Inhaltsverzeichnis

1 Einleitung

1.1 Industriepartner

2 Debugger

Es gibt diverse Debugger auf dem Markt. In dieser Arbeit beschränke ich mich aber auf den GNU-Debugger (*gdb*). Der *gdb* steht unter der PGL Lizenz und ist somit Open Source. Bei den meisten Linux Distributionen wird der *gdb* direkt mitgeliefert und kann sofort verwendet werden.

2.1 Anwendungsbeispiel gdb auf einem Linux-System

Mit diesem Beispiel will ich zum einen ein kurzes Tutorial bieten um den Umgang mit dem *gdb* zu lernen. Zum anderen will ich damit aber auch die verschiedenen Funktionen vom Debugger zeigen, damit die beschriebenen Probleme später im Kapitel besser in einen Kontext gestellt werden können.

Für dieses Tutorial verwende ich ein Linux Mint 18.1 (basierend auf einem Ubuntu 16.01). Solange *gdb* installiert ist, ist das verwendete Betriebssystem aber nicht relevant.

2.1.1 Grundlegende Funktionsweise

Auf Linux verwendet der *gdb* den System Call *ptrace* (Kurzform für "process trace"). Dieser System Call erlaubt dem *gdb* einen anderen Prozess zu inspizieren und zu manipulieren. Im Hardwaredebugger, den wir später bearbeiten, verwenden wir stattdessen JTAG in Verbindung mit der Debugginghardware im Prozessor.

2.1.2 Vorbereitung

Für dieses Tutorial verwenden wir folgendes Beispielprogramm:

```
1  #include <iostream>
2  using namespace std;
3
4  int divint(int, int);
5  int main()
6  {
7      int x = 5, y = 2;
8      cout << divint(x, y);
9
10     x =3; y = 0;
11     cout << divint(x, y);
12
13     return 0;
14 }
15
16 int divint(int a, int b)
17 {
18     return a / b;
19 }
```

Diese Applikation können wir jetzt Kompilieren und mit *gdb* starten:

```
# g++ gdbTest.cpp -o gdbTest
# gdb gdbTest
# run
```
// startet die Applikation im *gdb*

```
(gdb) run
Starting program: /home/mgehrig2/projects/gdbTest/gdbTest
```

Program received signal SIGFPE, Arithmetic exception.
0x00000000004007b5 in divint(int, int) ()

Obwohl die Applikation nicht mit Debug-Symbolen kompiliert wurde, wird nicht nur die Adresse des Ursprungs der Floating Point Exception angezeigt, sondern auch der Name der Methode.

2.2 Unterschied zwischen einem Software- und Hardwaredebugger

Auf einem Linux-System ist es sehr einfach ein Debugger einzusetzen.

2.3 Funktionen eines Debuggers

Ein Debugger kann verschiedene Funktionen besitzen. Die grundlegenden Funktionen sind sehr einfach und brauchen keine grosse Ïntelligenz"vom Debugger selber.

Erweiterte Funktionen erwarten vom

2.4 Erstellen einer Dummy-Applikation mit Debug-Informationen

2.4.1 Vorgehen

Das Ziel von diesem Kapitel ist es, eine Deep-Applikation zu erzeugen, die mit *gdb* und *OpenOCD*

Anhang