Hardware/Software Co-design

Protokoll Tag 2

Fritz Lukas

Steurer Elias

Aufgabe 2)

Das Modul „host_command“ ist hier um eine Decryption-Einheit (Modul „decrypt“) erweitert, welche verschlüsselte Pakete (Control-Byte 0x80) entschlüsselt. Des weiteren sollen Control und Logger zum Modell hinzugefügt werden.

Hauptaufgabe ist es dann, den Controller bestehend aus einem Befehlsdecoder und den 2 Statemachines FSM1 und FSM2 zu implementieren.

Top

Im top.h werden die Signale miteinander verlinkt, wie z.Bsp. die Buffer für machine1 und machine2. Sie verbinden den Control-Block mit dem Logger-Block. Dadurch werden die Command entschlüsselt und der entsprechenden FSM zugeordnet.

Control

Die internen Signale zwischen Controller und State-Machines müssen hier verbunden werden. Die State-Machines werden innerhalb von Threads ausgeführt. Eine Hilfsfunktion zur Dekodierung der Bytes wurde zusätzlich hinzugefügt.

Die Pakete, welche in Task1 von empfangen Bytes generiert wurden, müssen nun entsprechend aufgeschlüsselt und verarbeitet werden.

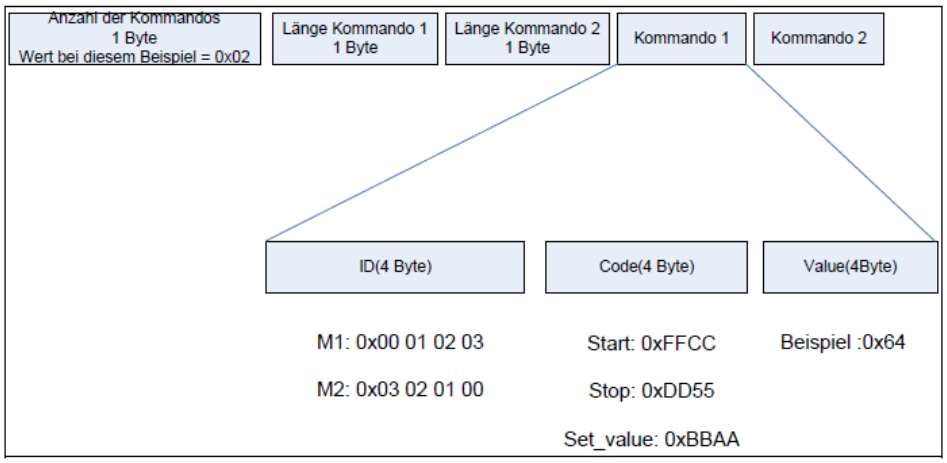

Abbildung 6: Aufbau eines Datenpaketes

Wie in obiger Abbildung zu sehen, enthalten die Pakete Daten zu Steurerung der FSMs. Für Details siehe Abbildung oben.

Die beiden States der FSMs sind STOP und RUN und Sie werden nachfolgend erklärt.

STOP: Sobald ein M_START-Kommando empfangen wird wechselt die State Machine in den State RUN.
RUN: Wird ein M_SET-Kommando empfangen, bleibt die State Machine im State RUN. Wird anderenfalls ein M_STOP-Kommando empfangen, wehcselt sie in den State STOP.

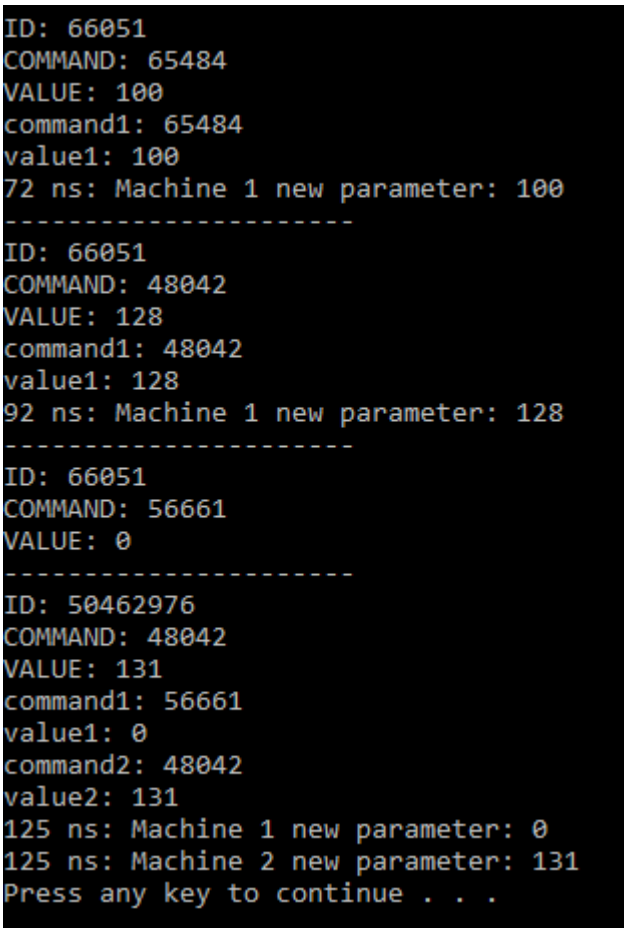

Abbildung 4: Output der erfolgreichen Durchführung
Zur besseren Darstellung wurden zusätzliche Prints für die ID, Command und Value eingefügt.

Aufgabe 3)
Ab dieser Aufgabe wurde das Modell des Gesamtsystems implementiert. Das aus den vorhergehenden Aufgaben bearbeitete System wurde mit einem FIFO-System erweitert. Es wurde überprüft ob das System auch mit Delays in der Übertragung umgehen kann. Durch diese Art der Kommunikation werden Packete zwischengespeichert und es gibt dadurch keine Datenverluste mehr.

Weil es einen Namenskonflikt mit der Klasse generate gab, wurde dies umbenannt in „generate2.cpp“ bzw. „generate2.h“.

Die jeweiligen Trigger der Threads wurden verändert. Thread „input“ triggert nun auf das in_generate Signal und schreibt anschließend Daten in den FIFO-Buffer. Thread „transmitter“ triggert auf das data_written Event, ausgelöst vom FIFO-Buffer.

```
Info: (I702) default timescale unit used for tracing: 1 ps (trace.vcd)
19 ns: Machine 1 new parameter: 100
----------------------> Machine received <----------------------
52 ns: Machine 1 new parameter: 0
52 ns: Machine 2 new parameter: 100
----------------------> Machine received <----------------------
72 ns: Machine 1 new parameter: 100
----------------------> Machine received <----------------------
92 ns: Machine 1 new parameter: 128
125 ns: Machine 1 new parameter: 0
125 ns: Machine 2 new parameter: 131
----------------------> Machine received <----------------------
----------------------> Machine received <----------------------
Press any key to continue . . .
```

Abbildung 5: Output der erfolgreichen Durchführung

Aufgabe 4)

Bisher waren die Komponenten des Systems alle direkt miteinander verbunden. In Übung 4 wird das System auf eine Zielarchitektur abgebildet. Der Controller agiert als Master und greift über ein Bussystem auf die Module „host_command“, „machine_command“(1) und „machine_command“(2) zu.

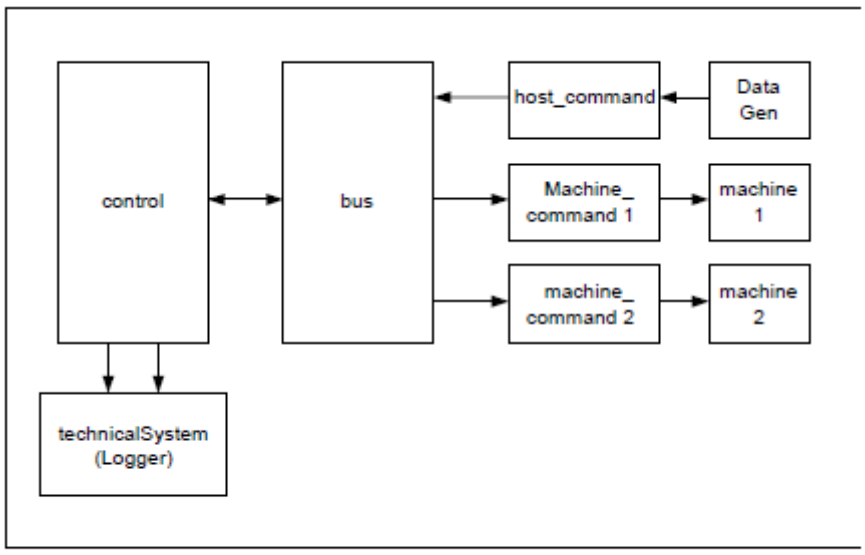

Abbildung 8: Gesamtsystem inklusive Busarchitektur

Die Hauptänderung in dieser Übung war einen Bus zu implementieren und ihn im Modell als Kommunikationsmedium zu nutzen. Dazu musste an mehreren Orten auf den Bus gewechselt werden. Ein Hauptvorteil des Busses ist zum Beispiel die exalte Zuweisung eines Adressbereichs auf einzelne Module und auch der Zugriff auf mehrere Slaves über einen Port.

Die FSMs schreiben nach unseren Änderungen auf den Bus und der Controller liest vom Bus anstatt der ürsprünglichen Methoden.

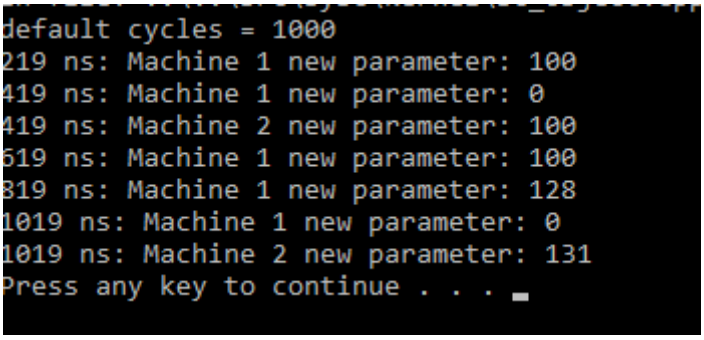

Abbildung 6: Output der erfolgreichen Durchführung

Die Bus.cpp wurde um Methoden für Lesen und Schreiben erweitert. Control.h wurde um einen Bus-Port zur Kommunikation erweitert und in der Control.cpp wurde das Auslesen des Buses implementiert. Ebenfalls wurde statt dem Output-Signal an den Bus geschrieben. Abschließend wurden die Änderungen der Interfaces im top.h geändert und an die neuen Gegebenheiten angepasst. Das Control-Modul wird als Master an den Bus angeschlossen und die Slaves erhalten eine Start- bzw. Endadresse.