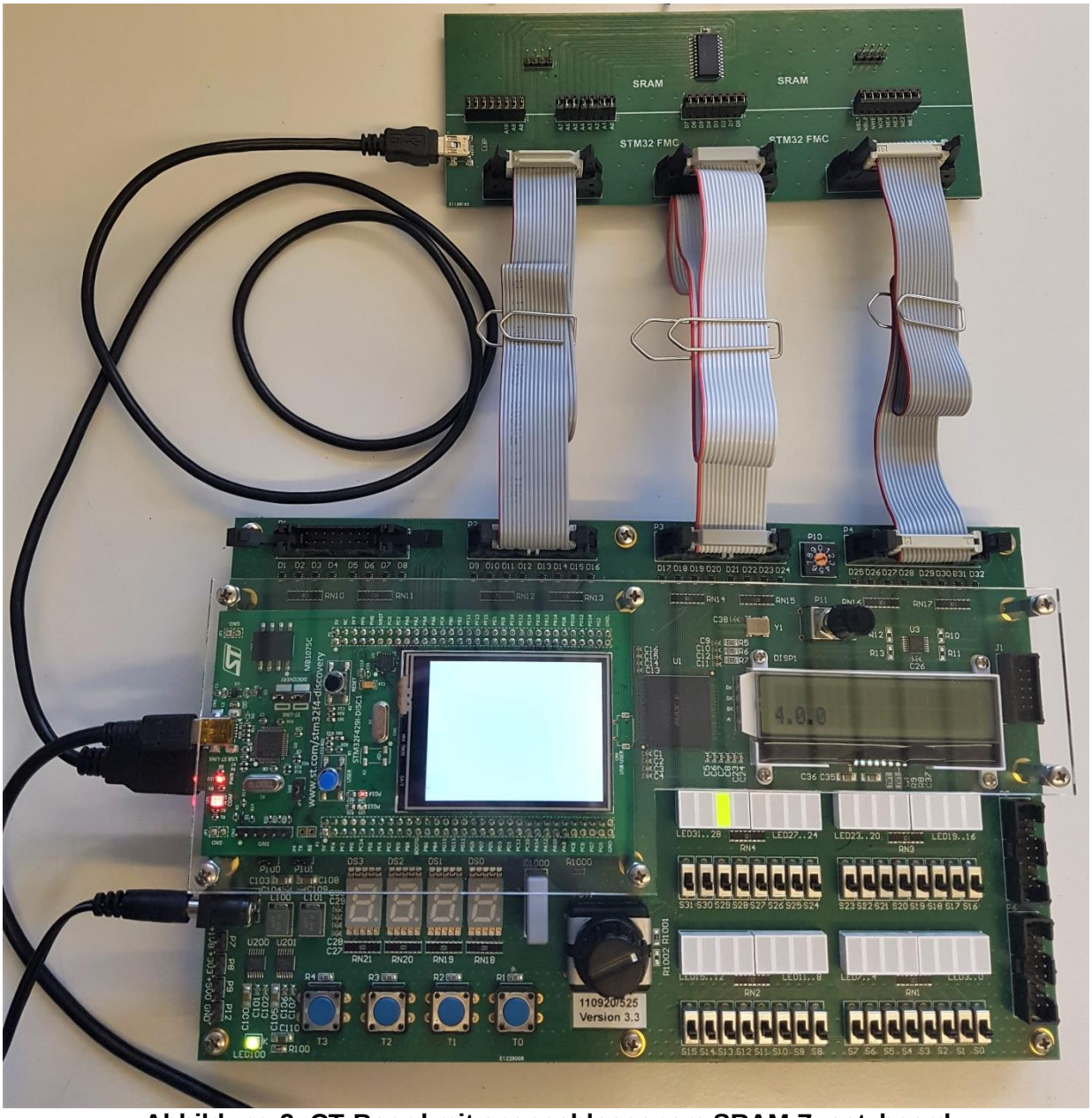

Abbildung 2: CT-Board mit angeschlossenem SRAM Zusatzboard

Für diese Funktion muss das CT Board in *Modus 2* betrieben werden!

In *Modus 2* schaltet das CT Board den externen Speicherbus auf die Schnittstellen P1 bis P4. Die genaue Belegung in diesem Modus ist im InES CT-Board Wiki beschrieben.

Für das Praktikum verwenden wir ein 2'048 x 8 bit grosses asynchrones SRAM, welches auf dem Zusatzboard an NE2 angeschlossen wird. Siehe Abbildung 3 und Abbildung 4.

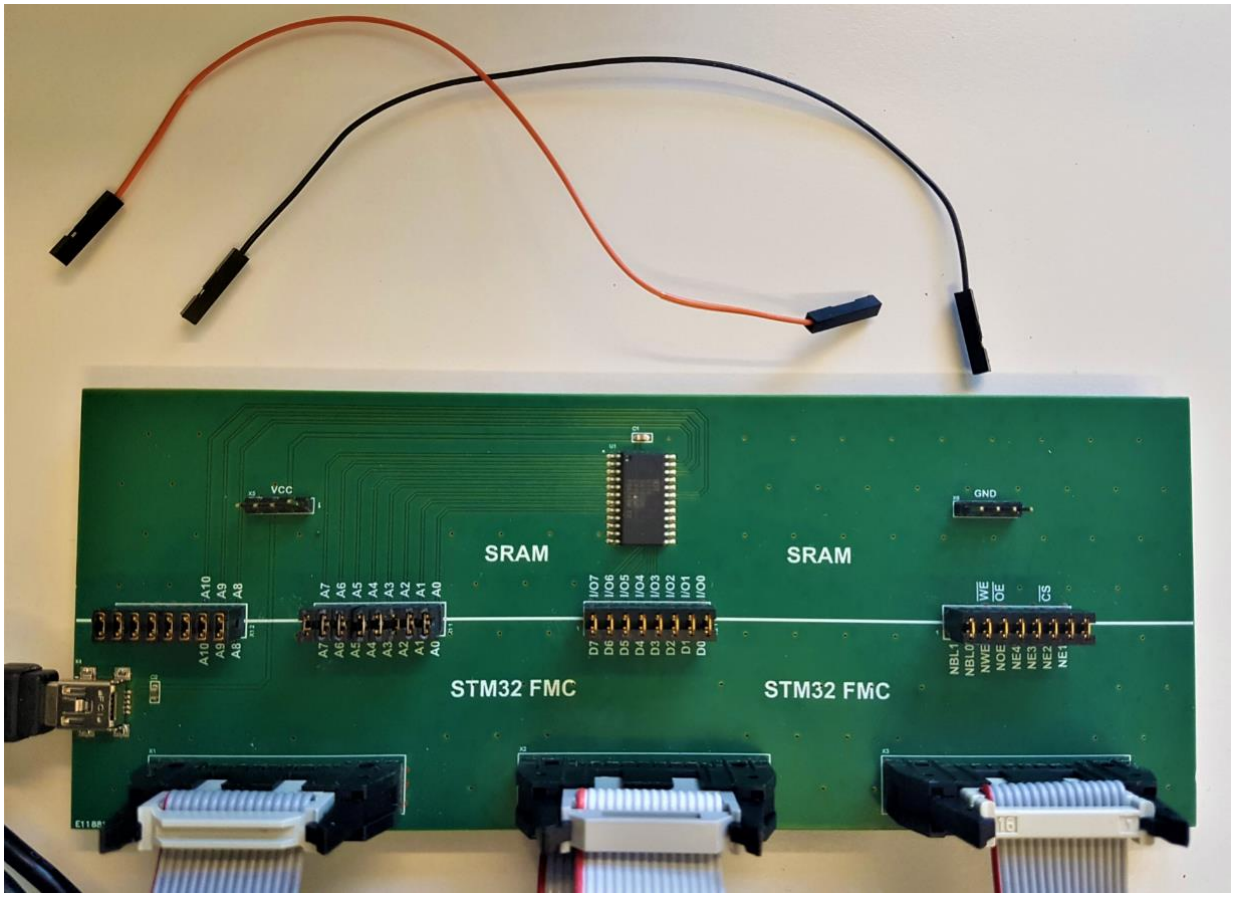

Abbildung 3: SRAM Zusatzboard mit Jumperkabel

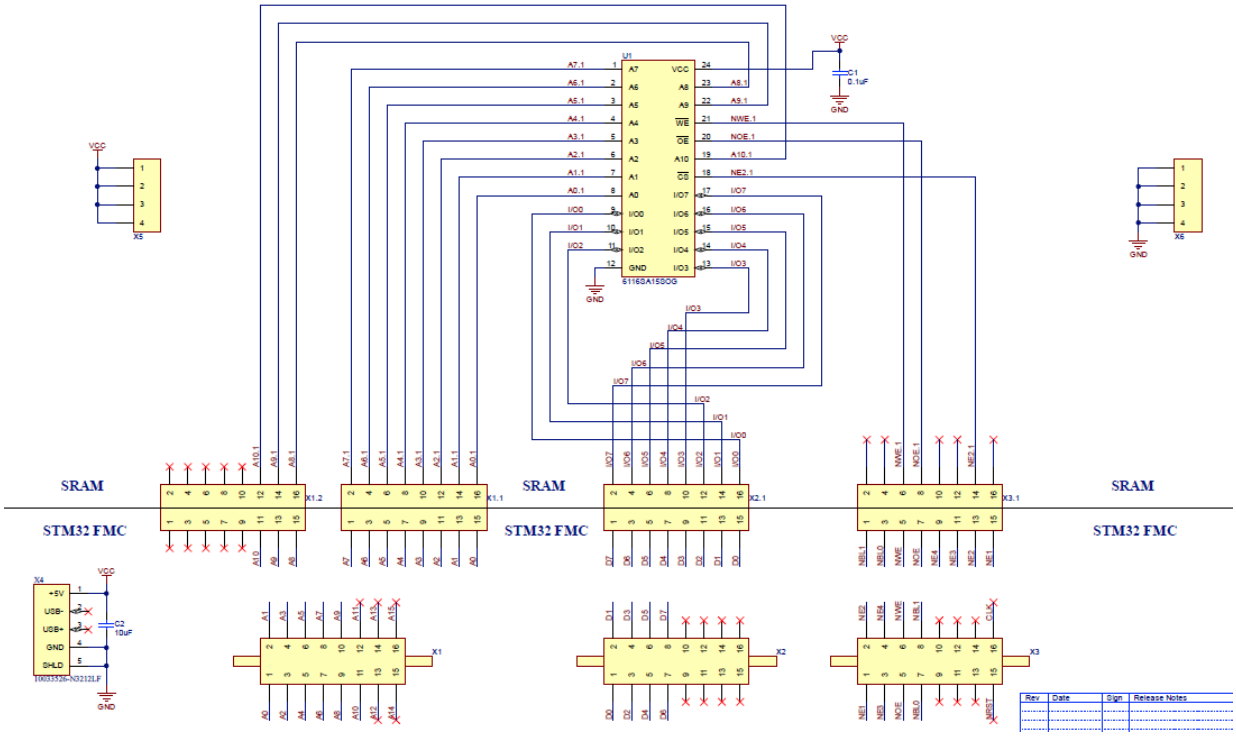

Abbildung 4: SRAM Zusatzboard Schema

4 Vorbereitung

Bestimmen Sie auf Grund des Schemas die Speicheradressen, unter denen das Memory in Software sichtbar ist. Tragen Sie die gesuchten Anfangs- und Endadressen in die Memory Map in Abbildung 5 ein. Grau schattiert ist der tiefste Adressbereich unter welchem das SRAM Memory angesprochen werden kann.

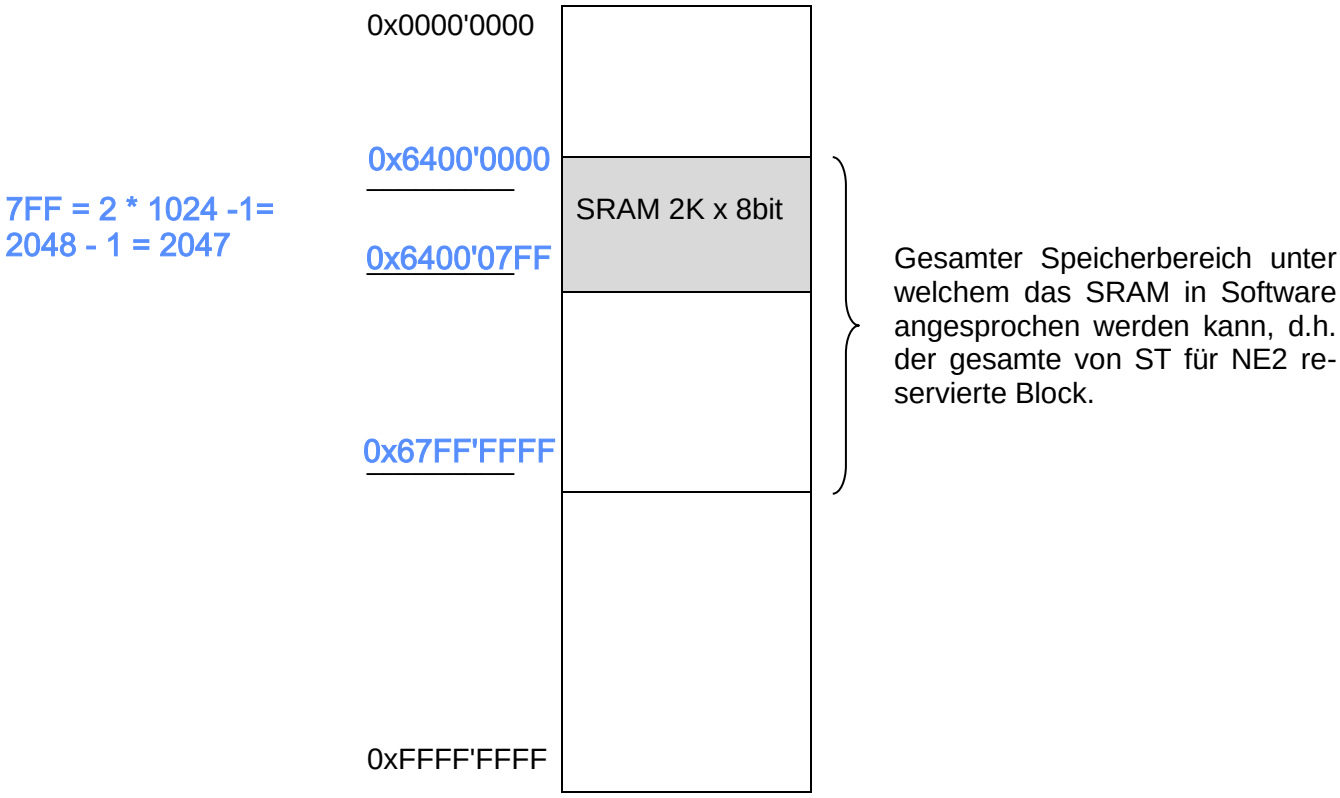

Abbildung 5: Memory Map mit Speicherbereichen

Auf Grund der unvollständigen Adressdecodierung ist das SRAM mehrmals im NE2 Block 'sichtbar', d.h. eine einzelne Adresse des SRAMs ist vom Microcontroller unter mehreren Adressen ansprechbar. Unter wie vielen einzelnen Unterbereichen innerhalb des NE2 Blocks kann das SRAM angesprochen werden?

---

4 Bit werden nicht codiert
2^4 = 16
Mit 16 Unterbereichen

Jede Adresse kann mit 16 verschiedenen Adressen aufgerufen werden

5 Grundlagen

Einsatzgebiete

In der Praxis werden Memory Tests in zwei Fällen eingesetzt: (A) Bei der Design Verifikation, um Fehler in der Entwicklung eines PCBs aufzudecken und (B) Im Production Test, um Fehler bei der Produktion der individuellen bestückten Leiterplatte zu entdecken. Je nachdem, für welchen der beiden Fälle der Test eingesetzt wird, erfolgt eine unterschiedliche Ausprägung des Tests und man verwendet leicht unterschiedliche Fehlermodelle. Beim Production Test hat die Laufzeit des Tests eine grosse wirtschaftliche Bedeutung.

Methode

Ein SRAM Memory Test verifiziert, dass jede Speicherstelle in einem Baustein einwandfrei funktioniert und angesprochen werden kann. D.h. wenn wir einen Wert an eine Adresse schreiben, dann soll dieser Wert unverändert unter der gleichen Adresse zurückgelesen werden können. Dies insbesondere auch dann, wenn zwischen Schreiben und Lesen der besagten Adresse Speicherzugriffe mit anderen Daten auf andere Adressen erfolgen. Ebenso sollen keine Speicherzellen an anderen Adressen überschrieben werden.

Im Folgenden entwickeln wir einen einfachen, sehr allgemeinen Test.

Fehlermechanismen

In diesem Praktikum konzentrieren wir uns auf Fehlermechanismen, welche auf der Leiterplatte, dem Printed Circuit Board (PCB) auftreten. Wir gehen hier davon aus, dass der Halbleiterteil des Speichers korrekt arbeitet.

Die nachfolgenden Fehlermechanismen können bei Adress-, Daten- und Kontrollleitungen auftreten:

- **Stuck-at Fault**
  Es besteht ein Kurzschluss zu Ground oder der Versorgungsspannung. Dadurch liegt eine Leitung fest auf ‘0’ oder ‘1’. Die Leitung kann nicht den anderen Wert annehmen.
- **Bridge Fault (shorted)**
  Es besteht ein Kurzschluss zwischen zwei nebeneinanderliegenden Leitungen, d.h. die Werte auf beiden Leitungen sind identisch. Beispiel: Bei einem Kurzschluss zwischen Adressleitungen A4 und A5 werden die durch den Microcontroller angesprochenen 8-bit Adressen 0x00, 0x10, 0x20 und 0x30 am Speicher alle entweder als 0x00 oder als 0x30 gesehen. Das Beispiel betrifft natürlich auch weitere Adressen.
- **Open**
  Ein Unterbruch in der Signalleitung. Ein vom Microcontroller ausgegebener Wert erreicht den Speicher nicht. Die betreffende Leitung liegt auf einem undefinierten Pegel.

6 Vorgehen

Wir implementieren unseren Test schrittweise in drei Teilen.

Wichtig: Bitte stecken Sie die Jumper und Kurzschlussdrähte nur bei **ausgeschaltetem** CT-Board um. Sie verhindern dadurch Beschädigungen des CT-Boards.

7 Aufgabe: Data Bus Test – ‘Walking Ones Test’

Dieser Test überprüft, dass sich jedes einzelne Bit des Datenbusses unabhängig auf ‘1’ bewegen lässt. Dazu wählen wir eine beliebige Adresse und führen alle Tests mit dieser durch. Im ersten Zugriff schreiben wir die tiefste Datenleitung D0 auf ‘1’ und alle anderen Datenleitungen auf ‘0’. Gleich anschliessend lesen wir den Wert der gleichen Adresse zurück und überprüfen ihn. Danach wiederholen wir den Vorgang für jede Datenleitung indem wir die ‘1’ zur höchsten Datenleitung durchschieben. Abbildung 6 zeigt die für unseren ‘Walking Ones Test’ auf dem Datenbus angelegten Bitpattern, sowie ein Beispiel für die Ausgabe von detektierten Fehlern.

Pattern	‘Walking Ones Test’ D[7:0]
p0	0000’0001
p1	0000’0010
p2	0000’0100
p3	0000’1000
p4	0001’0000
p5	0010’0000
p6	0100’0000
p7	1000’0000

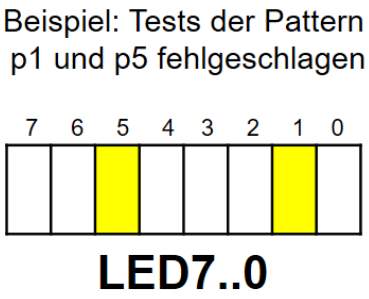

Abbildung 6: Bitmuster (pattern) für einen ‘Walking Ones Test’ auf einem 8-bit breiten Datenbus

a) Implementieren Sie den beschriebenen ‘Walking Ones Test’ in einer for-Schlaufe. Zeigen Sie fehlgeschlagene Tests gemäss Beispiel auf den LED7..0 an.

b) Verifizieren Sie Ihren Test. Führen sie ihn zuerst mit korrekt eingesetzten Jumpern durch. Danach entfernen Sie zielgerichtet einen oder mehrere Jumper auf dem **Datenbus** und erzeugen mit Kabeln Stuck-at ONE und Stuck-at ZERO Fehler. Siehe Abbildung 7.

Hinweis: Erzeugen Sie die Fehler auf der Seite des Microcontrollers (D7 – D0) und nicht auf der Seite des Memory’s (I/O7 – I/O0).

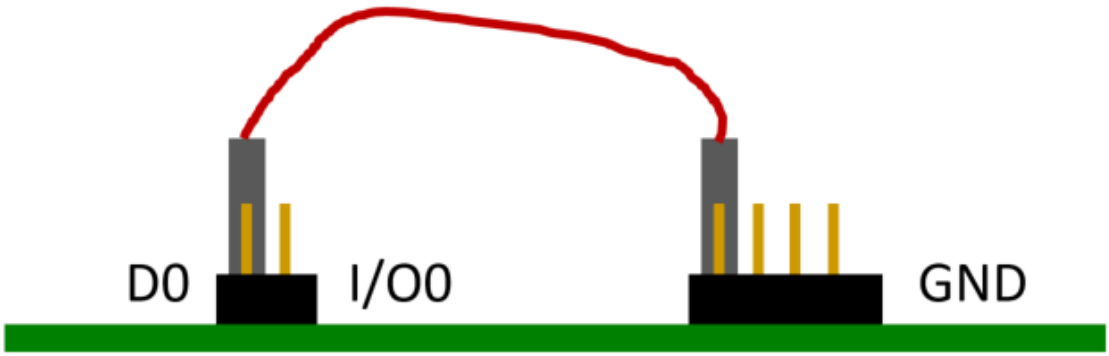

Abbildung 7: Emulation ‘Stuck-at ZERO’ Fault auf dem Datenbus Bit 0

c) Welche Patterns schlagen fehl, wenn Sie einen ‘Stuck-at ONE’ Fehler erzeugen?

Wenn D2 = 1
Dann leuchten alle LEDs ausser LED2

da zb. für p0 = 00000101 und eigentlich sollte es 00000001 sein.
nur p2 stimmt, da p2 = 00000100 gleich wie 00000100 also alles richtig

d) Welche Patterns schlagen fehl, wenn Sie einen ‘Stuck-at ZERO’ Fehler erzeugen?

Wenn D2=0
Dann leuchtet nur LED2
da p2 = 0000 0000 obwohl es eigentlich p2 = 00000100 wäre

e) Was passiert, wenn Sie eine einzelne Datenleitung offenlassen?

Wenn D2 nicht verbunden
alle LEDs ausser LED2 leuchten

Wenn D0 nicht verbunden
alle LEDs ausser LED0 leuchten

Wenn D2 und D0 nicht verbunden
alle LEDs leuchten
Wenn mehr als 1 D nicht verbunden leuchten alle,
wenn 1 D nicht verbunden leuchtet nur dieses nicht.

8 Aufgabe: Address Bus Test

Nachdem wir die Funktion des Data Buses überprüft haben, prüfen wir nun die einzelnen Adressleitungen. Wir wollen sicherstellen, dass jede einzelne Adressleitung sowohl auf ‘0’ als auch auf ‘1’ gesetzt werden kann, ohne andere Leitungen zu beeinflussen.

8.1 Grundidee – Ablauf

Ein Fehler auf einer Adressleitung führt in der Regel dazu, dass

(A) der geschriebene Wert nicht unter der gewünschten Adresse gespeichert, bzw. zurückgelesen wird und

(B) dass der geschriebene Wert unter einer anderen Adresse gespeichert und zurückgelesen wird.

Beispiel: Wenn der Microcontroller den Speicher an Adresse 0x008 adressieren möchte, aber das Bit A[3] in Folge eines Fehlers nicht auf ‘1’ setzen kann (stuck-at ZERO), so wird der auf dem Datenbus anliegende Wert nicht unter der Adresse 0x008 abgespeichert (Fall (A)) und stattdessen unter der Adresse 0x000 gespeichert (Fall (B)). Somit überschreibt der Zugriff den Wert, der eigentlich an Adresse 0x00 gespeichert ist. Siehe Abbildung 8.

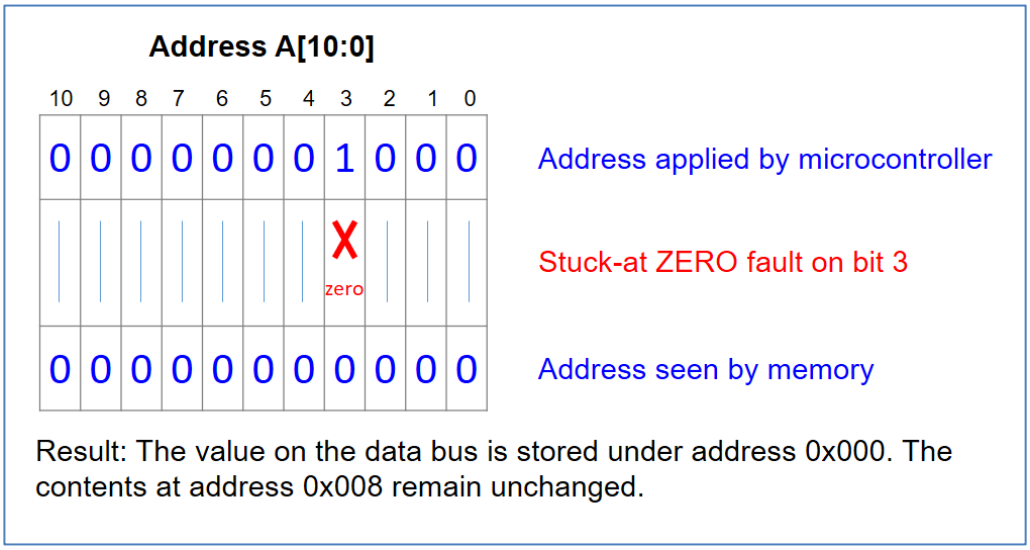

Abbildung 8: Beispiel eines Stuck-at ZERO Fehlers auf dem Address Bus

Damit wir nicht für jede mögliche Adresskombination einen Wert schreiben, zurücklesen und alle anderen Adresskombinationen überprüfen müssen, beschränken wir uns, analog zum «Walking Ones Test» auf diejenige Untermenge der Adressen, bei denen genau ein Adressbit gesetzt ist, plus die Adresse 0x0. Für einen 6-bit Adressbus A[5:0], sind dies beispielsweise die Adressen:

0x20	0010 0000b
0x10	0001 0000b
0x08	0000 1000b
0x04	0000 0100b
0x02	0000 0010b
0x01	0000 0001b
0x00	0000 0000b

Im Test überprüfen wir für jede unserer definierten Testadressen, ob das adressierte Byte im Speicher geschrieben und gelesen werden kann, und ob keine der anderen Speicherstellen beschrieben wurde.

Zunächst initialisieren wir die Speicherwerte aller Adressen aus unserer Untermenge auf einen definierten Wert. Wir wählen dazu den Wert 0xAA («Checker Board»). Danach wird für jede Adresse (*test_address*) die folgende Sequenz ausgeführt:

1. Ein anderer Wert, 0x55 («Inverse Checker Board») wird an *test_address* geschrieben.
2. Für alle Adressen wird überprüft, dass sie den richtigen Wert enthalten, also «Inverse Checkerboard» für *test_address* und «Checkerboard» für alle anderen Adressen.
3. Der Speicher an *test_address* wird wieder mit «Checkerboard» reinitialisiert.

8.2 Implementierung

Gehen Sie zur Implementierung schrittweise vor:

a) Implementieren Sie den Address Bus Test zuerst für eine fixe *test_address = 0x0400*.

Initialisieren Sie in einer Schleife alle betrachteten Speicherstellen (inklusive *test_address*) auf «Checkerboard».

Danach führen Sie die Schritte 1-3 durch. Dabei sollen detektierte Fehler gemäss Abbildung 9 auf den LED31..16 angezeigt werden.

Sie können diese Überprüfung in einer while Schleife wie folgt organisieren:

```
address = (uint16_t) 0x01 << NR_OF_ADDRESS_LINES;

while (address) {
    address >>= 1;
    // add your check of memory values here
}
```

Die erste Zeile stellt dabei die erste Adresse oberhalb des Memories dar.

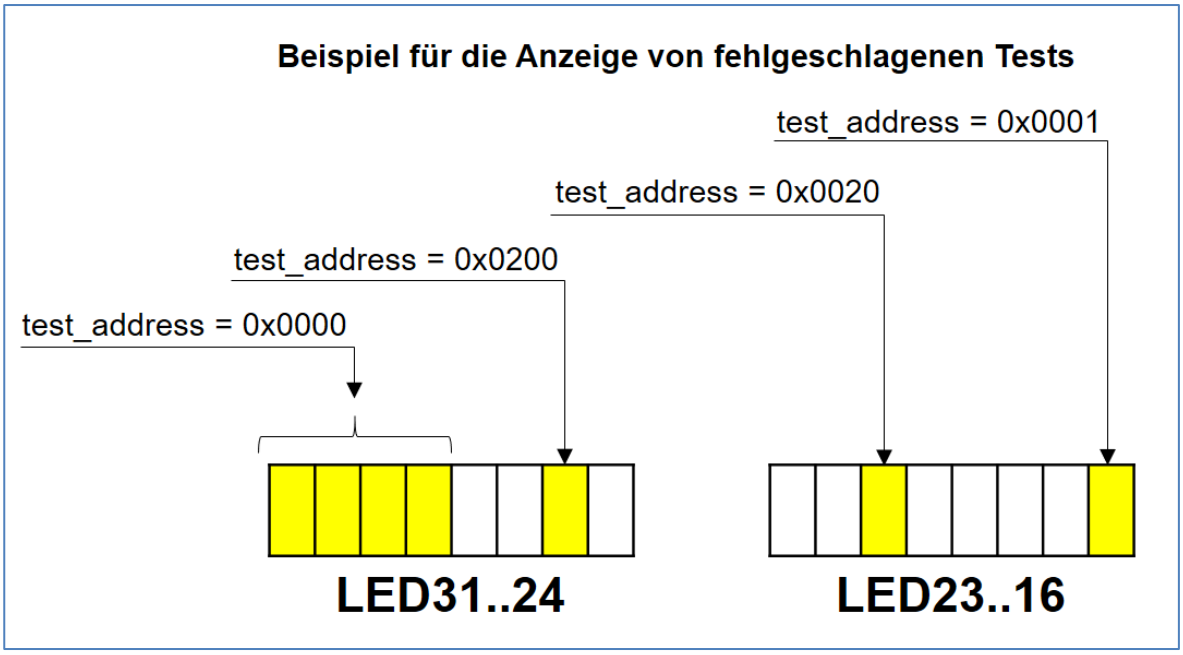

Abbildung 9: Anzeige von fehlgeschlagenen Address Bus Tests

10101010
01010101

A10 auf 1 oder 0: 100 0000 0000 eigentlich sollte es sein: 100 0000 0000. LED26 leuchtet bei beidem.

b) Verifizieren Sie Ihre Implementation indem Sie auf dem Address Bus des SRAM Zusatzboards Stuck-at ONE und Stuck-at ZERO auf A[10] erzeugen. Zeigt Ihr Test in beiden Fällen einen Fehler auf Adresse A[10], d.h. auf LED26 an? Die anderen LEDs sollten dunkel sein.

ACHTUNG: Da es sich beim Address Bus um Ausgänge des Microcontrollers handelt, müssen Stuck-at Faults auf der Seite des SRAMs und nicht auf der Seite des Microcontrollers erzeugt werden. Siehe Abbildung 10.

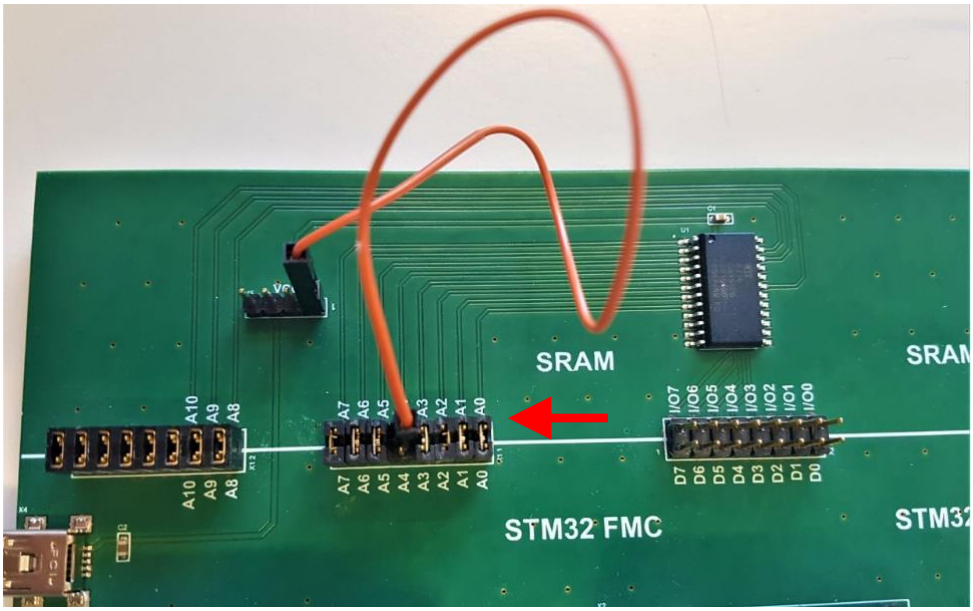

Abbildung 10: SRAM Zusatzboard mit Jumperkabel und Stuck-at-Address Fehler

c) Erzeugen Sie zuerst Stuck-at ONE und dann Stuck-at ZERO Fehler auf Adressleitung A[9]. Hier sollten durch Ihr Programm keine Fehler angezeigt werden. Begründen Sie an Hand von Beispielen für beide Fälle, wieso hier keine Fehler detektiert werden.

A9 auf 1: 010 0000 0000

Wir testen nur 0x400 = 100 0000 0000
Adresse, A[9] ist also irrelevant.

d) Bauen Sie Ihren Test mit einer Schleife so aus, dass alle Adressen aus unserer Untermenge überprüft werden. Beginnen Sie mit der höchsten *test_address* und iterieren Sie bis und mit 0x000.

e) Verifizieren Sie Ihre Implementation mit verschiedenen Fehlern auf dem Board und beschreiben Sie das Verhalten.

Jetzt wird alles richtig
angezeigt.
also wenn [A9] = 1
oder [A9] = 0 leuchtet
das led25

9 Aufgabe Device Test – ‘Increment Test’

Nachdem wir in den vorangegangenen Tests die Verbindungen des Data Bus und des Address Bus überprüft haben, überprüfen wir in diesem Schritt die Integrität des Speichers. Wir wollen sicherstellen, dass jedes Bit im Speicher sowohl eine ‘0’ als auch eine ‘1’ speichern kann. Dieser Test ist von der Laufzeit deutlich aufwändiger als die vorangehenden Tests.

Für den Test des kompletten Speichers schreiben und überprüfen wir jede Speicheradresse zweimal. Wir verwenden dazu ein Muster, welches sich mit der Speicheradresse verändert, aber nicht mit dieser identisch ist. Ein einfaches Beispiel dafür ist der in Abbildung 11 gezeigte Increment Test.

Im ersten Schritt füllen wir den gesamten Speicher mit dem gezeigten Increment Pattern. Im zweiten Schritt lesen wir jede Speicheradresse aus und überprüfen den Inhalt. Gleich nach dem Auslesen schreiben wir das bitweise invertierte Pattern in die entsprechende Speicherstelle. Im dritten Schritt lesen wir den Inhalt jeder Speicheradresse erneut aus und überprüfen, ob das Inverse Pattern gespeichert ist.

Address A[10:0]	Pattern Data D[7:0]	Inverse Pattern Data D[7:0]
0x000	0x01	0xFE
0x001	0x02	0xFD
0x002	0x03	0xFC
…	…	…
0x0FE	0xFF	0x00
0x0FF	0x00	0xFF
0x100	0x01	0xFE
0x101	0x02	0xFD
…	…	…
0x7FE	0xFF	0x00
0x7FF	0x00	0xFF

Abbildung 11: Pattern für Device Test – Increment Test

a) Implementieren Sie den Device Test. Geben Sie eine gefundene fehlerhafte Adresse auf der Siebensegmentanzeige aus und warten Sie auf einen Tastendruck auf Taste T0 um mit dem Test fortzufahren.

Ausgabe auf Siebensegmentanzeige und warten auf Tastendruck:

```
CT_SEG7->BIN.HWORD = test_address;
while (!hal_ct_button_is_pressed(HAL_CT_BUTTON_T0)) {
}
```

b) Verifizieren Sie Ihre Implementation mit verschiedenen Fehlern auf dem Board und beschreiben Sie das Verhalten.

10 Zusammenfassung: Anzeige von Fehlern

Abbildung 12 fasst die Anzeige der fehlgeschlagenen Tests zusammen.

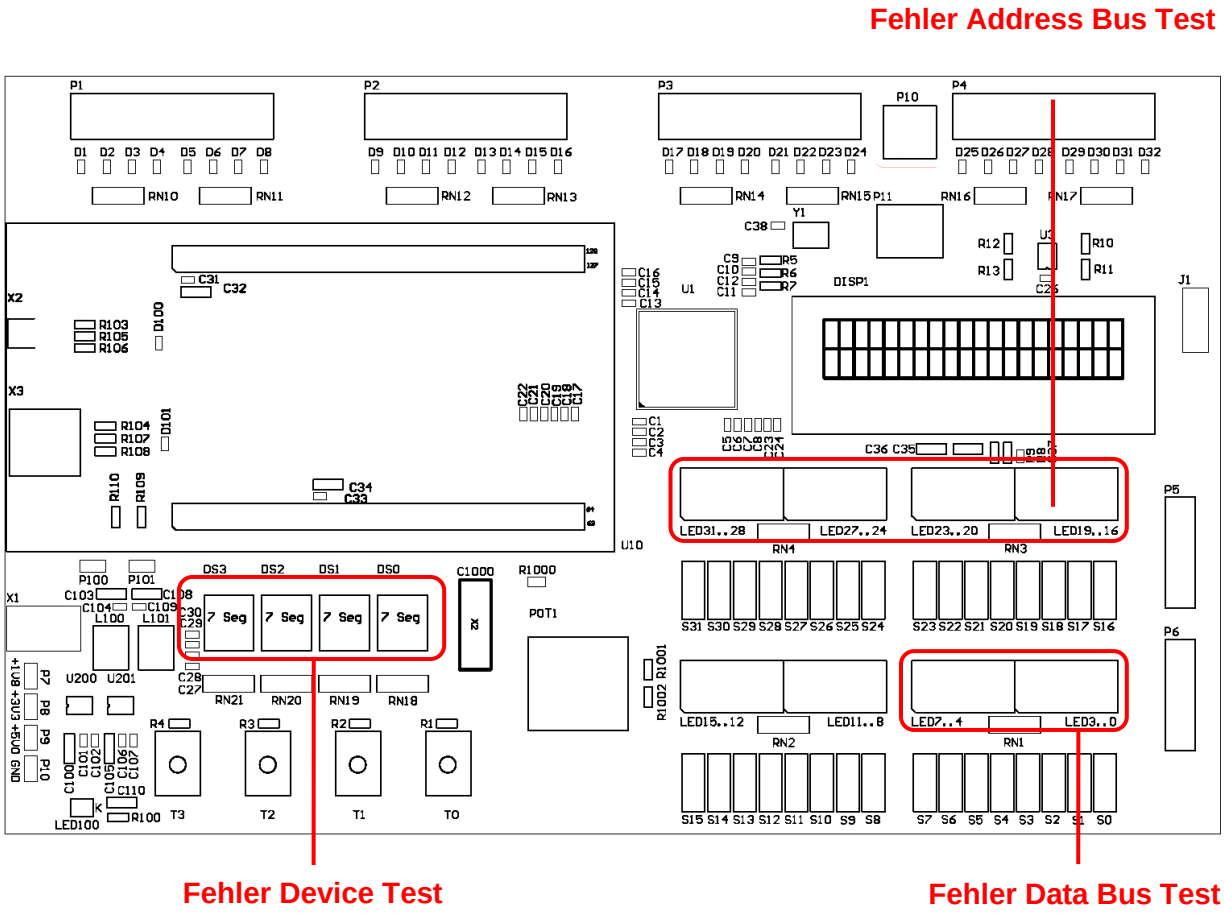

Abbildung 12: Anzeige von fehlgeschlagenen Tests auf CT-Board

11 Bewertung

Die lauffähigen Programme müssen präsentiert werden. Die einzelnen Studierenden müssen die Lösungen und den Quellcode verstanden haben und erklären können.

Bewertungskriterien	**Gewichtung**
Data Bus Test	2/4
Address Bus Test	2/4
Device Test – Zusatzpunkte	2/4