Fehlersuche innerhalb des µ-Controllers vom Typ PIC32MX
Fault tracing within the µ-Controller PIC32MX

Kurzfassung

Design und Technologie erreichen mehr und mehr ihre Grenzen, so dass eine simulationsbasierte Validierung nicht mehr genügt, um alle Möglichkeiten abzudecken [1]. Es ist allgemein bekannt, dass die Simulation alleine nicht ausreicht, um die Korrektheit komplexer Systeme zu gewährleisten [2]. Dadurch werden Bausteine aus der Industrie mit unbekannten Fehlern weiterverkauft. Die Fehler können von funktioneller oder struktureller Natur sein. Beim Kunden wird im Betrieb des Produktes nun ein Fehlverhalten beobachtet. Dieses Verhalten entsteht auf Grund eines Fehlers in einer Schaltung (oder eines nicht bekannten oder kommunizierten Features). Die Suche nach dem Fehler und dem Fehlerort bedingt Regeln, um ihn schnell zu finden und aufzulösen. Wichtig ist, dass man sich nach der Struktur orientiert. In dieser Arbeit geht es um die Lokalisierung eines Fehlers, der während des JTAG-Betriebs der µ-Controller vom Typ PIC32MX110, PIC32MX130 und PIC32MX210 auftritt. Für die Fehlersuche verwendet man den D-Algorithmus in einer neuartigen Implementierung.

Abstract

Design and technology are increasingly reaching their limits, so that simulation-based validation is no longer enough to cover all possibilities. It is well known that simulation alone is not enough to ensure the correctness of complex systems. As a result, building blocks from the industry are sold with unknown errors. The errors can be of a functional or structural nature. They are observed from the customer during the operation. This occurs because of a fault in a circuit (or an unknown or communicated feature). The search for the error and the fault location requires rules to quickly find and resolve it. It is important that you orient yourself according to the structure. This work deals with locating a fault that occurs during JTAG operation of the PIC32MX110, PIC32MX130 and PIC32MX210 µ-Controllers. For debugging, the D-algorithm is used in a novel implementation.

1 Motivation

Fehler sind Defekte, die auf Grund verschiedener Faktoren in einem System (oder mehreren gekoppelten Systemen) auftreten. Sie können bewirken, dass das System eine andere Funktion ausführt als die ursprünglich vorgesehene. In diesem Zusammenhang ist es wichtig Systeme auf Konsistenz überprüfen zu können, um dann mögliche auftretende Fehler beseitigen zu können. Mit zunehmender Anzahl an Eingängen bzw. Ausgängen ist das Durchmustern aller Möglichkeiten über die Eingänge unübersichtlich bis gar unmöglich [3]. Es kann damit gezeigt werden, dass das Problem, ob ein möglicher Fehler überhaupt detektiert werden kann, sogar NP-vollständig ist [4]. Dafür sind verschiedene Methoden und Algorithmen für die Fehlersuche entwickelt worden. Die Methoden streben danach, möglichst wenig Zeit und Aufwand einzusetzen. Sie arbeiten auf Grundlage folgender Prozedur: Zuerst werden die bekannten Fehler mit Hilfe von Fehlermodellen modelliert, dann werden sie mittels Methoden und Algorithmen gesucht [5]. Dabei wird das Grundprinzip der Steuerbarkeit und Beobachtbarkeit angewandt. Will man nun einen Fehler in einer Schaltung erkennen (beobachten), so muss man ihn stimulieren (steuern) und gleichzeitig beobachten können. Ein dafür geeigneter und bekannter Algorithmus für die Fehlersuche ist der sogenannte D-Algorithmus: Bei dieser Methode versucht man den Fehler am Ausgang beobachtbar zu machen indem man ihn durch passende Belegungen am Eingang (der Eingangssignale) nicht überschreibt. Diese Methode wird heutzutage angewandt. Zuerst wird also das Fehlerbild injiziert und dann der Algorithmus eingesetzt. Der Fehler soll über den kontrollierten Input beobachtbar am Ausgang gemacht werden [6]. In unserem Fall wurde dabei ein Fehlverhalten am Ausgang des µ-Controllers vom Typ PIC32MX der Firma Microchip Technology Inc. beobachtet. Dieses Fehlverhalten entsteht während des JTAG-Betriebs und sein Fehlerort ist nicht bekannt. In diesem Paper wird nun der D-Algorithmus verwendet aber mit anderer Vorgehensweise: man durchläuft vom Ausgang her rückwärts gerichtet die Schaltung und überlegt dabei wo und wie der Fehler auftreten könnte bzw. auftritt.

Paper Gliederung: Zuerst wird die theoretische Einführung und der Versuchsaufbau für den JTAG, in welchem das Fehlverhalten beobachtet wurde, erklärt. Dann wird das beobachtete Fehlverhalten beschrieben, und anschließend wird gezeigt wie der D-Algorithmus einzusetzen ist, um mögliche Fehler zu interpretieren. Schließlich wird zusammengefasst.

2 JTAG

2.1 Struktur und Betriebsart

JTAG ist eine bekannte Methode für den standardisierten Testzugang in eine Hardware. Sie dient dazu, integrierte Schaltungen mit ausschließlich 5 Pins (TCK, TMS, TDI, TDO und TRST) seriell zu testen und zu debuggen. Sie befindet sich heutzutage in fast jeder integrierten Schaltung und arbeitet im Normalbetrieb unabhängig von der Core-Logik. Dieses System besteht aus verschiedenen Komponenten wie etwa Register, Gatter und Multiplexer [7]. Bild 1 zeigt die Struktur des JTAG innerhalb einer integrierten Schaltung (IC):

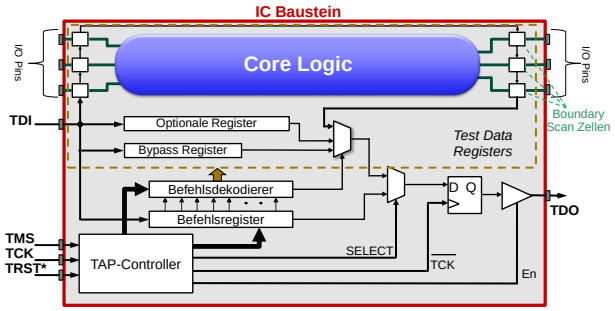

Bild 1: JTAG-Struktur in einem IC

Das Herzstück des JTAG ist der TAP-Controller. Er ist ein Automat mit 16 Zuständen und 12 Ausgängen, und steuert das gesamte System. Die Synchronisierung dieser Komponente erfolgt mit Hilfe des „Test-Clock“ (TCK). Durch den Eingang „Test-Mode-Select“ (TMS) wird entschieden, in welchen nächsten Zustand der TAP-Controller überführt werden kann. Die Zustandsüberführung erfolgt mit TCK steigender Flanke auf Grundlage des aktuellen Zustands. Bild 2 zeigt das Zustandsdiagramm (State Diagramm) eines JTAG-TAP-Controllers.
Im Zustandsdiagramm stellt der Zustand „Reset“ den initialen Zustand dar. In diesen kann der JTAG immer augenblicklich zurückkehren. Dieser Zustand wird erreicht entweder durch den optionalen „Test-Reset“ (Eingang TRST), wenn er zur Verfügung steht, oder mit 5-maliger Ausführung hintereinander von TCK steigender Flanke und TMS = 1, egal in welchem aktuellen Zustand man sich befindet. Das Zustandsdiagramm ist in zwei Spalten angeordnet: Die erste Spalte beschreibt das Arbeiten im Datenregister (DR), die zweite Spalte das Arbeiten im Instruktionsregister (IR). Die JTAG-Schaltung selber hat ein Instruktionsregister (Befehlsregister) und weitere verschiedene Datenregister. Das Instruktionsregister wählt über seinen Instruktionsdekodierer zwischen den Datenregistern. Das Schieben der Information in die Register erfolgt über den „Test-Data-Input“ (TDI) in genau und nur den Zuständen „ShiftIR“ und „ShiftDR“. Genauso ist der Ausgang TDO („Test-Data-Output“) nur in diesen Zuständen aktiv, sonst ist er auf High-Z gestellt. Die Aufnahme der Signale an den Eingängen TDI und TMS erfolgt bei TCK steigender Flanke, die Signaländerung am TDO erfolgt bei TCK fallender Flanke. Die Auswahl der Verbindung zwischen Register und TDO wird von Multiplexern über TAP-Controller-Ausgangssignale gesteuert.

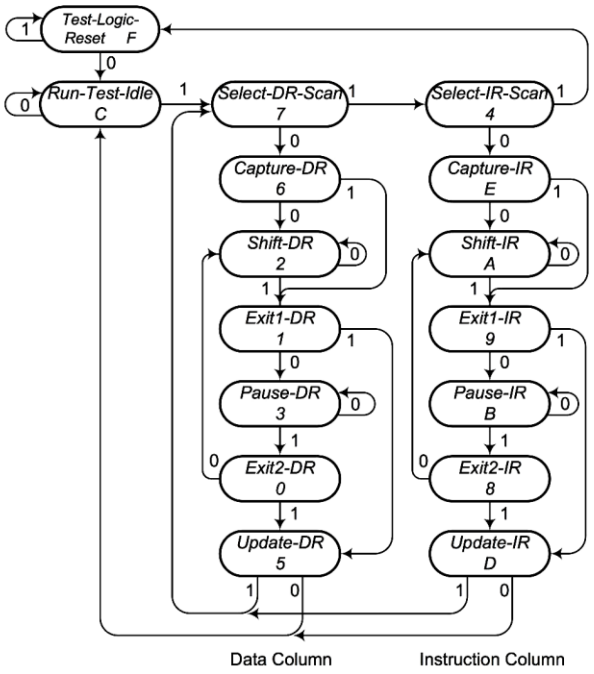

Bild 2: TAP-Controller State Diagramm

Die Standard-Datenregister sind das Bypass- und das Boundary-Scan-Register. Das Bypass-Register ist ein 1-bit Register, das Boundary-Scan-Register besteht aus einer Kette von Registerzellen (BSC), deren Anzahl (ihre Länge) von der Anzahl der Ein- und Ausgangspins im integrierten Baustein abhängig ist. Sie befinden sich zwischen den äußeren Pins des Bausteins und der Core-Logik und haben die Aufgabe Daten aus oder in die Core-Logik zu speichern oder I/O-Pins zu steuern. Der Speicher- und Schiebevorgang der Daten in den BSC geschieht während der Instruktion „Sample/Preload“ im Normalbetrieb der integrierten Schaltung (ohne Entkoppeln der Core-Logik), während die Instruktion „Extest“ das Gesamtsystem in den Testbetrieb schaltet (Core-Logik wird von den äußeren Pins entkoppelt) und die IO-Pins über BSC gesteuert [7]. Um die Funktionalität des JTAG auf Hardwareebene Schritt für Schritt nachzuvollziehen, wurde ein Demonstrator aufgebaut. Der JTAG kann damit im Handbetrieb (Schaltbetrieb, DC Betrieb) über Instruktionen und Daten gesteuert werden. Der Modulaufbau wird im nächsten Abschnitt erklärt.

2.2 Aufbau

Das Modul für den JTAG-Handbetrieb (Schaltbetrieb, DC Betrieb) ist aus zwei JTAG-Trainerboards (JTB) zusammengesetzt. Das Modul dient dazu, JTAG-Funktionalität

mit den µ-Controllern aus der Baureihen PIC32MX1XX bzw. PIC32MX2XX [8] zu steuern und zu demonstrieren. Es sind 2 Varianten von den Boards vorhanden, eine mit einem Sockel auf dem der Baustein aufgebracht wird und eine auf die der Baustein direkt gelötet wurde. Bild 3 zeigt wie das Modul verschalten ist:

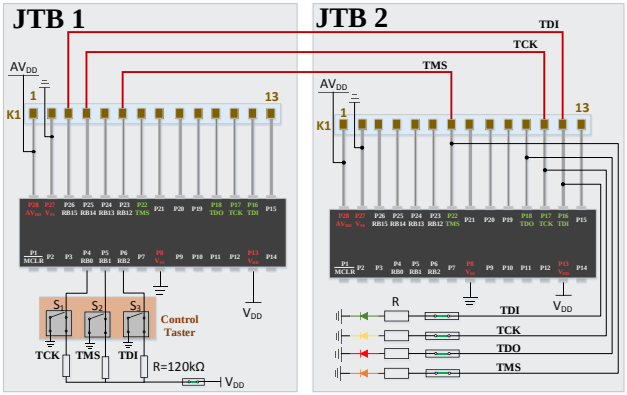

Bild 3: Schaltungsaufbau für den JTAG-Betrieb

Das Modul ist mit folgenden Bauteilen ausgestattet:

- Zwei µ-Controller PIC32MX110: Der erste dient zu Signalgenerierung für JTAG-Inputs und der zweite zum JTAG-Betrieb.
- 3 Taster: Dienen zur handbetrieblichen Steuerung der JTAG-Signale nämlich TCK, TMS und TDI.
- USB-Schnittstelle: Dient zur Spannungsversorgung mit einem Spannungsregler.
- Buchsenleisten: Sind über Leiterbahnen mit den Pins der µ-Controller verbunden: Dienen zur externen Spannungsversorgung und zur Verfügungsstellung der µ-Controller Pins.
- LED Pins: Dienen zur visuellen Kontrolle der JTAG-Ein- und Ausgangssignale.

Das Steuern der Signale erfolgt mit einem Knopfdruck auf einen Taster. Ein einmaliger Druck wechselt den Zustand des Signals (0→1 oder 1→0). Das Signal wird zuerst vom Eingangspin empfangen, mittels Firmware entprellt und über weitere Ausgangspins ausgegeben. Die Ausgangspins des µ-Controllers auf dem JTB1 sind mit den JTAG-Eingangspins des µ-Controllers auf dem JTB2 über Jumperkabeln miteinander verbunden. Zur visuellen Kontrolle der JTAG-Eingänge und der JTAG-Ausgänge sind vier LED auf dem Board mit diesen Pins verschalten. Über diese werden die logischen Werte der Signale gezeigt. Die Boards können entweder über USB bzw. externe Quelle versorgt werden. JTAG verfügt über einen eigenen Clock (TCK) und wird dadurch unabhängig vom Systemclock betrieben. Das Instruktionsschieberegister innerhalb unseres Bauteils besteht aus einem 5-bit Register und führt dazu, dass jede Instruktion im JTAG mit einer 5-bit Wortlänge kodiert ist. Die wichtigen Instruktionen für unseren Aufbau sind: „Bypass“, „IDCODE“, „Sample/Preload“ und „Extest“. Das Bypass Register ist ein 1-bit Register, der IDCODE ein 32-bit Register und die Boundary-Scan-Kette ein 52-bit Register.

3 Fehlersuche

3.1 Beobachtetes Fehlverhalten

Das Laden der Instruktionen im JTAG erfolgt im Zustand „UpdateDR“ des TAP-Controllers, nachdem der Instruktionscode in den Zustand „ShiftIR“ bitweise über TDI eingeschoben wurde. Die Datenwertausgabe am TDO erfolgt im Zustand „ShiftDR“. Das Auslesen der Datenwerte am TDO bei den Instruktionen „Bypass“ und „IDCODE“ hat richtige Ergebnisse geliefert. Man konnte eingeschobene Werte am Ausgang sehen. Bei den Instruktionen „Sample/Preload“ oder „Extest“ gibt der TDO keinen Wert aus. Das hat dazu geführt, den TDO am Pin mit dem Oszilloskop beobachten und messen zu wollen. Die Messung hat gezeigt, dass der TDO im Zustand „ShiftDR“ des TAP-Controllers inaktiv bleibt (High-Z). Er wird wieder aktiv, sobald ein JTAG-Reset stattfindet oder eine andere Instruktion, die das Fehlverhalten nicht zeigt, geladen wird. Um die Reproduzierbarkeit des Zustandes zu bestätigen, wurden verschiedene Bausteine aus der Reihe der PIC32MX getestet. Es sind folgender Bauteile getestet worden: PIC32MX110, PIC32MX130 und PIC32MX210. Die Testreihe umfasste neugekaufte, umprogrammierte und programmierte Chips. Das Fehlerverhalten lässt sich reproduzieren. Um zu erkennen, ob der Fehler bei anderen µ-Controllern mit anderen Architekturen auftritt, wurden Chips aus der Reihen dsPIC33 getestet. Der Fehler trat nicht auf und die Instruktionen wurden richtig am TDO-Ausgang ausgegeben.

3.1.1 Testen mittels ATE

Bevor man den JTAG-Aufbau realisiert hat, wurden die µ-Controller im Labor mittels eines automatischen Testsystems (ATE) überprüft. Das ATE stimuliert die JTAG-Eingänge über generierte Pattern und misst die Ausgangswerte des Testobjektes. Die Erwartungswerte wurden mit den gemessenen Werten über Komparatoren verglichen und ein PASS/FAIL-Entscheidung getroffen. Die Signale wurden mit einer Frequenz von 8MHz stimuliert. Die Ergebnisse haben ein PASS gezeigt und der TDO war im Zustand „ShiftDR“ aktiv. Wir nehmen an, dass unser Aufbau stabil ist, und setzen das Fehlermodell der Zeitabhängigkeit ein. Um dieses Fehlermodell zu testen, stimulieren wir den JTAG mit verschiedenen Frequenzen und beobachten den TDO. Das Resultat hat gezeigt, dass der JTAG ab einer Frequenz größer 500Hz das Fehlerverhalten nicht mehr zeigt. Unter dieser genannten Frequenz ist ein Fehlverhalten beobachtbar. Es tritt auf, sobald die Frequenz verlangsamt wird. Das heißt, wenn man beispielsweise im Zustand „ShiftDR“ in „Sample/Preload“

oder „Extest“ während des Werteschiebens die Frequenz unter 500 Hz verlangsamt, geht der TDO sofort in den High-Z-Zustand.

3.1.2 Logischer Zustand der IO-Pins

Die Instruktionen „Sample/Preload“ und „Extest“ verbindie TDI- und TDO-Pins mit der Boundary-Scan-Zellen (BSC), d.h. der Fehlzustand tritt auf, wenn eine Verbindung mit den BSC stattfindet. Daher soll geprüft werden, ob das Fehlverhalten auch in I/O-Pins auftritt. Die Instruktion „Sample/Preload“ dient dazu, im normalen Betrieb des Systems logische Werte an den Pins im BSC zu speichern oder Daten aus den BSC zu schieben, ohne seine Funktionen zu beeinträchtigen. Die Messung unseres Aufbaus hat ergeben, dass alle Pins des Bauteils sofort auf den High-Z-Zustand wechseln, sobald die Instruktion geladen wird (Zustand „UpdateDR“ des TAP-Controllers). Die Pins bleiben auf High-Z bis zu einem JTAG-„Reset“ oder bis eine andere Instruktion geladen wird.

3.2 Lokalisierung der Fehler

3.2.1 D-Algorithmus

Bei unserem Aufbau wird nach einem Fehler gesucht, der selbst sowie der Ort seines Auftretens unbekannt ist. Beim naiven Durchsuchen und Ausprobieren gerät man in die NP Vollständigkeit. Das ist aber nicht zielführend. Das geeignetere Vorgehen dazu ist, den D-Algorithmus einzusetzen. Es müssen für seinen Einsatz die gerichtete Realität und Strukturtreue der Schaltung berücksichtigt werden. Das Ergebnis des D-Algorithmus sind nun die erzeugten Testmuster, die für die Steuerbarkeit und Beobachtbarkeit des Fehlers zuständig sind. Unser veränderter D-Algorithmus geht aber so vor, dass vom Ausgang her rückwärts mit Entwurfswissen die Schaltung durchlaufen wird und dabei detektiert wird wo ein Fehler auftreten „könnte“. Dafür wird dann ad-hoc ein Fehlermodell angenommen. Damit werden die Testpattern für die Lokalisierung des Fehlers erzeugt. Man vermutet also genau dort den Fehlerort und erzeugt so die Pattern über Inputs. Anschließend überzeugt man sich, ob dieser Fehlerort auch tatsächlich detektiert werden kann. Ist der Fehler einstellbar, muss man überlegen welches Signal ihn einstellt. Also, sobald eine Abzweigung detektiert wird, muss entschieden werden, welcher Pfad den Fehler propagiert bzw. das Fehlverhalten verursachen. Sind mehrere Pfade mögliche, müssen diese nacheinander abgearbeitet werden. Der Abzweigungspunkt wird dazu markiert. Man kehrt nun immer wieder zu ihm zurück, und führt die restlichen Möglichkeiten aus. Der Algorithmus zeigt auf, welcher Fehler (oder welches nicht berichtete Verhalten) steuerbar und beobachtbar ist.

3.2.2 Prinzipschaltung eines I/O-Pin

Für den Zweck der Fehlersuche werden Schaltungen dem Referenz-Manual entnommen und nach möglichen Fehlerorten durchsucht. Bei der Schaltungsanalyse wurde auch festgestellt, dass einige Fehler im Manual stehen. Die Fehler wurden bestätigt und die Schaltung von uns neu gezeichnet. Das Bild 4 zeigt die aus dem Manual selbstständig geänderte Struktur der IO-Pins [9]:

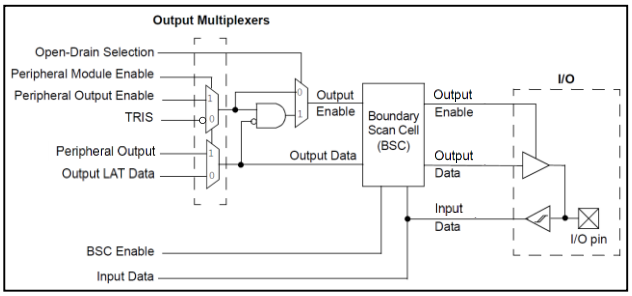

Bild 4: Struktur eines I/O-Pin (geändert für PIC32MX)

Die Struktur des µ-Controllers ist auf Grundlage von CMOS Schaltungen realisiert. Die Schaltung richtet sich demnach von der Core-Logik zu den Ausgangspins aus. D.h. die Core-Logik befindet sich hinten und die Ausgangspins vorne. Aus der obigen Schaltung sieht man auch, dass der I/O-Pin als Eingang oder Ausgang konfiguriert werden kann. Das Ausgangssignal „Output Data“ verläuft durch einen Ausgangstreiber, der über das Ausgangssteuersignal „Output Enable“ ein- und ausgeschalten wird (aktiv oder High-Z). Der Treiber wird mit „Output Enable“ = 1 aktiv. Das Eingangssignal von aussen nach innen (von vorne nach hinten) verläuft über einen Schmitt-Trigger. Für die Absicherung des Eingangs sind programmierbare Pull-Up- und Pull-Down-Schaltungen vorhanden. Hinter den Treibern ist die BSC mit den Leitungen verbunden. Die Verschaltung des Ausgangssignals wird über Multiplexer gesteuert. Der erste Multiplexer hinter der BSC entscheidet, ob der Ausgang als Open-Drain oder als Normal geschalten wird. Die Core-Logik wird über das Signal aus dem „Open-Drain“ Register gesteuert. Mit „Open-Drain“ = 1 wird der Ausgang als Open-Drain Ausgang verschaltet. Die hinteren zwei übereinanderliegenden Multiplexeren entscheiden, ob der Ausgang über die Peripherie oder über die Core-Logik gesteuert wird. Die Peripherie kann ebenso den Ausgangstreiber über das „Peripheral Output Enable“ Signal ein- und ausschalten. Für die Steuerung des Ausgangs über die Core-Logik stehen die Register „TRIS“ für Tristate und „LAT“ für Latch. Mit „TRIS“ = 0 wird der Ausgangstreiber eingeschalten und „LAT“ liefert den logischen Wert 0 oder 1 am Ausgangspin.

3.2.3 Ausschließen des Fehlers außerhalb der „Sample/Preload“ Instruktion

Die Beschreibung der I/O-Pins ist der Ausgangspunkt, um Fehler zu lokalisieren. Wir nehmen an, dass der Fehler nur in Boundary-Scan-Instruktionen auftritt. Das wird bestätigt, indem man die Beschreibung aus dem Manual prüft. Danach vergleichen wir die Ergebnisse der Spezifikation mit denen des Fehlverhaltens in „Sample/Preload“. Die Messungen und Verschaltungen der I/O-Signale ergeben die Tabelle 1:

PME	OD	TRIS	LAT	POE	PO	INPU	INPD	PIN Messwert	
								$\overline{S/P}$	S/P
0	0	0	0	-	-	-	-	0	Z
0	0	0	1	-	-	-	-	1	Z
0	0	1	-	-	-	0	0	Z	Z
0	0	1	-	-	-	0	1	0	Z
0	0	1	-	-	-	1	0	1	Z
0	1	0	0	-	-	-	-	0	Z
0	1	0	1	-	-	-	-	Z	Z
0	1	1	-	-	-	-	-	Z	Z
1	0	-	-	0	-	-	-	Z	Z
1	0	-	-	1	0	-	-	0	Z
1	0	-	-	1	1	-	-	1	Z
1	1	-	-	0	-	-	-	Z	Z
1	1	-	-	1	0	-	-	0	Z
1	1	-	-	1	1	-	-	Z	Z

Tabelle 1: Messergebnisse innerhallb und außerhalb „Sample/Preload“ Instruktion: Peripheral-Module-Enabe (PME), Open-Drain (OD), Tristate (TRIS), Latch (LAT), Peripheral-Output-Enable (POE), Peripheral-Output (PO), Input-mit-Pull-Up (IN PU), Input-mit-Pull-Down (IN PD), nicht-Sample/Preload ($\overline{S/P}$), Sample/Preload (S/P).

Die Messergebnisse zeigen, dass der Ausgangswert ständig auf High-Z gesetzt wird, sobald die „Sample/Preload“ Instruktion geladen wird. Im nächsten Schritt wird nun der D-Algorithmus angewandt.

3.2.4 Anwendung des D-Algorithmus

Das erste auftretende Fehlermodell wäre: Der Ausgangstreiber wird ausgeschalten und lässt kein Signal mehr durch. Für die Anwendung des D-Algorithmus soll der Treiber nun einstellbar sein. Die Einstellung kann hier mit einer BSC in „Extest“ stattfinden. Wir müssen uns also die Struktur der BSC auf Fehlermöglichkeiten untersuchen. Die Struktur der BSC wird im Bild 5 gezeigt [10].
Das Schaltbild musste ebenfalls an einigen Stellen korrigiert werden. Die BSC eines I/O-Pin besteht aus drei Zellen, eine für den Eingang (I), eine für den Ausgang (O) und eine für das Control-Signal (C). Eine Zelle im JTAG besteht zusätzlich aus zwei Multiplexer und zwei Bitregister. Zum besseren Verständnis wird nun die BSC von

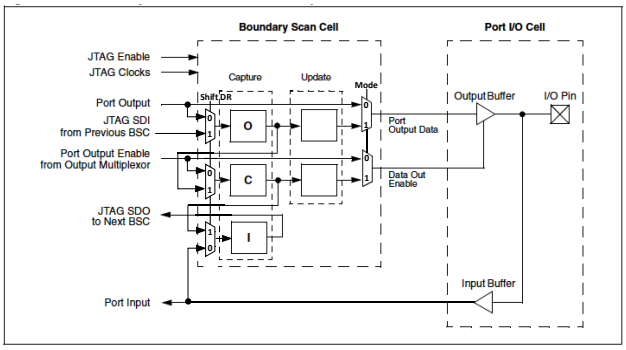

Bild 5: Struktur einer BSC im PIC32MX Manual

hinten nach vorne beschrieben. Der mit dem „Shift DR“ Signal gesteuerter Multiplexer entscheidet zwischen einem Signal aus der Core-Logik oder einem Signal aus der seriellen Verbindung des JTAG. Die Capture-Zelle speichert das aus dem Multiplexer belegte Signal. Die Update-Zelle speichert das Ausgangssignal der Capture-Zelle. Der Input-Port des I/O-Pin hat keine Update-Zelle und kann nur Signale aus dem Input-Buffer speichern. Die Aufnahme in die Capture-Zelle erfolgt entweder im Zustand „CaptureDR“ aus der Core-Logik oder im Zustand „ShiftDR“ über serielles Schieben am TDI.
Die Übertragung der in der Capture-Zelle gespeicherten Daten in die nächste „Update“-Zelle geschieht im Zustand „UpdateDR“. Der mit Mode Steuersignal Multiplexer entscheidet über dieses Signal mit 1 und 0 Werte entsprechend, ob der Chip im Testbetrieb oder im Normalbetrieb geschalten ist. Wenn die Instruktion „Extest“ ausgewählt ist, ist der µChip im Testbetrieb. Jetzt wird die Annahme getestet. Daten werden über seriell über TDI geschoben. Der Ausgang wird nun über Signalsteuerung in „Extest“ getestet und beobachtet. Die Möglichkeit der Treibersteuerung über serielles Schieben von 1 und 0 Werte in die BSC zeigt, ob die Multiplexer und die BSC richtig schalten, um den Fehlerort zu bestimmen. Ein Testbeispiel wäre eine 1 in der Update Zelle vom Control-Signal speichern und die Update Zelle vom Ausgang wechselnd mit 1 und 0 belegen mit simultaner Beobachtung des Ausgangspins. Man stellte durch die Messergebnisse am Ausgang fest, dass das Steuersignal und das Ausgang vom Treiber steuerbar sind und keinen Fehler zeigen.
Das zweite auftretende Fehlermodell wäre: Das Steuersignal des Ausgangstreibers wird von der Core-Logik mit einer Null beliefert. Das kann überprüft werden, indem das Steuersignal in BSC gespeichert, nach der Ausgangszelle über TDI geschoben und am Ausgang beobachtet wird. Darüber hinaus kann man testen, ob der Capture-Vorgang sowie der Input-Buffer schaltet. Man braucht dafür die Zuordnung der Boundary-Scan-Kette um die Pins richtig zu steuern. Die Daten werden vom Manual geholt. Das Bild 6 zeigt eine allgemeine Darstellung der Boundary-Scan-Kette unserer µ-Controller.

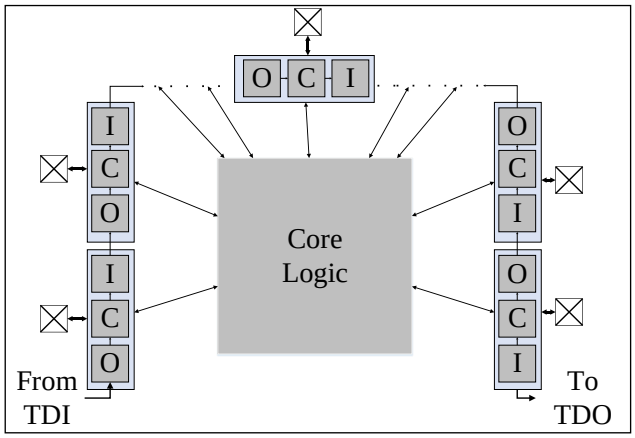

Bild 6: Zuordnung der BS Kette im PIC32MX

Um den Wert des Steuersignals am Ausgangspin beobachtbar machen zu können, wird der I/O-Pin von außen auf High (1) gesetzt und der Wert im Zustand „CaptureDR“ in die BSC Input gespeichert. Danach wird diese 1 in den Zustand „Shift-DR“ in die BSC Control-Signal geschoben. Simultan wird der Wert der BSC Control-Signal in BSC Output mit geschoben. Nun ist die BSC Control-Signal mit 1 und die BSC Output mit dem aus der Core-Logik belieferten Control-Signal Wert belegt. Während der Instruktion „Extest“ ist die BSC Output am Pin beobachtbar. Das Messergebnis zeigte eine Null am Ausgang. Da eine weitere Steuerung des Signals hinter der BSC nicht möglich ist, kann man feststellen, dass die Core-Logik bei Boundary-Scan-Instruktionen das Treibersteuersignal auf Null setzt und für das Fehlverhalten am Pin verantwortlich ist. Genauso sieht man am Ausgang TDO das Fehlverhalten, da er über einen Treiber gesteuert wird. Eine mögliche Interpretation lautet: Als Sicherheitsfeature werden bei niederfrequenten Betrieb von JTAG alle Ausgänge auf High-Z gesetzt. Das Feature sorgt dafür, dass vom Ausgang her - falls er nicht ordnungsgemäß betrieben wird - nichts abgezapft werden kann. Es könnten Änderungen an Ausgängen propagiert werden, falls Eingänge ihre Belegungen ändern.

4 Zusammenfassung

In diesem Paper wird gezeigt, wie man einen Fehler sucht, den man nicht kennt, und dabei trotzdem seinen Fehlerort bestimmen kann. Als Beispiel wurde das Fehlverhalten eines µ-Controller im JTAG während des niederfrequenten Betriebs beobachtet. Man sucht nach dem Fehler indem man den D-Algorithmus mit neuer Zielsetzung verwendet. Dieser Algorithmus muss die Realität (die Richtung und die transitive Hülle) nachbilden können. Er sei damit strukturtreu: Man vermutet erst einmal was der Fehler sein könnte und nimmt dann das Fehlermodell an. Dann generiert man das Testpattern (unter Berücksichtigung der Beobachtbarkeit und Steuerbarkeit) für genau dieses Fehlermodell und schließlich schaut man, ob der Fehler auftritt oder nicht. Wichtig ist, dass die Suchrichtung gerichtet und damit umkehrbar eindeutig ist. Dadurch gerät man nicht ins Ausprobieren aller Möglichkeiten. Dieser D-Algorithmus mit veränderter und neuer Zielrichtung soll verallgemeinert und implementiert werden. Die Effizienz der neuen Methode soll später an verschiedenen Schaltungen unter Beweis gestellt werden.

5 Literatur

[1] Y. Xu: Algorithms for automatic generation of relative timing constraints, Ph.D. dissertation, Salt Lake City,UT, USA, 2011.

[2] R. P. Kurshan; K. L. McMillan: Analysis of digital circuits through symbolic reduction IEEE Trans. on CAD of Integrated Circuits and Systems, vol. 10, no. 11, pp. 1356–1371, 1991.

[3] O.H. Ibarra and S.K. Sahni: Polynomially complete fault detection problems. IEEE Transactions on Computers, C-24(3):242_249, 1975.

[4] J. Sziray: Computational complexity in logic testing. In 2010 IEEE 14th International Conference on Intelligent Engineering Systems, page nil, 5 2010.

[5] Z. Navabi: Digital system test and testable design: using HDL models and architectures. Springer, 2011.

[6] J. Paul Roth: Diagnosis of automata failures: A calculus and a method. IBM Journal of Research and Development, 10(4):278_291, 1966.

[7] IEEE Standard Association: IEEE Standard for Test Access Port and Boundary-Scan Architecture, IEEE Std 1149.1™-2013

[8] Datasheet PIC32MX1XX/2XX 28/36/44-PIN FAMILY: Microchip Technology Inc. 2011-2016

[9] PIC32 Family Reference Manual, Section 12. I/O Ports: Microchip Technology Inc. 2007-2015

[10] PIC32 Family Reference Manual, Section 33. Programming and Diagnostics: Microchip Technology Inc. 2007-2012