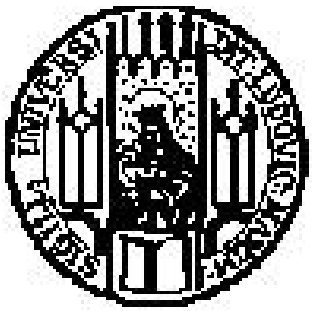
INSTITUT FÜR INFORMATIK
DER LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN



Diplomarbeit

Benchmarking von Java Cards

Monika Erdmann

INSTITUT FÜR INFORMATIK
DER LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Diplomarbeit

Benchmarking von Java Cards

Monika Erdmann

Aufgabensteller: Prof. Dr. Heinz-Gerd Hegering
Betreuer: Bernhard Lippmann (Infineon Technologies AG)
Dr. Helmut Reiser

Abgabetermin: 24.Mai 2004

Erklärung

Hiermit versichere ich, dass ich diese Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 24. Mai 2004

……..
Monika Erdmann

Zusammenfassung

In immer mehr Bereichen finden Chipkarten, die durch den integrierten Prozessor einfach zu nutzende Rechenkapazität an jedem beliebigen Ort bereitstellen, Verwendung. Lange war es so, dass für herkömmliche Smart Cards aufgrund ihrer umständlichen Programmierung der Zeitraum für die Entwicklung bis zur Marktreife recht groß war. Außerdem war es kaum möglich, mehr als eine Applikation auf einer Karte laufen zu lassen. Dies hat sich aber durch die Einführung neuer Smart Card Typen geändert, da diese mittlerweile über Kartenbetriebssysteme verfügen, die nicht nur Mehrfachanwendungen ermöglichen, sondern die in einer Hochsprache wie Java programmiert werden können, oder die es erlauben, dass Applikationen dynamisch nachgeladen werden. Solche java-basierten Karten werden auch Java Cards genannt.
Durch die immer komplexer gewordenen Applikationen und Mehrfachanwendungen haben sich die Anforderungen an eine Chipkarte nicht nur erhöht, sondern dies hat auch dazu geführt, dass je nach Anwendungsbereich eine optimale Zusammensetzung der Hardware- und Softwarekomponenten benötigt wird.
Deshalb soll im Rahmen dieser Arbeit ein Benchmarking für Java Cards durchgeführt und die Ergebnisse präsentiert werden.
Dazu wird nach einer Einleitung der Stand der Technik beschrieben, um im Anschluss eine Metrik zu definieren, wie das Benchmarking durchzuführen ist, und wie die Ergebnisse auszuwerten sind, damit man einen repräsentativen Vergleich erhält. Danach wird für die Erstellung von Applets auf das Applet Design eingegangen. Als Letztes wird der Messaufbau beschrieben, um dann mit den Ergebnissen des Java Card Benchmarking abzuschließen.

Danksagung

Diese Arbeit entstand im Rahmen meiner Diplomprüfungen am Lehrstuhl für Kommunikationssysteme und Systemprogrammierung der Universität München in Kooperation mit Infineon Technologies AG mit Sitz in München.

Auf Seiten der Universität gilt mein Dank meinem Betreuer Dr. Helmut Reiser, der mich mit seiner konstruktiven Kritik während der Ausarbeitung dieser Arbeit begleitet hat.

Auf Seiten von Infineon Technologies will ich mich insbesondere bei Bernhard Lippmann für seine hervorragende Unterstützung bedanken. Dank gilt auch allen anderen Mitarbeitern und Kollegen, die mir auf irgendeine Art und Weise geholfen haben.

Meinen Eltern und meinem Freund will ich für den seelischen Rückhalt und ihre Motivation danken.

München, 24. Mai 2004

INHALTSVERZEICHNIS

1. Einleitung

In den Anfängen der Chipkarte (Smart Card) ahnte noch niemand, wie vielseitig deren Anwendungen in Zukunft sein würde. Im Laufe der Zeit entwickelte sie sich vom reinen Datenspeicher (Speicherchipkarten) über einen einfachen Spezialprozessor (Mikrocontrollerkarte) zum vollwertigen, universellen Prozessor. Somit kann man die Chipkarte als Computer im Kleinstformat sehen. Man muss dabei bedenken, dass die Chipkarten von der Leistung her den Computer von heute bei Weitem nicht entsprechen. Die Leistung ist mit der von früheren, alten Computern vergleichbar. Darüber hinaus ist die Chipkarte streng genommen kein vollwertiger Computer, da ihr eine eigene Stromversorgung, sowie eine eigene Möglichkeit der Ein- und Ausgabe fehlt. Beides wird heutzutage noch von den Kartenterminals übernommen.

Aufgrund ihrer praktischen und einfachen Handhabung und ihrer vielseitigen Verwendbarkeit erlangten die Chipkarten eine große Verbreitung. Daher eroberten sie weite Bereiche des alltäglichen Lebens wie zum Beispiel den elektronischen Zahlungsverkehr oder das Gesundheitswesen. Ferner eignen sie sich hervorragend, um den Sicherheitsaspekt zu befriedigen, da durch logische und physikalische Schutzmechanismen nicht autorisierte Zugriffe auf die Daten oder geheimen Schlüsseln, die in den Chips gespeichert sind, verhindert werden. Außerdem stehen durch die integrierten Kryptocoprozessoren schnelle, kryptographische Berechnungen zur Verfügung. Wegen diesen Eigenschaften können Chipkarten sehr gut als portables Medium zur Identifikation und zur Autorisierung verwendet werden.

Smart Cards, die auf der Hochsprache Java basieren, nennt man Java Cards. Ihnen soll im Rahmen dieser Arbeit die ganze Aufmerksamkeit gewidmet werden, da die Java Card zurzeit und auch auf absehbare Zeit die einzig weltweit verbreitete Technologie ist, um programmbasierte Anwendungen auf Chipkarten zu realisieren.
Neben der Verbreitung und Akzeptanz der Smart Card (Java Card) wuchsen auch die Anforderungen an sie. Komplexere Anwendungen und wachsender Konkurrenz- und Kostendruck bedingen neue Konzepte in der Chipkartenwelt, zumal die mittelfristig erreichte Kompatibilität der Java Cards die Kartenhersteller austauschbar werden lässt. Je nach Art und Zielsetzung der Anwendungen werden verschiedene Anforderungen an die Karte gestellt, wie die Zusammensetzung der Hardware- und Softwarekomponenten der Chipkarte. So ist es zum Beispiel wenig sinnvoll für eine Krankenversichertenkarte, eine Java Card zu verwenden, die einen Kryptocoprozessor enthält, da die Art der Anwendung dies gar nicht verlangt. Man würde dadurch wertvolle Ressourcen auf der Karte verschwenden.

Um die optimale Zusammensetzung der Hardware und Software einer Karte abhängig von der Anwendung zu bestimmen, soll in dieser Arbeit ein Benchmarking von Java Cards durchgeführt werden.

Damit lässt sich für die Durchführung dieses Benchmarkings folgende Aufgabenstellung herleiten, die durch die nachstehenden Kapitel abgearbeitet wird:

- Untersuchung bestehender Benchmarking-Modelle
- Erarbeiten einer Benchmarking – Performance Metrik für Java Cards
- Erstellen von Test-Applets zur Durchführung von Java Card Performance Messungen
- Erstellen einer geeigneten Messumgebung zur Durchführung von Performance Messungen, Durchführung von Performance Messungen
- Bewertung der Messungen, Anregungen für Produktverbesserungen

2. Stand der Technik

Bevor man sich der eigentlichen Problematik zuwenden kann, ist es sinnvoll sich genauer mit der Chipkarte und der dahinter liegenden Technologie zu befassen, um Verständnisschwierigkeiten zu vermeiden. Deshalb soll in diesem Kapitel anfangend mit der Geschichte der Chipkarte und dem Aufbau derselben über die Konzepte und Komponenten der Java Card Technologie auf die Entwicklung und den Aufbau eines Java Card Applets eingegangen werden, um abschließend ausgewählte Java Card Entwicklungsumgebungen vorzustellen.

2.1. Geschichte und Anwendungen von Chipkarten

In den 50er Jahren begann in den USA die Geschichte der Chipkarten. Damals wurden erstmals Plastikkarten, die die Papierkarten ersetzten, stark verbreitet. Nachdem sich die langlebige, sicherere und viel robustere Plastikkarte im Zahlungsverkehr in den USA bewährt hatte, verbreitete sie sich sehr schnell auch in Europa. Vorerst waren auf den Karten nur aufgedruckte und aufgeprägte Daten vorhanden. Manche hatten auch ein Unterschriftsfeld. Danach folgte ein Magnetstreifen, auf dem zusätzliche Daten in maschinenlesbarer Form, digital gespeichert waren. Als nächste Verbesserung wurde eine persönliche Geheimzahl (PIN, Personal Identification Number) eingeführt. Leider konnten die Daten, die auf dem Magnetstreifen gespeichert wurden, mittels einer Schreib-/Lesevorrichtung für Magnetkarten beliebig verändert werden, so dass die Informationen nicht geheim bleiben konnten. Man brauchte eine bessere Lösung.

Schon 1968 haben die Erfinder J. Dethloff und H. Gröntrupp ein Patent für einen integrierten Schaltkreis in einer Identifikationskarte angemeldet. Die schnelle Entwicklung der Mikroelektronik in den 70er Jahren hat neue Möglichkeiten geschaffen. So war es möglich einen Datenspeicher und eine Einheit für Arithmetik und Logik auf einem einzigen, kleinen Chip unterzubringen. 1974 hat R. Moreno sein Chipkartenpatent in Frankreich angemeldet. Ab dann war auch die Halbleiterindustrie in der Lage, die integrierten Schaltkreise zu produzieren und zu akzeptablen Preisen zu verkaufen. Zehn Jahre später haben erfolgreiche Feldversuche mit Telefonkarten in Frankreich zum Durchbruch dieser Karte geführt. Auch in Deutschland wurde 1984/85 ein Pilotversuch mit Telefonkarten durchgeführt. Beide Versuche haben die erwartete hohe Zuverlässigkeit und Manipulationssicherheit von Chipkarten bestätigt. 1988 hat die Deutsche Bundespost eine moderne Mikroprozessorkarte mit der EEPROM (Electrically Erasable Programmable Read Only Memory) Technologie im Mobilfunk eingesetzt. Im Gegensatz zu den einfachen, bei Telefonkarten eingesetzten Schaltkreisen wurden bei den Mobilfunkkarten größere und komplexere Mikroprozessorchips verwendet. 1991 wurden die Chipkarten im digitalen GSM-Netz eingeführt.

1997, nachdem die modernen kryptographischen Verfahren eingesetzt wurden, gaben auch Sparkassen und andere Banken die neuen Chipkarten für den Zahlungsverkehr heraus.

Mitte der 90er Jahre wurden die ersten Java Cards entwickelt. Dabei handelte es sich um Chipkarten, die in der Hochsprache Java programmiert wurden. Die benötigte Speicherkapazität und Prozessorleistung für diese Chipkarten war jedoch zu hoch und bewirkte, dass eine Einschränkung des Befehls- und Funktionsumfanges von Java notwendig war. Die erste Java Card 1.0 Spezifikation wurde Ende 1996 veröffentlicht, und Anfang 1997 wurde die erste Java Card (von der Firma Schlumberger) präsentiert. 1997 wurde das Java Card Forum (JCF) von den führenden Kartenherstellern gegründet. Das Konsortium beschäftigt sich bis heute mit der Definition einer für Chipkarten geeigneten Java API (Application Programming Interface), der Spezifikation der Java Card VM (Virtual Machine) einschließlich ihrer Eigenschaften, dem Austausch von Informationen bezüglich der Implementierung der Java Card API, der Erstellung technischer Dokumente und unterstützt Software- und Hardwareentwickler im Kontext der Java Card. Die Java Card 2.0 Spezifikation [Sun3] wurde Ende1997 beschrieben und beinhaltet die Java Card 2.0 Language Subset Specification [Sun2] und Java Card 2.0 API [Sun1]. Ab 1999 wird Java Card als Standard in GSM genormt. Im gleichen Jahr wurde auch die Java Card 2.1 Spezifikation (API, VM, JCRE - Java Card Runtime Environment) eingeführt. Die ersten multifunktionalen Karten, die mehrere Anwendungen enthalten, wurden erst 2003 verwendet. Dank moderner Chipkarten-Betriebssysteme besteht die Möglichkeit, auch nach der Ausgabe der Chipkarte an den Benutzer neue Anwendungen auf die Karte zu laden.

Chipkarten finden in immer mehr Bereichen Verwendung, nämlich in Telefonanwendungen, im Personalverkehr, in der Krankenversicherung, im Zahlungsverkehr, usw. Für verschiedene Chipkarten werden unterschiedliche Anwendungsschwerpunkte gelegt.

Die Chipkarten werden nach Chiptyp und nach Datenübertragung klassifiziert (siehe dazu Abbildung 2.1.1). Die Klassifizierung nach Chiptyp wird in Speicherkarten und Mikroprozessorkarten eingeteilt. Die Speicherkarten werden in der Telekommunikation, als Telefonkarten, und dort, wo gegen Vorbezahlung Waren oder Dienste bargeldlos verkauft werden, also z. B. in Kantinen, Schwimmbädern, Parkhäusern und Verkaufsautomaten angewendet. In Deutschland wird diese Art von Karten als Krankenversichertenkarte eingesetzt. Die Mikroprozessorkarten werden vor allem als Bankkarte, aber auch als Identifikationskarten bei Zugangskontrollen, in der Telekommunikation (GSM Mobilfunknetz), im Privatfernsehen (Decoderkarten) oder als multifunktionale Karten verwendet. Die Klassifizierung nach Datenübertragung unterteilt die Chipkarten in kontaktlose, kontaktbehaftete und dual-interface Karten. Da die kontaktlosen und dual-interface Karten teurer als die kontaktbehafteten Karten sind, werden sie nicht überall eingesetzt. Wo aber Zeit eine große Rolle spielt (z.B. Zutrittskontrollen oder Fahrkarten) wird oft der höhere Preis in Kauf genommen.

Unterschiedliche Applikationen müssen verschiedene Funktionen erfüllen. Dazu ist eine optimal auf die Applikation abgestimmte Hardware notwendig. Nicht alle Chipkarten (z.B. Identifikationskarten) brauchen einen Kryptocoprozessor oder einen schnellen Prozessor (z.B. die Krankenversichertenkarte).

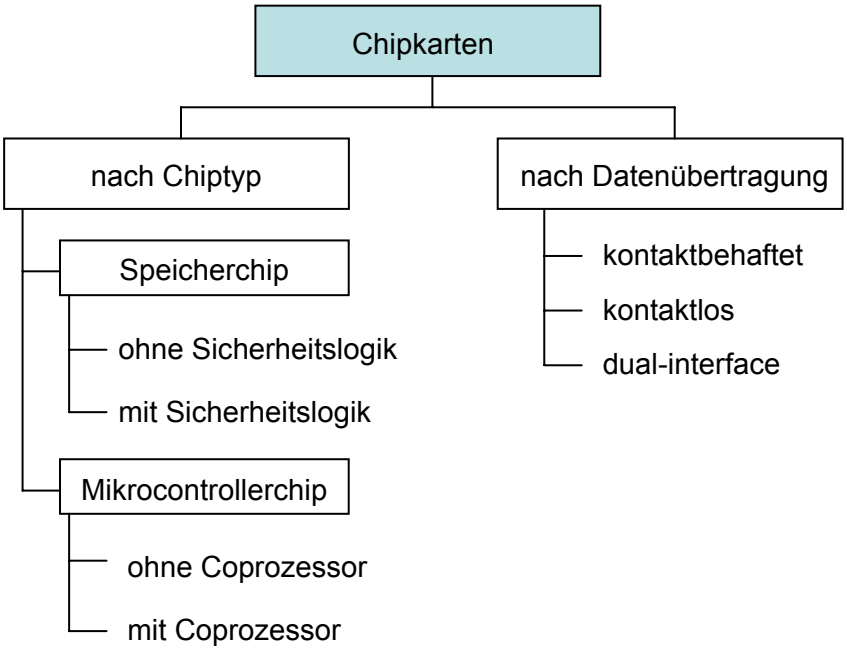


Abbildung 2.1.1 Klassifizierung der Karten mit einem Chip

2.2. Wertschöpfungskette von Chipkarten

Die Wertschöpfungskette einer Chipkarte beginnt mit der Entstehung des Halbleiters, geht über die Kartenherstellung und endet mit dem Recycling der Karte.
Eine Chipkarte besteht aus zwei Komponenten: zum einem aus dem Kartenkörper mit Aufdruck, Sicherheitsmerkmalen und (falls vorhanden) einem Magnetstreifen, zum anderem aus dem Modul, der einen Chip enthält.

Der Lebenszyklus einer Chipkarte ist von der zukünftigen Anwendung der Karte abhängig und ist in fünf Phasen unterteilt. In der ersten Phase werden der Chip und die Chipkarte hergestellt. In dieser Phase wird der Chip designed, das Betriebssystem der Chipkarte erstellt, die Chips und Module produziert, der Kartenkörper hergestellt und schließlich werden die Module in den Kartenkörper eingebettet. Die zweite Phase beschäftigt sich ausschließlich mit der Funktionalität der Karte. Hier wird das Chipkarten-Betriebssystem komplettiert. In der dritten Phase werden alle Anwendungsvorbereitungen getroffen. Es wird mit der Initialisierung der Anwendung begonnen, dann werden die Anwendungen optisch und elektrisch personalisiert.

Die Personalisierung, auch Individualisierung genannt, bedeutet, dass alle Daten, die einer einzelner Person oder einer einzelnen Karte zugeordnet sind, in die Karte ein bzw. aufgebracht werden. Das können z.B. der Vorname und der Nachname der Karteninhaber oder auch kartenbezogene Schlüssel sein. Auf jeden Fall sind das kartenindividuelle Daten. Die vierte Phase gehört der Kartenbenutzung. Erst hier werden die Anwendungen aktiviert und irgendwann deaktiviert. Die fünfte Phase ist das Ende der Kartenbenutzung; dann werden alle Anwendungen deaktiviert und anschließend findet auch die Deaktivierung der Karte statt.

Die Abbildung 2.2.1 stellt die Wertschöpfungskette von Java Card dar. Auf der rechten Seite stehen Namen von Firmen, die bei den jeweiligen Phasen mitwirken, bzw. Anwendungen der Java Cards.

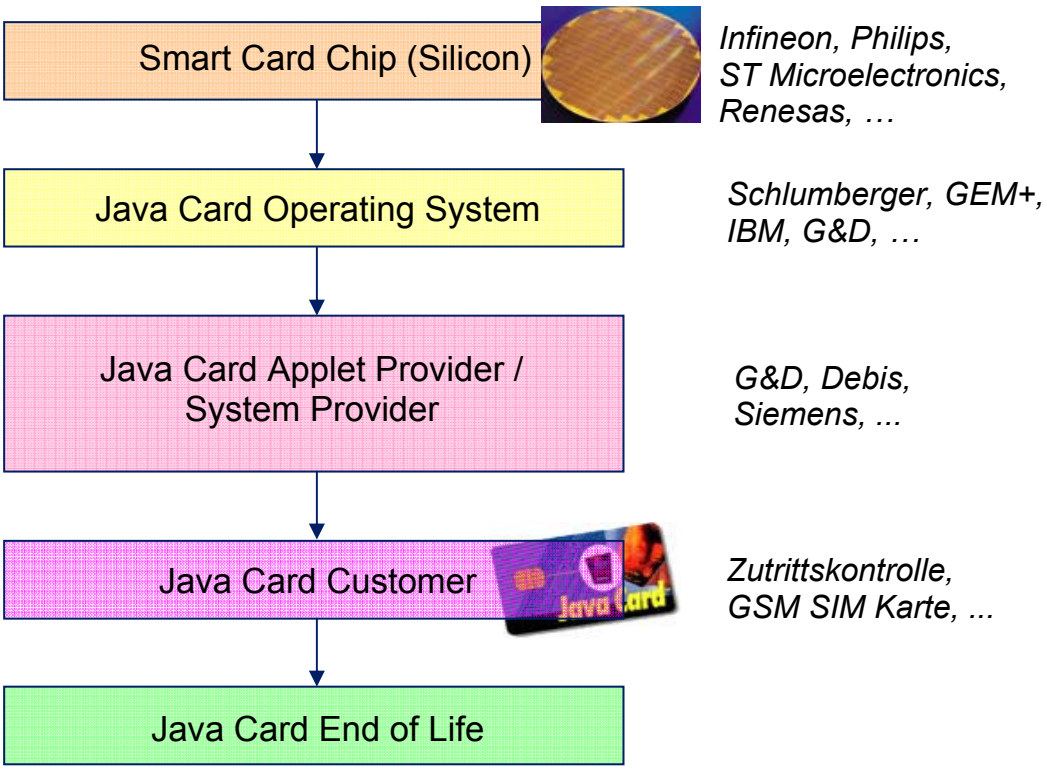


Abbildung 2.2.1 Wertschöpfungskette von Java Card

Die genauere Beschreibung eines Kartenlebenszyklus definiert die [ISO 10-202-1] (International Standardisation Organization). Alle Fertigungsschritte werden mit Chargen- und Chipnummern protokolliert. Somit ist es nach der Produktion möglich, die einzelnen Fertigungsschritte für jede Chipkarte nachzuvollziehen. Dies ist sehr hilfreich, wenn man die Ursachen für auftretende Fertigungsfehler sucht.

2.3. Aufbau von Chipkarten

Eine Java Card gehört zur Klasse der Mikroprozessorkarten. Eine Mikroprozessorkarte besteht aus einem Prozessor (CPU, Central Processing Unit), einem Kryptocoprozessor (NPU, Numeric Processing Unit), dem ROM (Read Only Memory), dem RAM (Random Access Memory), dem EEPROM (Electrically Erasable Programmable Read Only Memory) und dem I/O-Port. Die Abbildung 2.3.1 stellt alle Elemente einer Chipkarte graphisch dar.

Auf dem ROM befindet sich das Betriebssystem der Karte. Dieses wird während der Herstellung eingebrannt und bleibt unveränderbar gespeichert. Der Arbeitsspeicher des Prozessors ist das RAM. Die dort gespeicherten Daten sind flüchtig und werden gelöscht, sobald die Versorgungsspannung des Chips abgeschaltet wird. Das EEPROM dient als Datenspeicher - dieser Speicherbereich ist nicht flüchtig. Die Daten, auch der Programmcode, können dort unter Kontrolle des Betriebssystems gelesen und geschrieben werden. Auf dem EEPROM werden auch Betriebssystemteile gespeichert. Das I/O-Port besteht aus einem einzigen Register. Über diese serielle Schnittstelle werden die Daten Bit für Bit übertragen. Der Coprozessor wird meist für Kryptoalgorithmen benutzt. Normalerweise ist eine Mikroprozessorkarte für eine Anwendung verwendbar. Es ist aber möglich mehrere unterschiedliche Anwendungen in eine Mikroprozessorkarte zu integrieren. Diese Programme können später, auch nach dem die Karte personalisiert wurde, nachgeladen werden.

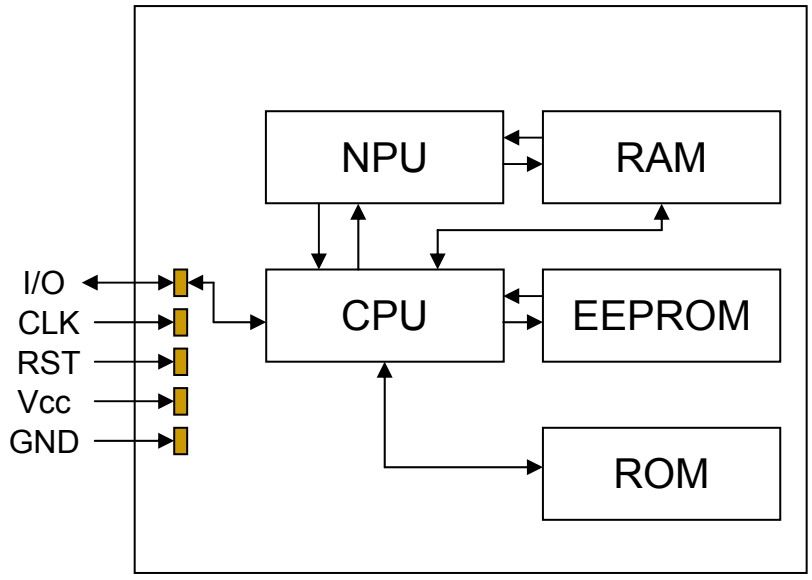


Abbildung 2.3.1 Aufbau einer Chipkarte

Die Abbildung 2.3.2 zeigt ein Foto einer Mikroprozessorkarte. Sie besteht aus der CPU, dem ROM, dem EEPROM und dem RAM. Das RAM besteht in diesem Fall aus zwei Teilen. Man versucht RAM und EEPROM relativ klein zu halten, da diese den meisten Platz (pro Bit) auf der Karte benötigen (siehe hierzu Abbildung 2.3.3 und [RE02]).

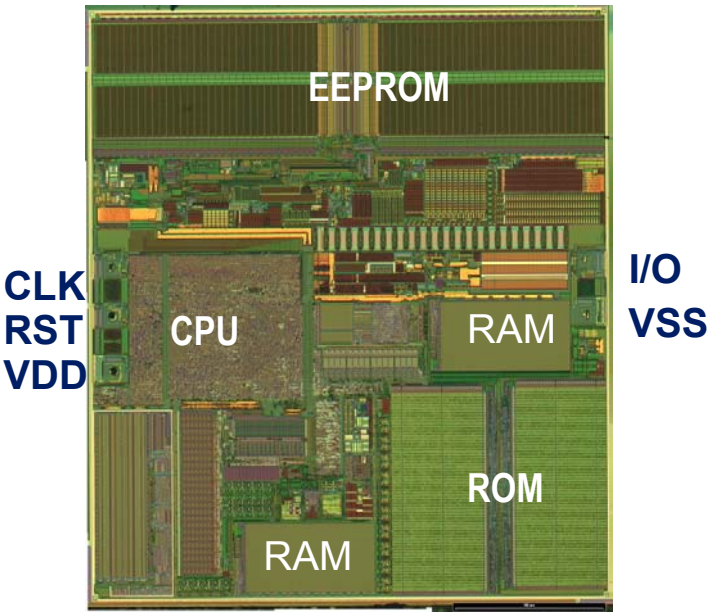


Abbildung 2.3.2 Aufbau einer Chipkarte (Foto, Infineon Technologies)

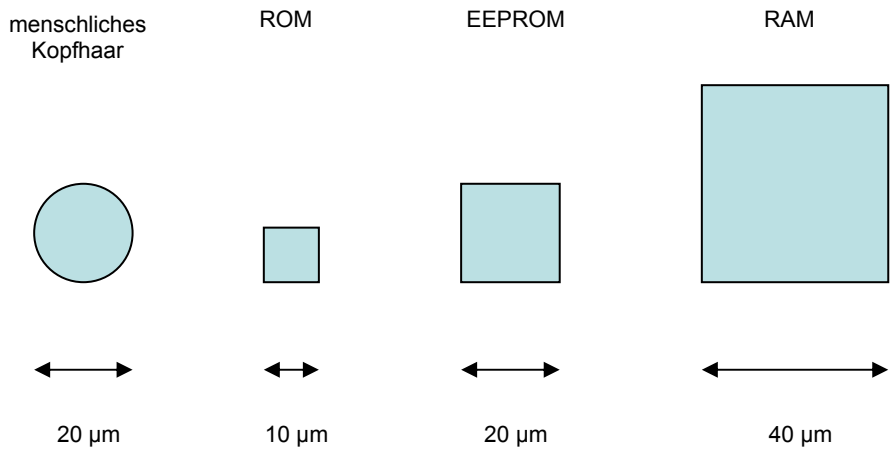


Abbildung 2.3.3 Vergleich des Platzbedarf für je 1 Bit in Abhängigkeit von der Speicherart

Chipkarten haben an der Vorderseite acht bzw. sechs Kontaktflächen, nämlich Versorgungsspannung (Vcc), Reset Eingang (RST), Clock (CLK), Masse (GDN), Programmier- und Löschspannung (Vpp), Ein-/Ausgang für die serielle Kommunikation (I/O) und die Hilfskontakte (AUX1 und AUX2).

Alle elektrischen Signale werden über diese Schnittstellen zwischen dem Terminal und dem Mikrokontroller der Chipkarten ausgetauscht. Die Abbildung 2.3.4 zeigt die elektrische Belegung der Kontaktflächen.

Die Kontaktfläche Vcc versorgt die Karte mit einem Betriebsstrom von 3-5 Volt. Beim Reset wird die Chipkarte auf einen klar definierten Ausgangszustand zurückgesetzt. Man unterscheidet zwischen Warm Reset und Cold Reset (Power on Reset). Ein Warm Reset wird durch ein Signal auf der Resetleitung zur Chipkarte ausgeführt, und die Versorgungsspannung bleibt davon unberührt. Ein Cold Reset wird durchgeführt, wenn der Strom abgeschaltet und wieder eingeschaltet wird (z.B. beim Entnehmen und Wiedereinstecken der Karte in das Terminal). Der Eingang für den Clock (Takt) bietet ein Zeitsignal für die Karte, meist 3,5-5 MHz. Die Programmier- und Löschspannung für das EEPROM wird seit einigen Jahren nicht mehr benutzt. Die Hilfskontakte werden momentan nicht benutzt, sind aber für die Zukunft reserviert, z.B. für die USB-Schnittstelle. Sie werden bei Chipkarten mit sechs Kontaktflächen weggelassen, um die Fertigungskosten zu senken.

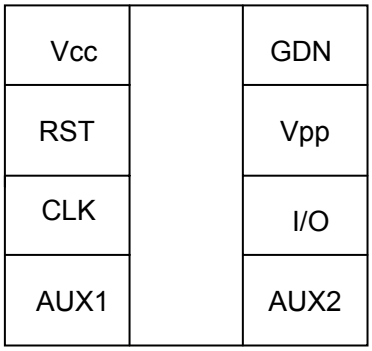


Abbildung 2.3.4 Kontaktfläche bei einer Chipkarte

2.4. Warum Java Card?

Chipkarten auf denen die Programmiersprache Java verfügbar ist, profitieren von fast allen Vorteilen dieser Hochsprache. Java ist leicht zu programmieren, robust, objektorientiert, stark typisiert und sicher. Deswegen ist die Verwendung von Java so interessant. Aufgrund zu niedriger Speicherkapazitäten und der Prozessorleistung auf der Chipkarte wird nicht das vollständige Java, sondern nur eine Untermenge von Java bei den Chipkarten benutzt. Diese Untermenge heißt Java Card (Java für Chipkarten).

Anders als bei herkömmlichen Smart Cards, wo Betriebssystem und Anwendungen nicht klar voneinander getrennt sind, haben die in Java Card programmierten Chipkarten neben einem beliebigen Betriebssystem einen Java Card Interpreter (Java Card Virtual Machine). Java für Chipkarten wird dadurch in Bytecode und nicht in Maschinencode übersetzt. Die nicht in Java implementierten Chipkarten verschiedener Hersteller waren nicht kompatibel, dagegen laufen die mit Java für Chipkarten geschriebenen Anwendungen auf jeder Smart Card, die über eine Java Card VM verfügt. Damit erreicht man Plattformunabhängigkeit und Portierbarkeit.

Da Java für Chipkarten in sicherheitsrelevanten Anwendungen eingesetzt werden, ist die eingebaute Robustheit und Sicherheit von Java von großer Bedeutung. Die wird dadurch gegeben, dass Java durch einen kontrollierten Interpreter ausgeführt wird, und dass die Möglichkeit besteht, den auszuführenden Bytecode zu verifizieren. Auch direkte Speicherzugriffe sind nicht möglich, da die Speicherarithmetik fehlt. Zur Robustheit der Sprache trägt außerdem die implizite Überprüfung von Array-Grenzen bei. Ein weiterer Vorteil von Java für Chipkarten ist die Multiapplikationsfähigkeit. Dank dieser Fähigkeit beschränkt nur der Speicherplatz auf der Karte die Anzahl der installierten Programme. Neue Applikationen können auch nachträglich installiert und deinstalliert werden. So können neue Funktionalitäten jederzeit hinzugeladen oder ausgetauscht werden.

Leider können Java Card Applikationen, was Geschwindigkeit und Leistungsfähigkeit betrifft, nicht mit herkömmlichen Smart Card Anwendungen konkurrieren. Besonderes macht sich das bei kryptographischen Anwendungen bemerkbar.

2.5. Java Konzepte und Java für Chipkarten

Obwohl Java für eingebettete Systeme entwickelt wurde, sind die Ressourcenanforderungen bezüglich Prozessorleistung und Speicherkapazität momentan noch zu hoch für die Chipkarten. Daher war es notwendig den Befehls- und Funktionsumfang zu beschränken, um die Programmiersprache Java in Chipkarten einsetzen zu können. Stark verringert wurde der Sprachumfang von Java, um den minimalen Systemanforderungen gerecht zu werden. Somit ist Java für Chipkarten eine Untermenge von Java.

Zu den Konzepten von Java, die nicht in Java für Chipkarten unterstützt werden, gehören unter anderem: dynamisches Laden von Klassen, Security Manager, Threads, Klonen, Garbage Collection und Finalization. Auch mehrdimensionale Arrays und Typen wie char (16 Bit Unicode), double (64 Bit), float (32 Bit), long (64) und String werden nicht unterstützt.

Die Nichtunterstützung des Dynamischen Ladens von Klassen führt dazu, dass die Bedrohung von bösartigen Applets reduziert wird, da nur ein Applet eines Packages zu einem Zeitpunkt aktiv sein kann, und dieses Applet keine Klassen anderer Packages nutzen darf.

Es gibt also keinen Weg, während der Ausführungszeit einer Anwendung neue Klassen in die Karte zu laden. Dies erleichtert die Programmanalyse von Kartenanwendungen, da bei der Analyse das vollständige Programm bekannt ist.
Keine Unterstützung des Security Managers bedeutet, dass ein anderes Sicherheitsmodell (siehe Kapitel 2.10) als in Java verwendet wird. Dieses Modell ist direkt in der Virtuellen Maschine eingebaut.
In Anbetracht der verwendeten Hardware (siehe Kapitel 2.4) ist leicht ersichtlich, dass das Threadkonzept nicht angebracht wäre. Threads erzeugen in einem Betriebssystem einen höheren Verwaltungsaufwand, der sich in einer langsameren Ausführungsgeschwindigkeit bemerkbar macht. Das Fehlen von Threads erleichtert die Analyse von Java Card Anwendungen, da keine extrem vorsichtigen Annahmen über die Ausführungsreihenfolge von einzelnen Programmteilen berücksichtigt werden müssen.
Die Basisklasse „Objekt“ hat keine „clone()“ Methode und es gibt kein „clonable“ Interface.
Java für Chipkarten verfügt aus Performance- und Platzgründen über keine Garbage Collection. Diese ist dafür zuständig, dass nicht mehr referenzierte Daten zerstört und deren Speicher wieder freigegeben wird. Auch explizites Freigeben von Speicherplatz (Deallokation) ist nicht möglich. Sonst wäre die Zeigersicherheit von Java verletzt worden.
Zwar bietet Java für Chipkarten transiente Objekte an, aber bei diesen Objekten wird lediglich der Inhalt von Objekten transient. Dies bedeutet genauer gesagt, dass nach dem Reset der Karte oder dem Stromausfall nur der Inhalt auf Standardwerte zurückgesetzt wird. Man darf also nicht davon ausgehen, dass einmal angeforderter Speicher jemals wieder freigegeben wird, deshalb auch kein „finalize“.

Zu den Java-Konzepten, die in Java für Chipkarten unterstützt werden, gehören: Packages, dynamische Objekterzeugung, Vererbung, Interfaces wie in Java, Exceptions und einfache Typen, wie: boolean (true/false), byte (8 Bit), short (16 Bit), int (32 Bit).

Es existiert ein hierarchisches Bibliothek-System, das in jeder Hinsicht dem von Java entspricht. Sowohl Klassen, wie auch eindimensionale Arrays können mit „new“ erzeugt werden. Dennoch ist diese Möglichkeit wegen der nicht vorhandenen Garbage Collection mit Vorsicht zu genießen, um Speicherlecks zu vermeiden.
Das Schlüsselwort „super“ kann verwendet werden, ebenso virtuelle Methoden und der „instanceof“ Operator. Wie in Java fehlen einige Exceptions, die weggelassene Konzepte betreffen. Aber es gibt zusätzlich eine Exception, die innerhalb des Frameworks als Mechanismus zur Rückgabe von Ergebnissen an den Kartenleser verwendet wird. Vom Java für Chipkarten Standard wird die Implementierung des "int" Datentyps freigestellt, aber empfohlen. Eine Unterstützung von „nativen“ Typen ist zwar auf Bibliotheksebene notwendig und native Methoden in Applets sind zulässig, aber der Kartenimplementierer kann selbst entscheiden, ob und wie er es ausnutzt. Aus Sicht der Portabilität ist die Verwendung von nativen Methoden natürlich sehr ungünstig.

2.6. Komponenten der Java Card Technologie

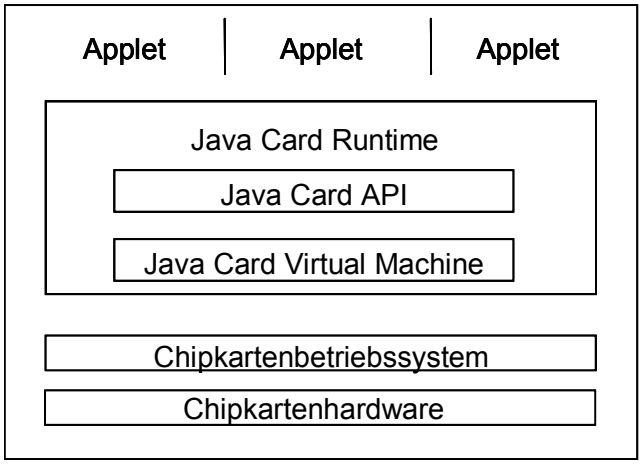


Abbildung 2.6.1 Komponenten der Java Card Technologie

Die Abbildung 2.6.1 zeigt die Komponenten der Java Card Technologie, die bei der Entwicklung von Java Cards von besonderer Bedeutung sind. Die Java Card Runtime Environment (JCRE), die das Laufzeitverhalten spezifiziert, ist ein Hauptbestandteil der gesamten Java Card Architektur. Die Java Card Runtime Environment ist in zwei Teile gegliedert. Erstens die Java Card Virtual Machine (JCVM), und zweitens das Java Card Application Programming Interface (JCAPI).

Mit dem aktuellen Stand der Hardware ist es nicht möglich eine vollständige Java Virtual Machine auf der Karte zu implementieren. Daher ist eine Zweiteilung der Architektur notwendig. In Abbildung 2.6.2 wird die Zweiteilung der Java Card Virtual Machine visualisiert.

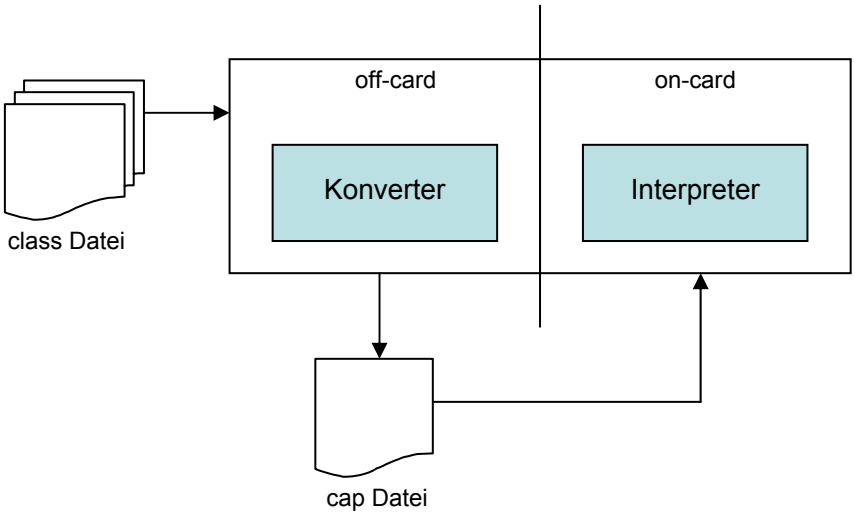


Abbildung 2.6.2 Java Card Virtual Machine

Auf der Karte befindet sich der erste Teil, der Java Card Interpreter (on-card VM), der den Java für Chipkarten Bytecode ausführt. Der Java Card Interpreter entspricht der Laufzeitunterstützung für das Java Sprachmodell.
Der zweite Teil ist der Konverter (off-card VM), der außerhalb der Karte, z.B. auf einem PC, läuft. Der Konverter lädt die class Dateien, die zu einem Paket gehören, und verarbeitet diese zu einer cap Datei (Converted Applet). Eine cap Datei enthält die Binärdateien eines kompletten Java Card Packages. Alle Informationen, die für dynamisches Linken notwendig sind, werden in einer exp Datei (Java Card Export File) ausgelagert. Nur die cap Datei wird anschließend auf die Karte geladen und vom Java Card Interpreter ausgeführt. Die Lebensdauer einer Java Card Virtual Machine ist mit der Lebensdauer ihrer Java Card identisch. Bei der Versorgungsspannungsunterbrechung hält die Java Card Virtual Machine an, wird aber bei dem nächsten Reset ihre Tätigkeit fortsetzen.

Die Java Card API besteht aus einer Menge von Klassen für die Programmierung von Smart Card Applikationen. Sie beinhaltet drei Kernklassen (java.lang, javacard.framework und javacard.security) und eine Erweiterungsklasse (javacardx.crypto). Die Java Card API ist eine Untermenge der normalen Java API's.

Die JCRE ist für das Ressourcenmanagement, die Netzwerkkommunikation, die Ausführung von Applets und die Applet-Sicherheit verantwortlich.

Ein besonderes Merkmal bei der Laufzeitumgebung einer Java Card ist die Lebensdauer. Die Java Card Runtime Environment ist persistent auf der Karte abgelegt. Die Laufzeitumgebung wird bei der Erstellung der Karte initialisiert und bleibt erhalten bis die Karte zerstört wird. Die Initialisierung erfolgt also nur einmal. Die Laufzeitumgebung wird bei Unterbrechung von Stromversorgung nicht zerstört, sondern suspendiert. Mit Hilfe der Java Card Technologie können die in der Smart Card laufenden Programme in der Java-Programmiersprache geschrieben werden.

2.7. Entwicklung eines Java Card Applets

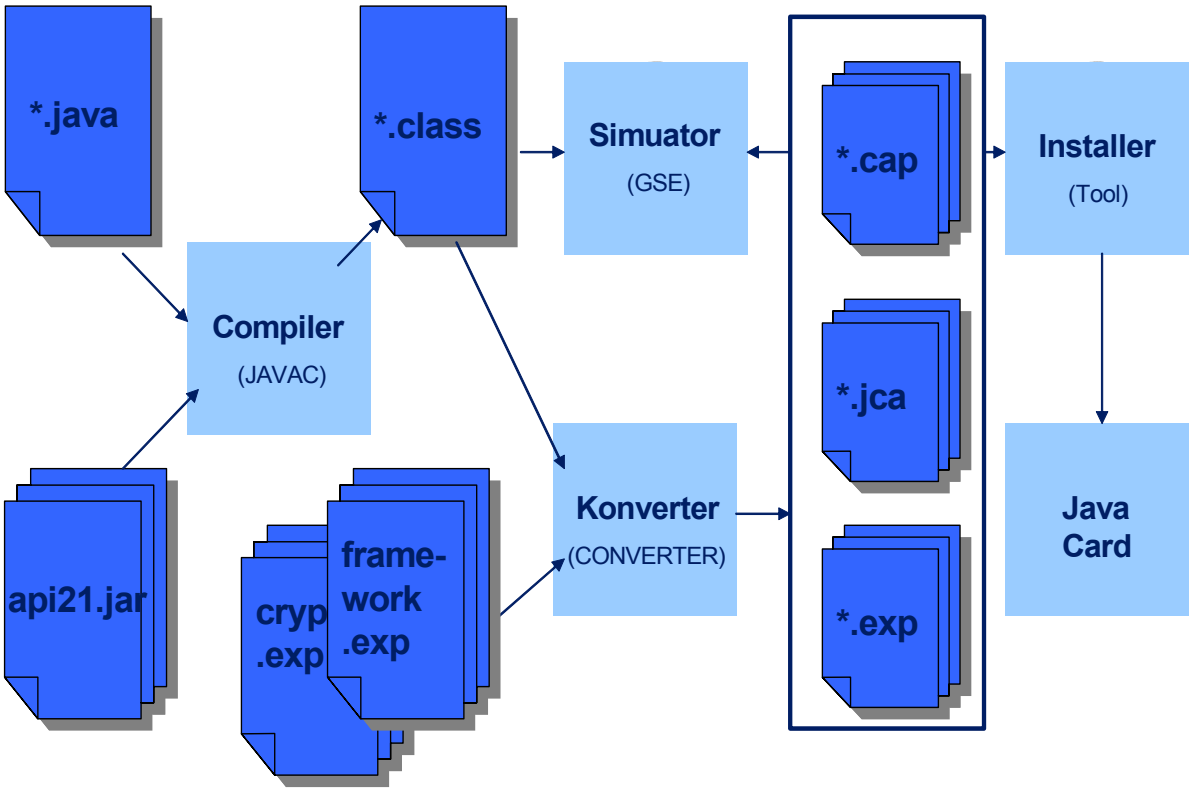


Abbildung 2.7.1 Entwicklung eines Java Card Applets

Die Abbildung 2.7.1 beschreibt die Entwicklung eines Java Card Applets. Die Ausgangsdatei ist eine Java-Datei. Die Java-Datei wird normalerweise in einem Editor geschrieben, sie kann aber auch als vorgefertigtes Java Card Applet Skeleton entstanden sein. Diese Datei wird in Verbindung mit einer Jar-Datei, die die Klassen des Java Card Framework zusammenfasst, in einem Compiler in eine Class-Datei umgewandelt. Die Class-Datei besteht aus Java Bytecode. Die Class-Datei zusammen mit Exp-Dateien (Java Card Bibliotheken wie javacard.framework, javacard.security, javacardx.crypto, die auch aus Java Bytecode bestehen) wird mit dem Konverter konvertiert. Aus dieser Konvertierung entstehen drei Dateien: *.cap (Converted Applet), *.jca (Java Card Assembler) und *.exp (Java Card Export File). Die Cap-Datei enthält die Daten eines ganzen Packages in kompakter Form. Das Format kann auf eine Karte mittels einer Entwicklungsumgebung geladen werden oder in einem Simulator getestet werden. Die Jca-Datei ist eine für den Menschen lesbare ASCII-Datei und repräsentiert den Inhalt einer Cap-Datei. Die Exp-Datei enthält die Abbildung von symbolischen Referenzen auf Tokens, die zum Binden auf der Karte benutzt werden können. Diese Datei ist ein Teil des zu publizierenden Interfaces eines Packages, falls dieses Package Klassen oder Methoden exportiert.

2.8. Kommunikation zwischen Terminal und Java Card

Um Informationen zwischen einer Java Card und einem Terminal auszutauschen, muss eine Kommunikation zwischen den beiden stattfinden. Die Informationen werden in digitaler Form über eine serielle Leitung (Halbduplex) ausgetauscht. Deswegen kann nur ein Kommunikationspartner senden, der andere muss in der Zeit empfangen. Ein Vollduplex-Verfahren ist derzeit in der Chipkartenwelt noch nicht möglich. Die Kommunikation wird immer vom Terminal angestoßen. Solches Verhalten nennt man auch Master-Slave-Verhalten. Dabei ist das Terminal der Master und die Java Card der Slave.
Zuerst wird eine Java Card in ein Terminal gesteckt, dabei werden die Kontakte der Java Card mit denen des Terminals verbunden. Danach werden die fünf Kontakte der Java Card in der richtigen Reihenfolge elektrisch aktiviert (siehe dazu Kapitel 2.3). Daraufhin führt die Java Card einen Power-On-Reset automatisch aus und sendet einen Answer To Reset (ATR) zum Terminal. Der ATR enthält verschiedene Übertragungs- und Kartenparameter. Das Terminal wertet den ATR aus. Ein Applet wird über das Install Kommando auf die Karte geladen. Der erfolgreich abgeschlossene Ladevorgang bewirkt, dass ein Applet auf die Karte geladen wird. Um ein bestimmtes Applet zu aktivieren, schickt das Terminal ein Select Kommando ab. Das Select Kommando wird dann von Java Card bearbeitet und ein Applet wird ausgewählt. Java Card erzeugt eine Antwort und schickt diese an das Terminal zurück. Danach folgen die anderen benutzerdefinierten Kommandos, z.B. das Debit Kommando oder das Credit Kommando. Die werden über die Process Methode aufgerufen. Diese Methoden werden einzeln über die switch-case Anweisung ausgewählt und abgearbeitet. Nach der Beendigung der einzelnen Methoden wird eine Antwort an das Terminal gesendet.
Die anschauliche Beschreibung der Kommunikation zwischen dem Terminal und der Java Card wird in Abbildung 2.8.1 dargestellt.

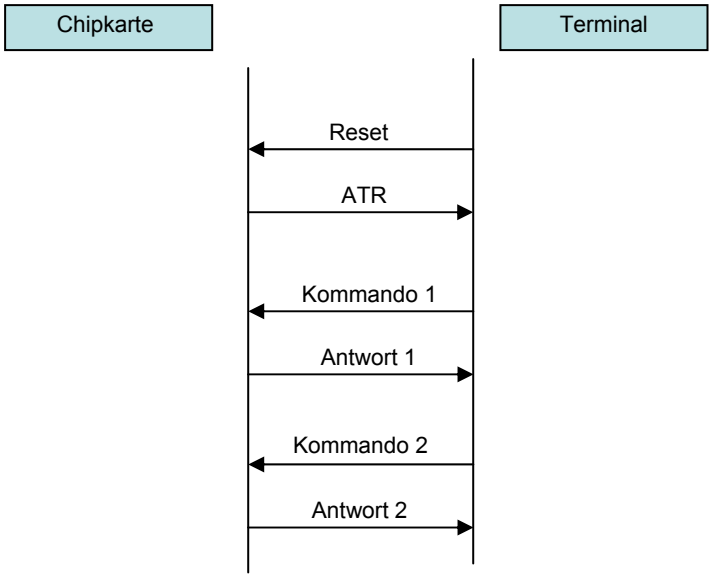


Abbildung 2.8.1 Kommunikation zwischen Chipkarte und Terminal

Übertragungsprotokolle

Es gibt verschiedene Möglichkeiten die Kommunikation aufzubauen, so dass auch im Falle eines Übertragungsfehlers eine erneute Kommunikation möglich ist. Die Übertragungsprotokolle beschreiben das Verhalten bei Übertragungsstörungen, aber auch die genauere Realisierung der Kommandos und der Antworten. Es gibt 15 Übertragungsprotokolle, deren Bezeichnung sich aus dem Ausdruck „T=“ (transport protocol) und Nummern zwischen 0 und 15 zusammensetzt. Die Protokolle T=0 und T=1 haben sich international durchgesetzt. Das Protokoll T=0 ist ein asynchrones Halbduplex-Protokoll, das byteorientiert Daten überträgt, was genau bedeutet, dass die kleinste durch das Protokoll bearbeitete Einheit, ein Byte ist. Das T=1 Protokoll ist ebenso ein asynchrones Halbduplex-Protokoll, das aber blockorientiert die Daten überträgt. Die beiden Protokolle sind in [ISO/IEC 7816-3] spezifiziert. Die Dateneinheiten, die durch die Übertragungsprotokolle transportiert werden, bezeichnet man als TPDU (Transmission Protocol Data Unit). Die TPDUs sind protokollabhängige Container, die Daten von und zur Chipkarte transportieren. Das Protokoll T=0 wird am häufigsten verwendet, da es auf minimalen Speicherbedarf und maximale Einfachheit ausgelegt wurde. Weil das T=0 byteorientiert ist, muss nach einem erkannten Übertragungsfehler nur das nicht korrekt empfangene Byte nochmals angefordert werden, was bei T=1 den gesamten Block bedeutet. Die Erkennung von Übertragungsfehlern erfolgte hier mit Hilfe des Paritätsbits. Das Paritätsbit wird jedem übertragenen Byte nachgestellt.

Die gesamte Datenübertragung von und zur Chipkarte kann im Rahmen des OSI-Schichtenmodells dargestellt werden [RE02]. Dabei legen mehrere internationale Normen die Abläufe zwischen den Schichten fest. In Abbildung 2.8.2 werden die OSI -Schichten und die wichtigsten Normen dargestellt.

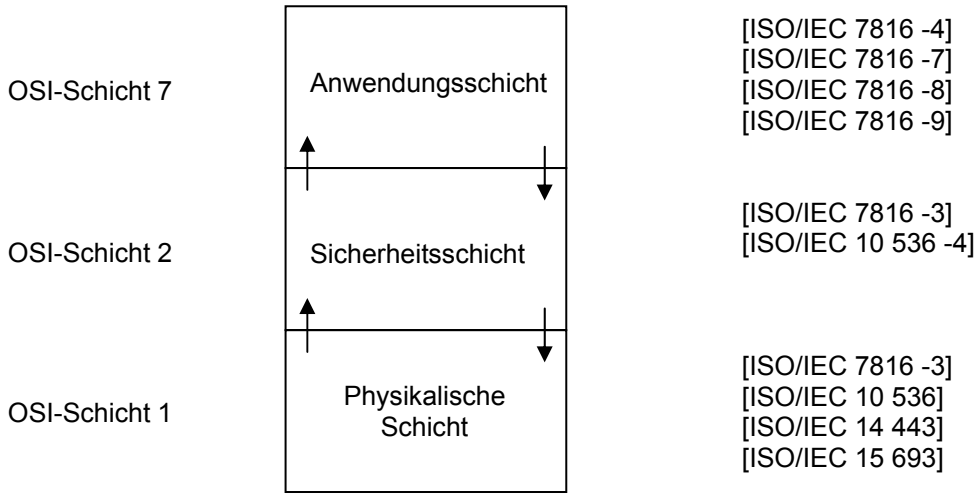


Abbildung 2.8.2 Datenübertragung im OSI-Schichtenmodell

Nachrichten-APDUs

Bei dem gesamten Datenaustausch zwischen einem Terminal und einer Java Card werden Nachrichten-APDUs (Application Protocol Data Unit) verwendet. Nach dem OSI-Schichtenmodell ist dies Schicht 7. Diese ist direkt oberhalb der Sicherheitsschicht angesiedelt. In der direkt darunter liegenden Schicht sind die protokollabhängigen TPDUs. Es werden zwei Arten von Nachrichten-APDUs unterschieden: die Kommando-APDUs (command-APDU) und Antwort-APDUs (response-APDU). Die ersten sind Anfragen an die Java Card und die zweiten sind Antworten der Java Card. Allgemein sind die APDUs Container, die eine vollständige Anfrage bzw. Antwort enthalten. Nach [ISO/IEC 7816-4] sind die APDUs so aufgebaut, dass sie unabhängig vom Übertragungsprotokoll sind. Somit sind der Aufbau und der Inhalt jeder APDU unabhängig von verschiedenen Übertragungsprotokollen. Ein Kommando-APDU besteht, wie man in Abbildung 2.8.3 sieht, aus einem Header und einem Body.

	CLA	INS	P 1	P 2	Lc-Feld	Daten-Feld	Le-Feld
Länge in Byte:	1	1	1	1	0 oder 1	Lc	0 oder 1

Header Body

Abbildung 2.8.3 Vollständiger Aufbau eines Kommando-APDU

Der Header setzt sich aus vier Elementen zusammen: CLA (Application Class), INS (Instruction) und P1, P2 (Parameter 1 / Parameter 2 of the command header). Das CLA wird benutzt um Anwendungen und ihren spezifischen Befehlssatz zu kennzeichnen. Das INS spezifiziert, welche Instruktion ausgeführt werden soll. Hier wird die entsprechende Methode (d.h. das entsprechende Kommando) gewählt, die die Instruktion ausführt, und die Antworten für das Terminal gespeichert. Die P1 und P2 werden verwendet, um das Kommando besser zu beschreiben. Der Body setzt sich aus dem Lc-Feld (Data Length Command), dem Daten-Feld und dem Le-Feld (Data Length Expected) zusammen. Das Lc-Feld spezifiziert die Länge des Datenfelds und das Le-Feld legt die Länge des von der Karte zurückzusendenden Datenfelds fest. Falls das Le-Feld den Wert „00“ hat, erwartet das Terminal von der Karte das Maximum der für dieses Kommando zur Verfügung stehenden Daten.
In Abbildung 2.8.4 sind vier mögliche Fälle (cases), die sich aus der Kombination der Teile der Kommando-APDU zusammensetzen, zu sehen. In Abbildung 2.8.5 sind die dazugehörigen Antwort-APDUs zu sehen. Der erste Fall besteht nur aus dem Header also aus CLA, INS, P1 und P2. Die Antwort auf dieses Kommando besteht immer nur aus SW1 und SW2 (Status Word 1 / Status Word 2), da keine Daten zurückgesendet werden. Die Antwort informiert, ob die Daten angekommen sind oder nicht.

Die verschiedenen Antwortmöglichkeiten befinden sich in der Tabelle 2.8.1. Der zweite Fall besteht aus dem Header und dem Le-Feld. In der Antwort werden Daten erwartet. Die Länge der Daten steht im gesendeten Le-Feld. Auf ein so aufgebautes Kommando kommen eine Antwort und die erwarteten Daten zurück. Der dritte Fall besteht aus dem Header, dem Lc-Feld und dem Daten-Feld. Es werden die Daten im Daten-Feld gesendet. Die Daten haben die im Lc-Feld angegebene Länge. Zurück kommt nur eine Antwort SW1/SW2, d.h. ob die Daten angekommen sind oder nicht. Im vierten Fall werden der Header und der Body gesendet. Es werden die Daten der Länge, die im Lc-Feld steht, gesendet und als Antwort erwartet man aber Daten der Länge, die im Le-Feld steht. Zurück kommen eine Antwort und die erwarteten Daten mit einer Länge, die im Le-Feld stand.

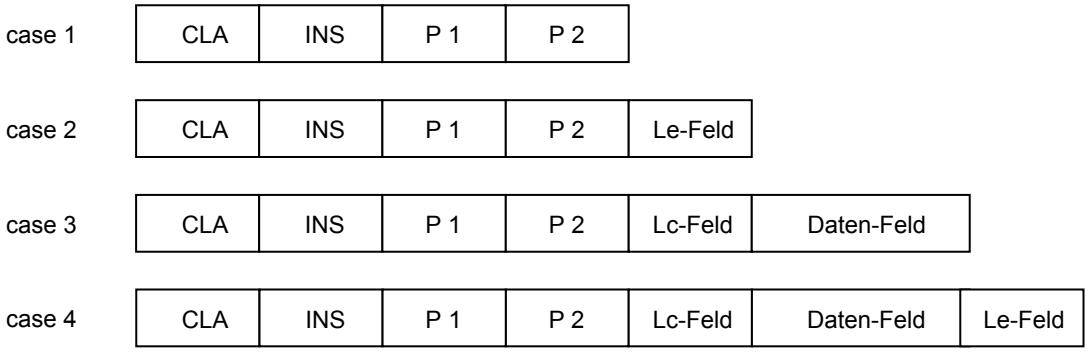


Abbildung 2.8.4 Vier mögliche Kommando-APDUs

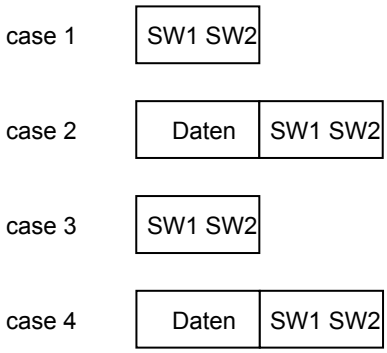


Abbildung 2.8.5 Die dazugehörigen Antwort-APDUs

Eine Antwort-APDU besteht aus einem optionalen Body und einem obligatorischen Trailer. Das zeigt auch die Abbildung 2.8.6.

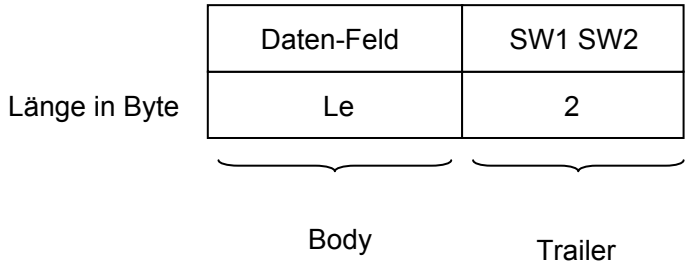


Abbildung 2.8.6 Aufbau einer Antwort-APDU

Der Body besteht aus einem Datenfeld. Dessen Länge wurde in der Kommando-APDU im Le-Feld festgelegt. Der Trailer beinhaltet zwei Bytes SW1 und SW2. SW1 und SW2 werden auch als Returncode bezeichnet und beinhalten die eigentliche Antwort auf das Kommando. Der Returncode „90 00“ bedeutet, dass das Kommando erfolgreich abgearbeitet wurde. Andere Antworten bedeuten, dass etwas nicht in Ordnung ist.

Die Tabelle 2.8.1 [RE02] stellt eine Liste der wichtigsten und am häufigsten vorkommenden Antworten (Returncode) dar.
Die Spalte Status bezeichnet:

- CE (process aborted, checking error) Prozess abgebrochen, Prüfungsfehler
- EE (process aborted, execution error) Prozess abgebrochen, Ausführungsfehler
- NP (process completed, normal processing) Prozess ausgeführt, normale Bearbeitung
- WP (process completed, warming processing) Prozess ausgeführt, Warnung

Returncode	Status	Bedeutung
61xx	NP	Kommando erfolgreich ausgeführt, xx Byte Daten sind als Antwort vorhanden und können mit GET RESPONSE angefordert werden
6281	WP	Die zurückgegebene Daten können unter Umständen fehlerhaft sein
6282	WP	Es konnten weniger als Le Bytes gelesen werden, da das Dateiende vorher erreicht wurde
6283	WP	Die selektierte Datei ist reversibel gesperrt (invalidated)
6284	WP	Die File Control Information (FCI) ist nicht nach [ISO/IEC 7816-4] strukturiert

62xx	WP	Warnung: Zustand des nichtflüchtigen Speichers unverändert
63Cx	WP	Zähler hat den Wert x erreicht (0 ≤ x ≤ 15), die genaue Bedeutung ist vom jeweiligen Kommando abhängig
63xx	WP	Warnung: Zustand des nichtflüchtigen Speichers verändert
64xx	EE	Ausführungsfehler: Zustand des nichtflüchtigen Speichers unverändert
6581	EE	Speicherfehler , z.B. bei Schreiboperationen
65xx	EE	Ausführungsfehler: Zustand des nichtflüchtigen Speichers verändert
6700	CE	Länge falsch
67xx bis 6Fxx	CE	Prüfungsfehler
6800	CE	Funktionen im Class Byte werden nicht unterstützt
6881	CE	Logische Kanäle werden nicht unterstützt
6882	CE	Secure Messaging wird nicht unterstützt
6900	CE	Kommando nicht erlaubt
6981	CE	Kommando inkompatibel zur Dateistruktur
6982	CE	Sicherheitszustand nicht erfüllt
6983	CE	Authentisierungsmethode gesperrt
6984	CE	Referenzierte Daten sind reversibel gesperrt (invalidated)
6985	CE	Benutzungsbedingungen nicht erfüllt
6986	CE	Kommando nicht erlaubt (kein EF selektiert)
6897	CE	Erwartete Secure Messaging-Datenobjekte fehlen
6988	CE	Secure Messaging Datenobjekte inkorrekt
6A00	CE	Falsche Parameter P1/P2
6A80	CE	Parameter im Datenteil sind falsch
6A81	CE	Funktion wird nicht unterstützt
6A82	CE	Datei wurde nicht gefunden
6A83	CE	Record wurde nicht gefunden
6A84	CE	Ungenügend Speicherplatz in der Datei
6A85	CE	Lc inkonsistent mit TLV Struktur
6A86	CE	Inkorrekte Parameter P1/P2
6A87	CE	Lc inkonsistent mit P1/P2
6A88	CE	Referenzierte Daten nicht gefunden
6B00	CE	Parameter 1 oder 2 falsch
6Cxx	CE	Falsche Länge Le, xx gibt die korrekte Länge an

6D00	CE	Kommando (Instruktion) wird nicht unterstützt
6E00	CE	Class wird nicht unterstützt
6F00	CE	Kommando abgebrochen, genaue Diagnose nicht möglich (z.B. Fehler im Betriebssystem)
9000	NP	Kommando erfolgreich ausgeführt
920x	NP	Schreiben ins EEPROM nach x-maligen Versuch erfolgreich
9210	CE	Ungenügend Speicherplatz
9240	EE	Schreiben ins EEPROM nicht erfolgreich
9400	CE	Kein EF selektiert
9402	CE	Adressbereich überschritten
9404	CE	FID nicht gefunden, Record nicht gefunden, Vergleichsmuster nicht gefunden
9408	CE	Selektierter Dateityp unpassend zum Kommando
9802	CE	Keine PIN definiert
9804	CE	Zugriffsbedingungen nicht erfüllt, Authentifizierung fehlgeschlagen
9835	CE	ASK RANDOM/GIVE RANDOM nicht ausgeführt
9840	CE	PIN-Prüfung nicht erfolgreich
9850	CE	INCREASE/DECREASE kann nicht ausgeführt werden, da Grenzwert erreicht
9Fxx	NP	Kommando erfolgreich ausgeführt, xx Byte Daten sind als Antwort vorhanden und können mit GET RESPONSE angefordert werden

Tabelle 2.8.1 Die häufigsten Antwort-APDUs [RE02]

Skript

Um Java Cards zu testen, reicht es nicht aus, nur ein Applet zu schreiben. Ein anderes wichtiges Dokument ist die Skript Datei (Scr-Datei). Ein Skript besteht aus einer Reihe von Kommando-APDUs (Befehlen), die mit einer Karte ausgetauscht werden können. Diese Skript Datei muss der Entwickler selber in einem Editor schreiben. Es ist dabei zu berücksichtigen, dass die im Skript benutzten Teile des Kommando-APDUs vorher im Java-File definiert wurden. Alle APDU Kommandos werden sowohl im Quellcode als auch im Skript als hexadezimale Zahlen dargestellt und so weiterbenutzt.

Anhand eines Beispiels aus der Aspects Developer Entwicklungsumgebung (siehe unten), können weitere Skripts erstellt werden.

Beispiel:

```
powerup;

//:Select Applet Eeprom_Test1
0x00 0xA4 0x04 0x00 0x06 0x04 0x03 0x04 0x00 0x01 0xEE 0x00;
//:ReadGroesseEeprom
0x90 0x01 0x00 0x00 0x00 0x02;
//:Eeprom_Eeprom
0x90 0x02 0x00 0x00 0x00 0x02;
//:Ram_Eeprom
0x90 0x03 0x00 0x00 0x00 0x02;
//:Dummy_Null
0x90 0x04 0x00 0x00 0x00 0x02;

powerdown;
```

Powerup ist ein Startbefehl und powerdown ein Endbefehl. Das erste Kommando-APDU ist ein Select. Hier wird das Applet „Eeprom_Test1“ mit AID (Applikation Identifier) 04 03 04 00 01 EE ausgewählt und auf eine Java Card geladen. Erst nachdem dieser Befehl erfolgreich abgeschlossen ist, können die anderen Kommandos abgearbeitet werden. Die CLA von diesem Befehl ist „0x00“, die INS ist „0xA4“, P1 ist „0x04“und P2 ist „0x00“. Diese vier Bytes sehen bei jedem Select-Befehl gleich aus. Danach kommt das Lc-Feld mit dem Wert „0x06“. Dieser Wert beschreibt die Länge der Applet AID. Im Daten-Feld wird der ganze Applet AID geschrieben, also hier: „0x04 0x03 0x04 0x00 0x01 0xEE “. Am Ende kommt das Le-Feld mit dem Wert „0x00“. Somit ist das Applet auf der Karte geladen und selektiert.
Das zweite Kommando-APDU ruft die Methode „ReadGroesseEeprom“ aus dem Applet „Eeprom_Test“ auf. Hier hat die CLA den Wert „0x90“, die INS den Wert „0x01“, P1 und P2 haben jeweils den Wert „0x00“, das Lc-Feld existiert in dem Fall nicht, das Datenfeld hat den Wert „0x00“ und das Le-Feld hat den Wert „0x02“. Die erwartete Antwort wird zwei Bytes groß sein. Nächste Kommando-APDUs werden wie die vorherige Kommando-APDU aufgebaut, nur das INS ändert sich und nimmt den Wert „0x02“, „0x03“ oder „0x04“ an.

2.9. Aufbau eines Java Card Applets und die Java Card APIs

Der Aufbau eines Java Card Applets ist in der Literatur [C00], [GJ98], [HNSSS02] bzw. in den Spezifikationen der Firma SUN [Sun3] ausreichend verfügbar.
In diesem Abschnitt werden alle Elemente eines Java Card Applets eingeführt, und anhand von kurzen Java für Chipkarten Codefragmenten vorgestellt.

Package Deklaration

In den ersten Zeilen eines Java für Chipkarten Code wird ein Package deklariert. Zu einem Package gehören ein Applet und andere Klassen, falls die benötigt und vorhanden sind (z.B. beim Shareable Interface werden zwei Packages mit jeweils einem Applet benötigt, ein Package für den Client und eins für den Server. Zusätzlich werden eine oder mehrere Klassen benötigt, z.B. für Fehlermeldungen).

Beispiel:
package *packagename;*

Schnittstellen importieren (Java Card APIs)

Um die Programmierung von Chipkarten in Java zu ermöglichen, gibt es vier Pakete. Die standardisierten Schnittstellen (API) stellen nützliche Funktionen zur Verfügung.
Das obligatorische Paket java.lang ist die Basis für Java auf Chipkarten. Die unterstützten Klassen des Pakets sind: Objekt, Throwable und Exceptions.
Das ebenfalls obligatorische Paket javacard.framework, das die Kernfunktionalität eines Java Card Applet definiert, ist eine Ergänzung zu java.lang. Die Kernfunktionen werden unter anderem durch die elementaren Klassen für die Appletverwaltung, durch den Datenaustausch mit dem Terminal und den verschiedenen Konstanten im Rahmen von [ISO/IEC 7816-4] beschrieben.
Das Paket javacard.security stellt Schnittstellen für eine Reihe von Kryptoalgorithmen zur Verfügung und dient der Sicherheit der Java Card. Es werden die symmetrische Verschlüsselung DES (Data Encryption Standard), die asymmetrische Verschlüsselung RSA (Rivest, Shamir and Adlemann Asymmetric Algorithm) und Signaturen unterstützt. Das Paket ist aber nicht als allgemeines Ver- und Entschlüsselungsinstrument nutzbar. Das Paket ist obligatorisch.
Das optionale Paket javacardx.crypto enthält die Schnittstellen zu den dazugehörigen Entschlüsselungsmethoden.

Beispiel:
import javacard.framework.*;
import javacard.secutity.*;
import javacardx.crypto.*;

Appletklasse

Jedes Applet ist von der Klasse *javacard.framework.Applet* abgeleitet. Der Appletname ist der Name von einem Applet und wird vom Entwickler vergeben.

Beispiel:
public class *Appletname* ***extends Applet {…}***

Konstantendeklaration

Konstanten werden üblicherweise im Hauptteil des Quellcodes definiert, weil es die Lesbarkeit enorm erleichtert. Die Konstanten können aber auch bei den jeweiligen Methoden definiert werden.
Die Konstantendeklaration wird benötigt um die Kommunikation zwischen der Karte und dem Terminal herzustellen. Die Kommunikation erfolgt auf Basis der ausgetauschten APDUs (siehe Kapitel 2.6). Für einen lesbaren Code hat es sich als unabdingbar herausgestellt, dass eine Definition der APDU Bytes im Konstantenblock erfolgt. Die eingeführten Bezeichnungen werden ebenfalls im Skript (siehe Kapitel 2.8) verwendet, welches während der Simulation oder nach dem Download des Codes auf die Karte den Datenstrom zwischen Karte und Terminal/Simulator definiert. Die Konstantendefinition erfolgte mit mehreren Variationen immer nach folgendem, bewährtem Schema.
Die erste Konstante mit dem Namen Appletname_CLA identifiziert die Kommandostruktur. Die benötigt man, um dem aktuellen Applet die dazugehörigen Methoden zuzuordnen. Dieser Konstanten ist ein fester Wert 0x90 zugewiesen.
Die danach folgenden Konstanten mit der Endung _INS spezifizieren Applikationsinstruktionen und bezeichnen benutzerdefinierte Methoden. Die Namen der Appletmethoden werden vom Entwickler vergeben (z.B. credit, debit usw.). Den Konstanten mit der Endung _INS ist ein beliebiger Wert zugewiesen. Im Laufe dieser Arbeit haben sich für die Konstanten _INS die Werte 0x01, 0x02 usw. etabliert.

Beispiel:
final static *Appletname****_CLA = (byte)*** *0x90;*
final static *Appletmethode1****_INS = (byte)*** *0x01;*
final static *Appletmethode2****_INS = (byte)*** *0x02;*

Deklaration von Instanzvariablen

Instanzvariablen können nur von den einfachen Typen byte, short oder boolean sein.

Beispiel:
short *variablename1;*
byte [ ] *variablename2;*

Spezifikation des Konstruktors

Hier sollten alle jemals im Applet benötigten Objekte erzeugt (um die Kontrolle des Speicherverbrauchs zu vereinfachen) und das Applet bei der JCRE (siehe auch Kapitel 2.6) registriert werden. Applets, die mit der Methode *register( )* registriert werden, existieren während der gesamten Lebensdauer der Karte. Die Parameter bArray, bOffset und bLength werden, wie im Beispiel zu sehen, mit vom System vorgegebenen Standardwerten initialisiert.

Beispiel:
private** Appletname **(byte [ ] bArray, short bOffset, byte bLength)
{…
register(); }

Die Parameter:
bArray – das Array, das die Installationsparameter beinhaltet
bOffset – das Anfangsoffset beim bArray
bLength – die Länge der Parameterdaten im bArray, der maximale Wert von bLength ist 32

Wichtige Appletmethoden

Das JCRE interagiert mit dem Applet über die vier Methoden install, select, deselect und process.

1) install
Diese Methode wird nur einmal bei der Installation des Applet aufgerufen und erhält als Parameter die Daten der Install-APDU. Die Aufgabe von install ist es, ein Objekt zu erzeugen (ähnlich zur main()- Methode bei Java Applikationen), indem dessen Konstruktor aufgerufen wird, und es zu registrieren.

Beispiel:
public static void install(byte [ ] bArray, short bOffset, byte bLength) {…}

2) select
Ausgewählt wird ein Applet dadurch, dass das JCRE ein Select-APDU erhält, welches den AID (siehe auch Kapitel 2.8) des gewünschten Applets enthält. Nachdem ein Applet ausgewählt wurde, werden sämtliche APDUs direkt der process-Methode dieses Applets gesendet. Der Rückgabewert entscheidet darüber, ob die Selektion als erfolgreich an den Kartenleser gemeldet wird.

Beispiel:
```
public boolean select()
{
return true;
}
```

3) deselect
Wird von dem JCRE bei der Applet-Deselektion aufgerufen, um das aktuelle Applet abzuwählen, z.B. wenn ein anderes Applet selektiert wird. Sie übernimmt alle Aufräumarbeiten (wie Löschen des Applet und des Packages), welche für das Ausführen von anderen Applets nötig sind. *deselect* wird nicht aufgerufen, wenn die Karte aus dem Lesegerät gezogen wird.

Beispiel:
public void deselect() { }

4) process
Nach der Selektierung wird bei allen weiteren Anfragen des Lesegerätes diese Methode aufgerufen. Die Aufgabe von process besteht darin, alle ankommenden APDUs zu interpretieren und die entsprechenden Aktionen auszuführen.

Beispiel:

```
public void process (APDU apdu)  throws ISOException
{
   byte buffer[] = apdu.getBuffer();

//überprüfe ob Selektierung abgeschlossen
   if (selectingApplet())
                return;

//wenn Appletbezeichnung nicht korrekt, dann Ausnahmebehandlung
   if (buffer [ISO7816.OFFSET_CLA] != APPLETNAME_CLA)
            ISOException.throwIt (ISO7816.SW_CLA_NOT_SUPPORTED);

//ausgewählte Methoden werden aufgerufen und ausgeführt
     switch (buffer [ISO7816.OFFSET_INS])
     {
       case  APPLETMETHODE1_INS   : Appletmethode1(apdu);      return;
       case  APPLETMETHODE2_INS   : Appletmethode2(apdu);      return;

       default: ISOException.throwIt (ISO7816.SW_INS_NOT_SUPPORTED);
     }
}
```

Benutzerdefinierte Methoden

Hier werden die benutzerdefinierten Methoden, wie debit, credit usw. abgearbeitet.

Beispiel:

```
public void Appletmethode1 (APDU apdu) {…}
```

2.10. Java Card Sicherheit

Sicherheit ist ein wichtiges Thema bei Java Card, weil viele darauf gespeicherte Informationen geheim bleiben sollen. Sie müssen vor unbefugten Zugriffen geschützt werden. Sicherheitsvorkehrungen werden daher schon bei Chipkarten auf vier Ebenen realisiert. Mehr über die Sicherheit bei Chipkarten und über Arten von Angriffen findet man in [G02], [HL02], [HLL03], [K98], [KJJ98/I] und [RE02].

Kartenkörper

Auf der Karte werden gleich mehrere Sicherheitsmerkmale eingearbeitet. Das Foto vom Karteninhaber, die Lasergravur, die Hochprägung von Schriftzeichen und Unterschriftstreifen identifizieren eindeutig den Karteninhaber. Auf dem Magnetstreifen werden zusätzliche Informationen abgespeichert. Manche Karten enthalten noch ein Hologramm, die Mikroschrift, das Multiple Laser Image und/oder Guillochen, die nur schwer kopierbar sind. Die bei den Chipkarten benutzten Hologramme heißen auch Prägehologramme, weil sie auch bei diffusem, weißem Tageslicht in Draufsicht erkannt werden müssen. Die Mikroschrift basiert auf feinen, gedruckten Strukturen. Für das Auge ist dies nur als Linie erkennbar, unter der Lupe wird aber die Schrift erkannt. Das Multiple Laser Image ähnelt einem Hologramm, es ist eine Art Kippbild. Der große Unterschied zum Hologramm ist aber, dass beim Multiple Laser Image kartenindividuelle Informationen auf dem kleinem Bild dargestellt werden. Die Guillochen sind meist ovale oder runde Linienfelder, die geschlossen und miteinander verbunden sind.

Halbleitertechnologie/Hardware

Die Sicherheit auf Hardwareebene wird in zwei Blöcke unterteilt: passive und aktive Sicherheitsmassnahmen. Zu den ersten gehört beispielsweise eine scheinbar chaotische Anordnung der Leitungen im Chip. Zusätzliche Metallisierungen verhindern das Auslesen von Daten. Das Metallisieren bedeutet, dass eine Schutzschicht über die gesamte oder über gefährdete Bereiche gelegt wird. Diese Schicht verhindert die Analyse von elektrischen Potenzialen auf dem Chip während des Betriebes. Auf der Karte befindet sich ein einziger Chip, und alle Busse laufen intern. Zu den aktiven Sicherheitsmassnahmen gehört unter anderem die Spannungsüberwachung, die Analysen aufgrund induzierter Fehler verhindert. Die Chipkarte hat eine Passivierungsschicht, die Beschädigungen erkennen lässt und so vor Manipulation schützt. Außerdem wird die Untertaktung verhindert, weil es kein Einzelschrittbetrieb gibt. Die Chipkarte verfügt zusätzlich über die Power-On-Erkennung.

Software

Das Betriebssystem muss auf die Hardware abgestimmt sein. Es ist sinnvoll, dass Teile des Betriebssystems im EEPROM gespeichert werden. Somit kennt der Chiphersteller nicht alle Informationen über das Betriebssystem. Das Betriebssystem soll für konstanten Stromverbrauch sorgen, und das Übertragungsprotokoll soll falsche Eingaben abfangen. Die gesamte Kommunikation mit der Karte wird durch das Betriebssystem überwacht. Als Letztes sollte die Karte am Ende ihrer Lebenszeit komplett deaktiviert sein.

Anwendung

Anwendungen, die in Java geschrieben sind, profitieren sowohl von Java als auch von Java für Chipkarten.

(1) Java

Durch das Sicherheitskonzept von Java wird die Sicherheit der Kartenapplikation zusätzlich erhöht.

- Wie beim normalen Java können verschiedene Zugriffsrechte (public, private, protected) vergeben werden. Java trennt die Objekte der verschiedenen Anwendungen und erlaubt keine Zugriffe auf private Bereiche anderer Objekte.

(2) Java für Chipkarten

- An die Stelle des Sicherheitsmanagers, der für das „Sandboxing“ sorgt, tritt das Prinzip der Applet-Isolation. In Java für Chipkarten ist jedes Objekt genau einem Objekt der Klasse Applet zugeordnet. Ein Applet kann nicht auf Objekte eines anderen Applets zugreifen, es sei denn, dies wird ausdrücklich erlaubt (Object-Sharing). In diesem Zusammenhang spricht man von Firewalls zwischen einzelnen Applets.
- Da kein dynamisches Laden von Klassen erlaubt ist, kann der Anwender sicher gehen, dass, während er die Karte benutzt, keine andere Software auf die Karte gelangen kann. Es ist auch nicht möglich, dass mehrere Programme gleichzeitig auf der Karte laufen können (nur ein Applet kann aktiv sein).
- Auch das Fehlen von Threads sorgt für mehr Sicherheit (siehe Kapitel 2.9)
- Java für Chipkarten hat sich für das Atomicity Prinzip bei Transaktionen entschieden, um den Datenverlust bei Stromunterbrechungen zu minimieren. Wird die Abarbeitung des Programms aus irgendwelchen Gründen unterbrochen, so werden bereits durchgeführte Update-Operationen automatisch rückgängig gemacht. Die Daten bleiben konsistent.
- Die Java Card Klassenbibliothek enthält Klassen zur Implementierung von PIN Abfragen und Verschlüsselungstechniken. Die entsprechende Funktionalität ist also nicht dem Programmierer überlassen, sondern fest im JCRA integriert. Wenn eine PIN nach wiederholtem Versuch nicht korrekt präsentiert wird, wird das Applet oder sogar die ganze Karte blockiert. Das gewährleistet erhöhte Sicherheit.

Java und Java für Chipkarten haben aber verschiedene Sicherheitskonzepte. Java versucht feindliche Applets in ihrer Wirkungsmöglichkeit einzuschränken. Das Sicherheitskonzept von Java ist in der Sprache selbst formuliert. Java für Chipkarten versucht viele Fehler bei der Erstellung korrekter Programme durch die Sprache selbst zu vermeiden. Damit bietet sie Unterstützung gegen Angriffe von außen.

2.11. Java Card Entwicklungsumgebungen

Die Entwicklung eines Java Card Applets kann auf verschiedene Arten ablaufen. Die im Rahmen dieser Arbeit zur Verfügung gestellten Entwicklungsumgebungen (IDE, Integrated Developer Environment), wie Aspects Developer, Sm@rtCafe Professional, GemXpresso RAD und JCOP, erwiesen sich als sehr hilfreich. Fast alle Entwicklungsumgebungen unterstützen den Appletentwurf, indem sie vorgefertigte Java Card Applet Skeletons bereitstellen. Das vorgefertigte Skeleton beinhaltet einen Packagenamen, einen Appletnamen, alle wichtigen Methoden und den Konstruktor. Der Entwickler muss anschließend nur die benutzerdefinierten Methoden schreiben und die process-Methode anpassen.

Der Entwickler kann also eine Entwicklungsumgebung oder die Programmierplattform für Java für Chipkarten von SUN benutzen. Die Firma SUN stellt alles, was man zum Entwerfen und Arbeiten mit dem Java Card Applet braucht, im Netz frei zur Verfügung. Hier schreibt der Entwickler das Applet wie ein gewöhnliches Javaprogramm in einem Editor. Dabei dürfen nur die Klassen des Java Card Frameworks, bzw. sonstige Klassen, die auf der Karte vorhanden sind, verwendet werden. Mit der SUN-Plattform kann man es kompilieren, so dass man ein Class File erhält. Das Class File wird dann durch den Konverter verarbeitet. Der Konverter erstellt ein CAP File (Java Card spezifisches Format) und ein Export File. Das CAP File kann danach mittels eines Kartenlesers auf die Karte geladen werden. Die Programmierplattform für Java Card Applets von SUN bietet leider keinen Debugger. Auch ein Simulator steht bei der Programmierplattform für Java Card Applets von SUN nicht zur Verfügung.

Die Tabelle 2.11.1 stellt alle Entwicklungsumgebungen und die Programmierplattform für Java Card Applets von SUN zusammen. Hier sind die wichtigsten Merkmale der verschiedenen Entwicklungsumgebungen aufgelistet. Weitere Details werden bei den jeweiligen Beschreibungen der Entwicklungsumgebungen erklärt.

Alle Entwicklungsumgebungen (außer von Sun) bieten eine Benutzeroberfläche an. Alle diese Programme haben eine Homepage, aber nicht alle haben Arbeitsgruppen. Leser bedeutet, dass eine Schnittstelle zur terminalunabhängigen Einbindung von Chipkarten in PC-Programmen vorhanden ist. Es werden weltweit zwei Industriestandards, nämlich PC/SC (Personal Computer/Smart Card) und OCF (Open Card Framework) benutzt. Die PC/SC Spezifikation funktioniert auf allen von Windows unterstützten Rechnern und ermöglicht die Anbindung von Chipkarten in beliebige Anwendungen unabhängig von einer Programmiersprache (C, Java, Basic). Sie benötigt einen passenden Treiber für das verwendete Terminal, und die verwendete Chipkarte muss PC/SC kompatibel sein. Die OCF Spezifikation unterscheidet sich von der PC/SC Spezifikation dadurch, dass sie unabhängig vom Betriebssystem des PCs und von der jeweiligen Anwendung auf der Chipkarte ist. Die javabasierte Schnittstelle ermöglicht einen Zugriff aus PC-Programmen auf die Chipkarten-Anwendung.

	SUN	Aspects	Sm@rtCafe	GemXpresso	JCOP
Hersteller	SUN	Aspects	G&D	GEM+	IBM
Benutzer-oberfläche	nein	ja	ja	ja	ja
Homepage/ Arbeitsgruppen	ja/ja	ja/nein	ja/nein	ja/?	ja/ja
Unterstütze Java Cards	alle	Aspects, G&D, GEM+, IBM, Schlumberger	nur G&D	nur GEM+	nur IBM
Leser		PC/SC	PC/SC	PC/SC, OCF	PC/SC
Ausgangdatei	java	java	java	java	java
Compiler	möglich => class	möglich => class	1. mal nicht möglich	möglich => class (JBuilder)	möglich => class (IBM IDE)
Converter	möglich => cap	möglich => cap	möglich => cap	möglich => cap	möglich => cap
vorgefertigtes Java Card Applet Skeleton	nein	ja	nein	ja	ja
Debugger	nicht möglich	möglich	möglich	möglich	möglich
Simulator	nein	ja	ja	ja	ja
Skript (Skriptarten)	ja (SUN spezifisch)	ja (Aspects spezifisch)	ja (SUN spezifisch)	ja (?)	ja (?)

Tabelle 2.11.1 Entwicklungsumgebungen Vergleich

Aspects Developer

Aspects Developer ist eine Entwicklungsumgebung der Firma Aspects (www.aspect-sw.com). Die Entwicklungsumgebung unterstützt Java Cards diverser Kartenhersteller. In einem XML File werden die Load und Security Parameter jeder Java Card eingegeben, damit man später die gewünschte Java Card auswählen und testen kann. Die graphische Oberfläche von Aspects Developer verfügt über mehrere Views. Zu den Views gehören unter anderem der Source Code Editor, der Project Explorer, der Output, die JavaScript Console, der Card Explorer, das APDU Script und das Properties Window.

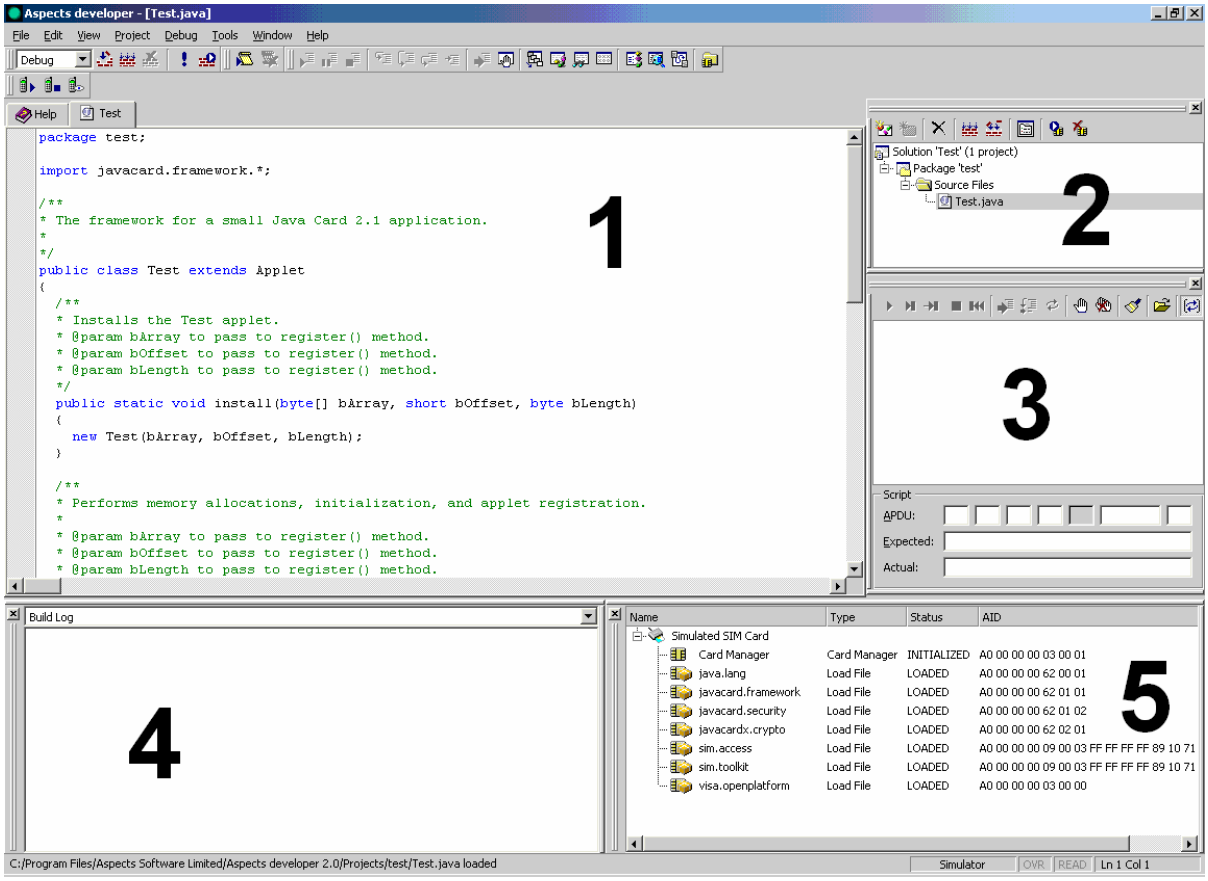


Abbildung 2.11.1 Aspects developer Entwicklungsumgebung

In Abbildung 2.11.1 sind nur die meistbenutzten Views zu sehen: Source Code Editor (1), Project Explorer (2), APDU Script (3), Output (4) und Card Explorer (5).

Das Bild zeigt auch die Ausgangssituation mit einem Java Card Applet Skeleton, nachdem man ein Projekt angelegt hat. Die Entwicklungsumgebung liefert oben genanntes, vorgefertigtes Java Card Applet Skeleton. Um das Skeleton zu erhalten, muss der Entwickler zuerst ein Projekt anlegen, indem er in einem Hilfsfenster die Package- und Appletnamen, sowie die Package- und Applet AID einträgt. Wenn die Angaben den spezifischen Normen des Aspects Developer entsprechen, bekommt man im Source Code Editor ein Applet Skeleton. Dieses Skeleton kann man übernehmen und benutzerspezifisch ergänzen. Es besteht auch die Möglichkeit, das Java Card Applet Skeleton durch ein woanders geschriebenes (in Sun, in einem Editor, usw.) Java Card Applet zu ersetzen. Das Project Explorer Fenster enthält den Inhalt eines Projekts. Es werden dort alle Packages, Java Files und Applets, die zu einem Projekt gehören, aufgelistet. Später kommen noch die Class Files, CAP-, JCA- und EXP Files dazu. Im APDU Script Fenster wird ein Skript aufgerufen und geöffnet. Ein Skript muss der Entwickler selbständig, ohne dass etwas vorgegeben ist, schreiben. Dabei ist es wichtig, dass das Skript dieselben Kommandos enthält, welche im Quellcode definiert worden sind, da sonst die Kommunikation zwischen Karte und Terminal nicht zu Stande kommen kann.

Eine Skript Datei hat immer die Dateiendung .scr. Im Aspects Developer wird das Skript als im Ganzen abgearbeitet, ein einzelnes APDU Kommando abzufragen ist nicht möglich. Im Output Fenster sieht man, wenn man APDU Log auswählt, den Kommunikationsaustausch zwischen Terminal und Karte. Wenn man dagegen den Build Log auswählt, bekommt man Informationen, ob ein Package erfolgreich kompiliert und/oder konvertiert wurde. Wenn Fehler auftreten, werden diese dort dann angezeigt. Im Card Explorer Fenster stehen die Informationen über den Inhalt einer ausgewählten Java Card. Dort sieht man die auf der Java Card gespeicherten Standardpakete, wie java.lang, javacard.framework usw. aber auch die benutzerdefinierten Packages und Applets (die werden allerdings als Package- und Applet AID angezeigt). Properties Window beinhaltet alle genaueren Informationen über das Applet, das Package und das Projekt. Aspects Developer kann den Quellcode kompilieren und konvertieren. Wenn alles erfolgreich durchgeführt wurde, enthält man im Project Explorer die Notiz, dass ein *.class, ein *.cap, ein *.exp und ein *.jca File (siehe Kapitel 2.7) im Projekt vorhanden ist. Falls Fehler auftreten, werden sie im Output Fenster unter Build Log aufgelistet. Über einen Mausklick auf den aufgelisteten Fehler lokalisiert man die Zeile im Quellcode, wo sich der Fehler befindet. Diese Funktion ist sehr hilfreich und reduziert die Korrekturzeit. Falls man die Fehler nicht nachbessern kann, kann ein vorhandener Debugger hilfreich sein. Aspects Developer verfügt über einen Simulator, der eine Standard Java Card simulieren kann. Man kann also ein Java Card Applet zuerst in einem Simulator testen, bevor man es mit einer realen Java Card versucht. Der erfolgreiche Ablauf beim Simulator garantiert allerdings nicht, dass das Java Card Applet auf jede beliebige Java Card geladen und ausgeführt werden kann. Dagegen garantiert ein nicht erfolgreicher Ablauf auf dem Simulator, dass man das Java Card Applet auf keiner realen Java Card ausführen kann.

Sm@rtCafe Professional

Sm@rtCafe Professional ist eine Entwicklungsumgebung von Giesecke & Devrient (www.smartcafe.gieseckedevrient.com) und wird für die Entwicklung von Java Applets für Java Cards der Firma Giesecke & Devrient verwendet. Eine freie Version von Sm@rtCafe Professional ist im Netz erhältlich. Sm@rtCafe Professional stellt eine Reihe von Views zur Verfügung. Zu sehen sind folgende Views: Source Code Editor, Project, Stack, Locals, Watch, Heap, Breakpoints, Logbook, Script Editor und Output.

Die Abbildung 2.11.2 zeigt die Sm@rtCafe Professional Entwicklungsumgebung, nachdem man ein Projekt angelegt hat. Da das Programm kein vorgefertigtes Java Card Applet Skeleton bereitstellt, ist das Source Code Editor Fenster noch nicht aktiv.

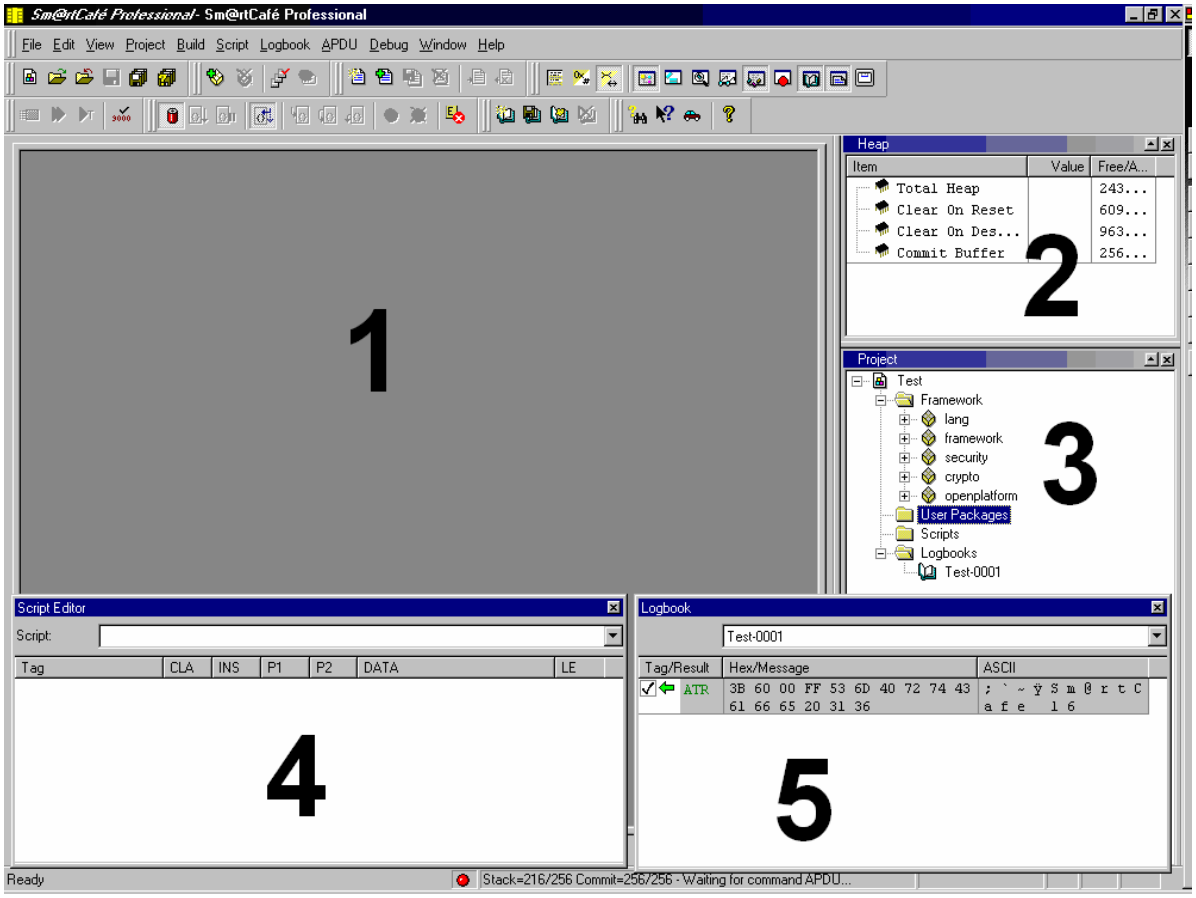


Abbildung 2.11.2 Sm@rtCafe Professional Entwicklungsumgebung

In Abbildung 2.11.2 sind nur die meistbenutzten Views zu sehen: Source Code Editor (1), Heap (2), Project (3), Script Editor (4) und Logbook (5).

Die Entwicklungsumgebung liefert kein vorgefertigtes Java Card Applet Skeleton. Der Entwickler muss ein Java Card Applet woanders (z. B. bei SUN) schreiben, kompilieren und konvertieren. Danach kann man das Applet über das User Package in der Project-View aufmachen. Erst dann kann man den Quellcode verändern und erneut kompilieren und konvertieren. Später wird der Source Code im Source Code Editor angezeigt. Im Heap Fenster können die Werte (z. B. ein durch eine Methode geänderter Kontostand) in zwei verschiedenen Darstellungsformen angezeigt werden, entweder in hexadezimaler oder in dezimaler Form. Das Project Fenster zeigt den Inhalt eines Projekts, unter anderem die Standardpakete wie java.lang, javacard.framework, die Packages, die Applets, die Skripts und die Logbooks. Im Script Editor Fenster wird später der Inhalt des Skripts angezeigt. Das Skript wird von einem Entwickler nach Vorgaben von SUN geschrieben (unterscheidet sich vom Skript von Aspects). Im Logbook Fenster wird man die Kommunikation zwischen Terminal und Karte sehen. Nachdem das Kommando Select erfolgreich abgearbeitet wurde kann jedes APDU Kommando einzeln ausgewählt und abgearbeitet werden. Sm@rtCafe Professional stellt sowohl einen Debugger als auch einen Simulator zur Verfügung.

GemXpresso RAD

GemXpresso RAD von GemPlus (www.gemplus.com/developers) wird als Plug-In in JBuilder integriert und kann damit für die Java Card Entwicklung benutzt werden. GemXpresso RAD kann nur GemPlus Java Cards testen.
Die Entwicklungsumgebung stellt eine Reihe von Fenstern zur Verfügung. Die wichtigsten sind: Project, Content, Structure, Messages und Status Bar.
In Abbildung 2.11.3 sieht man: Project (1), Structure (2), Content (3) und Messages (4). Die Elemente eines Java Card Applet stehen im Structure Fenster. Es werden sowohl die Standardpakete importiert als auch ein Paket com.borland.*. Im Project Fenster sind alle Dateien aufgelistet, wie Java File, Class File usw. GemXpresso liefert ein Java Card Applet Skeleton, das im Content Fenster zu sehen ist. Das Java Card Applet wird durch JBuilder4 kompiliert. Das Messages Fenster informiert darüber, was gerade passiert. Ein Simulator sowie ein Debugger sind in diesem Programm enthalten.

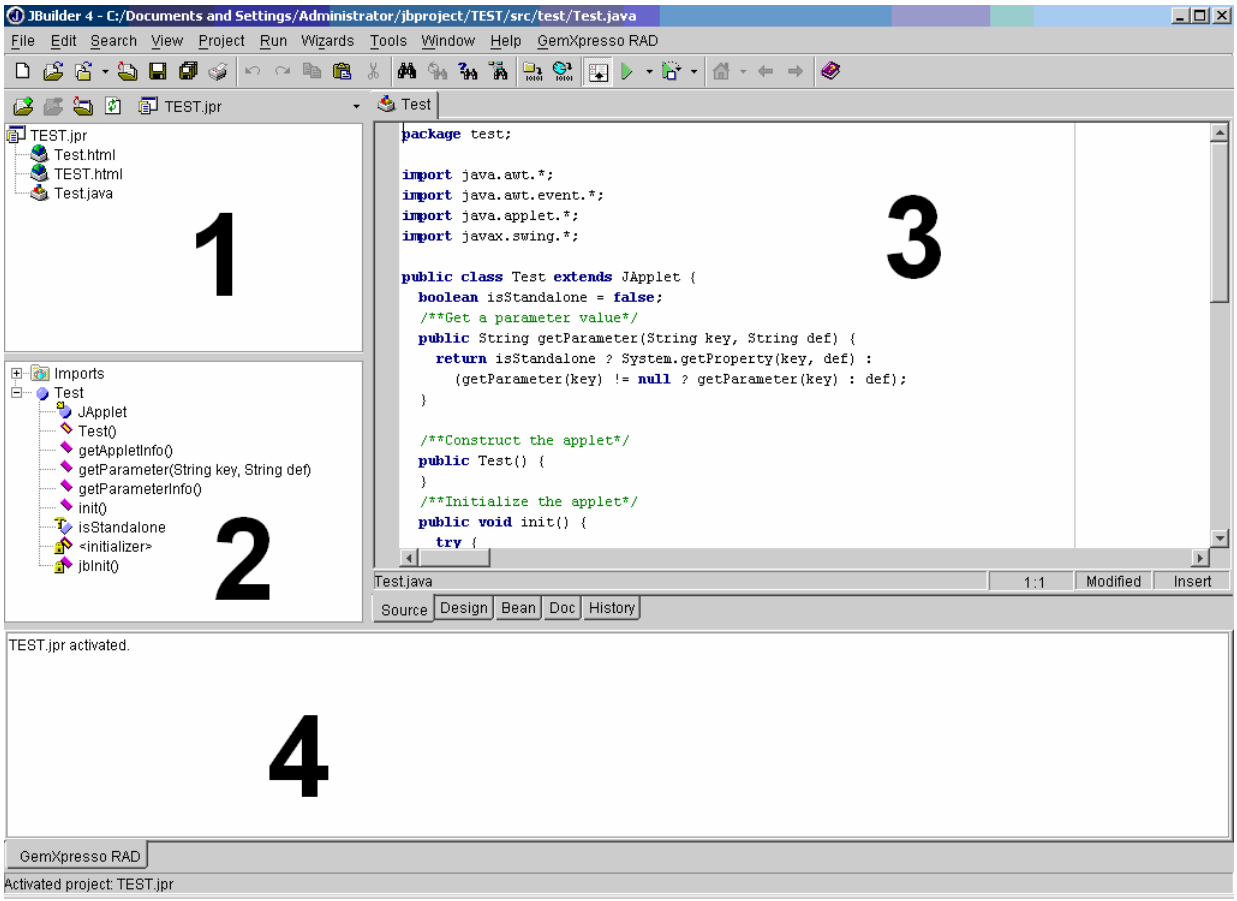


Abbildung 2.11.3 GemXpresso RAD Entwicklungsumgebung

JCOP

JCOP ist eine Entwicklungsumgebung der Firma IBM (www.zurich.ibm.com/javacard). JCOP kann nur Java Cards der Firma IBM testen. Wie man in Abbildung 2.11.4 sehen kann, enthält die Entwicklungsumgebung weniger Fenster als die vorher erwähnten Entwicklungsumgebungen. Es sind der Source Code Editor (1) und das Messages Fenster (2). JCOP liefert ein vorgefertigtes Java Card Applet Skeleton. Das Source Code Editor Fenster beinhaltet den Quellcode, den man kompilieren und konvertieren kann. Der Quellcode beinhaltet automatisch das Standardpaket javacard.framework und auch die JCOP spezifischen Pakete sim.toolkit.* und sim.access.*, die GSM Anwendungen unterstützen (Java Card API 2.1). Das Messages Fenster ist dafür da, um den Benutzer über die durchgeführten Aktionen, wie z.B. das Kompilieren zu informieren. Das Programm stellt einen Simulator und Debugger zur Verfügung.

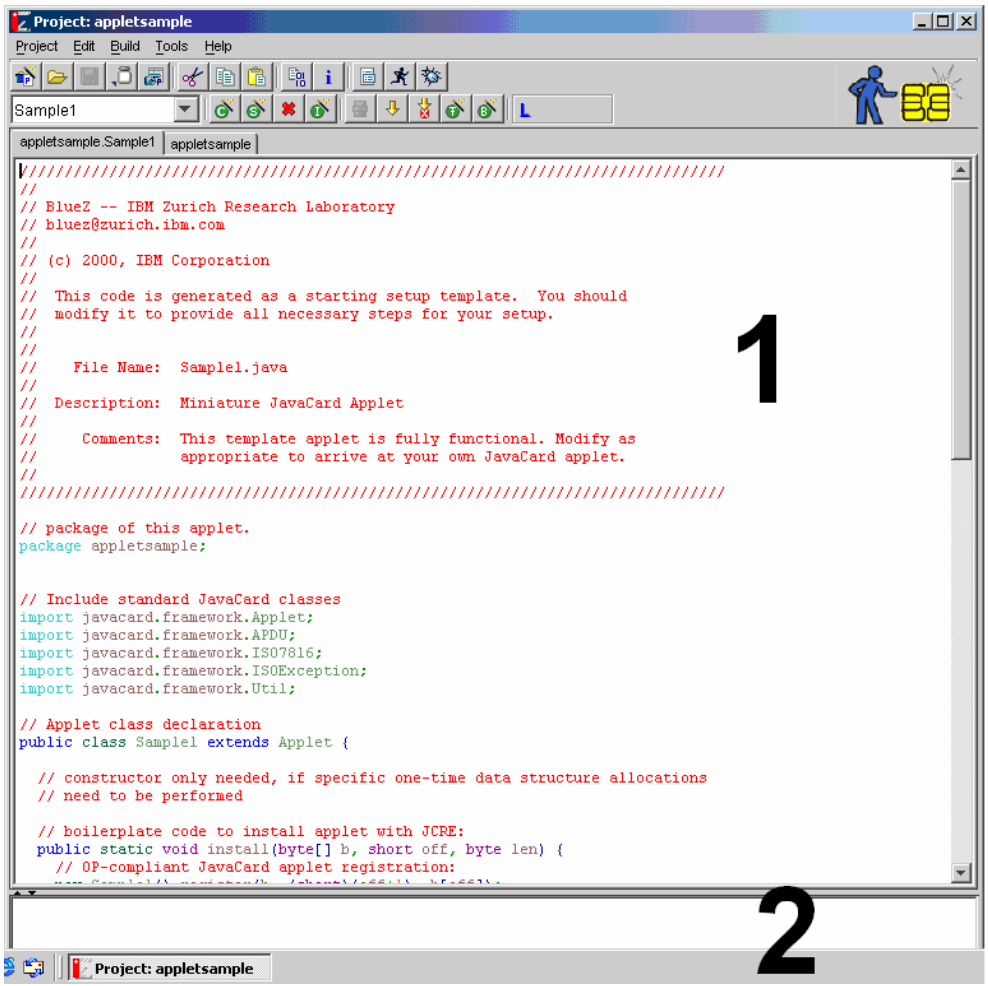


Abbildung 2.11.4 JCOP Entwicklungsumgebung

In der Tabelle 2.11.2 wurden für alle Entwicklungsumgebungen alle Pakete aufgelistet. Es sind sowohl die Standardpakete (javacard.framework, javacardx.crypto und javacard.security) als auch GSM Pakete. Die Entwicklungsumgebungen Aspects, GemXpresso, GemXplorer und JCOP beinhalten zusätzlich GSM Pakete, die es ermöglichen GSM Anwendungen auf der Java Card zu laden und zu testen.

IDE Pakete	Aspects	Sm@rtCafe	GemXpresso, GemXplorer	JCOP	Scard (Infineon Technologies)
	Visa OP		Visa OP		Visa OP
GSM	sim.access, sim.toolkit		SIM Package	SIM Package toolkit.sim	
	JC 2.0	JC 2.0	JC 2.0	JC 2.0	JC 2.0
standard	framework	framework	framework	framework	framework
standard	crypto	crypto	crypto	crypto	crypto
standard	security	security	security	security	security

Tabelle 2.11.2 Pakete

3. Metrik

Mit Hilfe der Metrik schafft man sich die Vorraussetzung, aussagekräftige Benchmarks durchzuführen, da sie festlegt, was, wie getestet werden kann und soll, bzw. was sinnvoll ist zu testen. Deshalb wurden eingangs Experten befragt, um oben genannte Fragen beantworten zu können. Anschließend wird mit Hilfe dieser Basis eine Benchmarking Performance Metrik erarbeitet und vorgestellt. Außerdem wird eine Methodik präsentiert, wie die Ergebnisse der Messungen auszuwerten sind.

3.1. Zusammenfassung der Expertenbefragung

In Rahmen der Arbeit wurden Chipkartenexperten aus den verschiedenen Abteilungen wie Applikation Engineering, Technologie Entwicklung, Operations und Marketing befragt. Ziel der Befragung war es, Themengebiete kennen zu lernen, um Schwerpunkte für das Benchmarking von Java Cards zu identifizieren. Die Befragten hatten einen vorgefertigten Fragebogen ausgefüllt und sich einem 20-minutigen Interview unterzogen.

Dabei haben sich mehrere Interessenschwerpunkte gebildet:

- Ausführungszeiten von Kryptoalgorithmen, insbesondere von DES und RSA
- Laufzeiten von Sprachelementen, wie z. B. arithmetische Operatoren (Addition, Division, Multiplikation), logische Operatoren (!, =,<,>), Java Card Sprachelemente (if_else, switch, while, for, do), Arrays, Objektzugriffe
- Ladezeiten von Packages und Zeitverbrauch für Select Applet
- Speicheraulastung und Speicherzugriffszeiten, die über Tests von EEPROM, RAM und CPU ermittelt werden können
- Sicherheit von Java Cards (siehe dazu Kapitel 2.10)
- Energieverbrauch bei Java Cards, den man durch Oszilloskopmessungen ermitteln kann
- Qualität der Java Card Virtual Machine, z.B. bezgl. Schnelligkeit und Speichermanagement

Auf Basis der Expertenbefragung wurde das Themenspektrum auf die Ausführungszeiten von Kryptoalgorithmen, die Laufzeiten von Java Card Sprachelementen, den Energieverbrauch, die Speicheraulastung und die Speicherzugriffszeiten reduziert.

3.2. Analyse der Chipkartenfunktionen

Für jede Anwendung wird eine Chipkarte mit unterschiedlichen Aufgaben benötigt. So unterscheiden sich z.B. die Aufgaben einer Bankkarte von denen einer Zutrittskarte. Die verschiedenen Aufgaben benötigen eine optimale Hardware. Für eine Bankkarte wird die optimale Hardware anders zusammengesetzt als für eine Zutrittskarte. Erstere benötigt vor allem einen Kryptocoprozessor, Letztere dagegen eine schnelle CPU.
Infineon Technologies will für jede Anwendung einen optimalen Chip anbieten. Deswegen haben die Mitarbeiter der Firma Infineon Technologies, S. Rüping und H. Hewel in einem Innovation Letter [RH01] die Methodik zum Durchführen eines Benchmarking von Smart Cards präsentiert. Sie haben alle ihnen frei zugängliche Smart Cards Quellcodes für diverse Anwendungen (C++, Assembler, Java) untersucht. Die gesammelten Informationen sollen helfen, die beste Hardware für Smart Cards zu bestimmen, um danach die Hardwarekomponenten aufeinander abzustimmen. Es sollte auch möglich sein, die beste Cachegröße zu ermitteln und die Systemparameter zu optimieren. Durch das Benchmarking sollte auch der Flaschenhals des Systems gefunden werden, der bei Performanzmessungen berücksichtigt werden muss.

Die Chipkartenfunktionen wurden im Innovation Letter [RH01] in Gruppen aufgeteilt. Diese Aufteilung wurde für unten angegebene Metrik übernommen (siehe die Tabelle 3.4.1; 1. Spalte). Es gibt folgende Chipkartenfunktionen:

- Checksums/Hash
- File System
- Cryptography
- Communication
- Biometric
- Java
- Multiplication

Diese Chipkartenfunktionen beinhalten mehrere untergeordnete Funktionen. Jeder einzelnen Unterfunktion wird eine Abkürzung zugeordnet (siehe die Tabelle 3.4.1; 2. Spalte). Basierend auf diesen Unterfunktionen und deren individuellen Laufzeiten können Performanzergebnisse für mehrere, verschiedene Anwendungen generiert werden.

Die Applikationen wurden im Innovation Letter [RH01] nach dem Anwendungsbereich in Applikationsklassen unterteilt, nämlich:

- Banking
- GSM (SIM Card)
- PKI (ECommerce)
- Authentication
- Personalization (source)
- Personalization (pain)
- Biometrics
- Java Card

Für jede Applikationsklasse wird eine Formel aufgestellt. Diese Formel enthält eine Menge von Unterfunktionen der Chipkartenfunktionen. Diese Unterfunktionen werden zusätzlich mit Gewichten versehen. Die Gewichte besagen, wie oft eine Unterfunktion in einer Applikation abgearbeitet wird.

Um sich die Situation besser vorzustellen, wird ein Beispiel (für GSM Anwendung) vorgestellt (siehe Abbildung 3.2.1). Diese Anwendung benötigt verschiedene Unterfunktionen.

Die Performance für eine GSM Applikation setzt sich aus denjenigen Zeiten zusammen, die die folgenden Unterfunktionen mit ihren Gewichten (Count) für ihre Verarbeitung benötigen:

- 2x CryptInitialize ($t_{C\_I\_3DES}$)
- 2x CryptUpdate ($t_{C\_U\_3DES}$)
- 2x CryptFinalize ($t_{C\_F\_3DES}$)
- 10x FileOpen (t_{FS_O})
- 2x Write (t_{FS_W})
- 200x Read (t_{FS_R})
- 10x FileClose (t_{FS_C})
- und jeweils Zeiten zwei anderer, namenloser Funktionen

Function	Count	Description	Remarks
SIM (GSM)			
CryptInitialize	2	3DES, CBC	
CryptUpdate	2	→ Data (24 Byte)	
CryptFinalize	2	→ Data (8 Byte)	
FileOpen	10		
Write	2	EEPROM Write	
Read	200		
FileClose	10		
	1	looking for simple data in a complex TLV structure, according to ISO 7816 / EMV	Implemented in C-Code of Benchmark!
	1	I/O Operations (10kBytes)	??????

Application SIM (GSM)

$$Perf_{GSM} = 2x\ t_{C\_I\_3DES} + 2x\ t_{C\_U\_3DES} + 2x\ t_{C\_F\_3DES} + 10x\ t_{FS\_O} + 2x\ t_{FS\_W} + 200x\ t_{FS\_R} + 10x\ t_{FS\_C} + time(\text{C-routines "search"}) + time(I/O)$$

Abbildung 3.2.1 GSM Applikation [RH01]

Jede andere Applikation beinhaltet eine andere Formel, d.h. eine andere Zusammensetzung von Unterfunktionen und deren Gewichtung als die vorgestellte GSM Applikation. Die unterschiedlichen Formeln für verschiedene Anwendungen, deren Unterfunktionen und dazugehörige Gewichtungen, werden für die endgültige Metrik für das Benchmarking von Java Cards übernommen und weiterverwendet.

3.3. Gruppierung für Funktionsklassen

Um die Java Cards zu vergleichen, braucht man unabhängig von der Applikation eine geeignete Metrik. Im Rahmen dieser Arbeit wurden verschiedene Funktionsklassen gruppiert. Diese Funktionsklassen setzen sich aus der API, dem Java Card Native Set und den Language Extensions zusammen.

Daraus sind acht Funktionsklassen entstanden:

- Operatoren
- Schreib- und Lesezugriffe
- Appletmethoden und Objektmethoden
- Arrays
- Java Card Sprachelemente
- Java Card System
- Sicherheit
- Visa Open Platform

Jede Funktionsklasse besteht aus verschiedenen Elementen, die in der Tabelle 3.3.1 genauer beschrieben werden.

Funktionsklassen	Library	Beschreibung
Operatoren	java.lang	Arithmetische Operationen wie Addieren, Multiplizieren,... und Logische Operatoren wie Negation, AND, OR,…
Schreib- und Lesezugriffe	framework	Schreib und Lesezugriffe auf RAM und EEPROM, Dateioperationen
Appletmethoden und Objektmethoden	framework	Select, Deselect, Register, Install, Process, Aufrufen von Methoden, Anlegen von Objekten
Arrays	framework	Array Copy, Array Compare
Java Card Sprachelemente	java.lang	if_else, do, for, while, switch, ternary, break_continue
Java Card System	framework	Transaktionen, Sharable Interface
Sicherheit	crypto, security	DES, RSA, PIN, Key, Signature
Visa Open Platform	VOP	Security Domains, OPSystem

Tabelle 3.3.1 Die genauere Beschreibung des Spinnennetzes

Zur Funktionsklasse Operatoren gehören Elemente wie arithmetische Operationen und logische Operatoren. Die Elemente der Funktionsklasse Schreib- und Lesezugriffe setzen sich aus allen Schreib- und Lesezugriffen auf dem RAM und dem EEPROM und verschiedenen Dateioperationen zusammen. Die Elemente der Funktionsklasse Appletmethoden und Objektmethoden beinhalten alle Appletmethoden, wie z. B. Select, Install und Process. Zu dieser Funktionsklasse gehören außerdem alle Objektmethoden, die das Anlegen von Objekten und das Aufrufen von Methoden durchführen. Das Anlegen, Kopieren, Hinzufügen und Löschen von Arrays sind Elemente, die zu der Funktionsklasse Arrays gehören. Die Elemente der Funktionsklasse Java Card Sprachelemente sind alle Sprachelemente der Programmiersprache Java, die auch in Java für Chipkarten benutzt werden können. Dazu gehören unter anderem if_else, do, while, switch und andere (siehe dazu Kapitel 6.3). Zum Java Card System gehören Transaktionen und Elemente aus dem Shareable Interface. Die Funktionsklasse Sicherheit beinhaltet z. B. die Algorithmen DES und RSA. Zur Visa Open Platform gehören unter anderem die Elemente OP System (Open Platform System) und Security Domains.

Um das Benchmarking auf Javaebene durchzuführen, müssen alle Funktionsklassen mit genügend vielen Elementen getestet werden. In einem Applet kann man z.B. mehrere Elemente aus einer Funktionsklasse testen. Die Werte der Ergebnisse werden dann auf einem Spinnennetz, siehe Abbildung 3.3.1, platziert. Jedem Namenspfad ist eine Funktionsklasse zugeordnet.

Im Folgenden soll beschrieben werden, wie ein Wert, den man in das Spinnennetz eintragen will, berechnet wird:

- Die Ergebnisse (Zeiten in msec) von allen getesteten Elementen aus einer Funktionsklasse werden zusammengerechnet. Die Gesamtzeit wird durch die Anzahl der Elemente geteilt. So enthält man einen Mittelwert (in msec) einer Funktionsklasse. Dieser Wert wird für alle Java Cards errechnet.

$$\text{א} = (\sum_{1\ldots n} t) / (n)$$

  א = Mittelwert einer Funktionsklasse in msec
  t = Zeit in msec
  n = Anzahl der getesteten Elementen aus einer Funktionsklasse

- Man enthält m (m = Anzahl der Java Cards) Mittelwerte. Der beste Mittelwert enthält 100 Punkte. Für alle anderen Karten werden deren Werte proportional verkleinert. Die Karte, mit der man keine Elemente einer Funktionsklasse testen konnte, (das Applet, das die jeweiligen Elemente einer Funktionsklasse enthält, konnte nicht auf die Karte geladen und/oder ausgeführt werden) enthält einen Nullpunkt. Wenn kein Wert eingetragen wird, bedeutet das, dass die jeweiligen Elemente einer Funktionsklasse noch nicht getestet worden sind.

$$\text{Ж} = 100$$

- In gleicher Weise werden auch die Werte der anderen Funktionsklassen berechnet.

Für jede Java Card wird ein separates Spinnennetz erstellt. Dort werden alle Werte für getestete Funktionsklassen eingetragen.

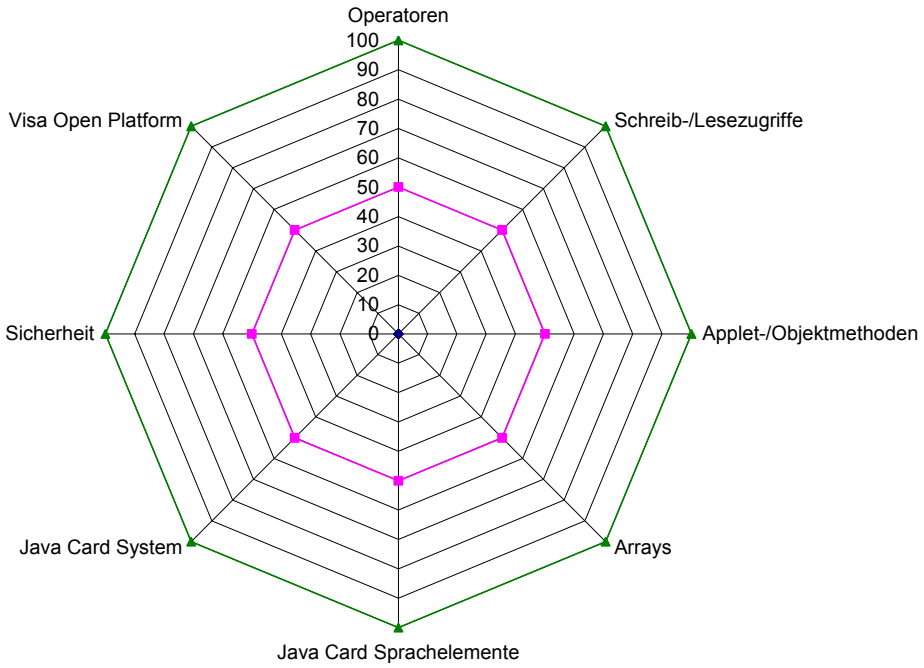


Abbildung 3.3.1 Spinnennetzvorlage

Die minimalen Anforderungen für die jeweiligen Funktionsklassen werden durch die verschiedenen Applikationsanforderungen gegeben. Das bedeutet, dass die verschiedenen Anwendungen verschiedene Anforderungen an die jeweiligen Funktionsklassen haben. Für eine Geldkarte, zum Beispiel, wird es vermutlich so sein, dass sie eine sehr hohe Anzahl an Punkten (zwischen 80 und 100 Punkte) für den Namenspfad Sicherheit benötigt. Für andere Namenspfade gilt, sie sollten mindestens 50 Punkte erreichen. Für andere Applikationen sind natürlich andere Zusammensetzungen von Punkten für die jeweiligen Namenspfade erforderlich.

3.4. Auswertungsmethodik

In Kapitel 3.2 wurde die Applikationsebene und in Kapitel 3.3 die Javaebene vorgestellt. In diesem Kapitel sollen die beiden Ebenen zusammengeführt werden. Die Abbildung 3.4.1 zeigt, dass sowohl die Art der Applikation als auch Java Card Elemente aus der Javaebene das Benchmarking von Java Card Chipkarten beeinflussen.

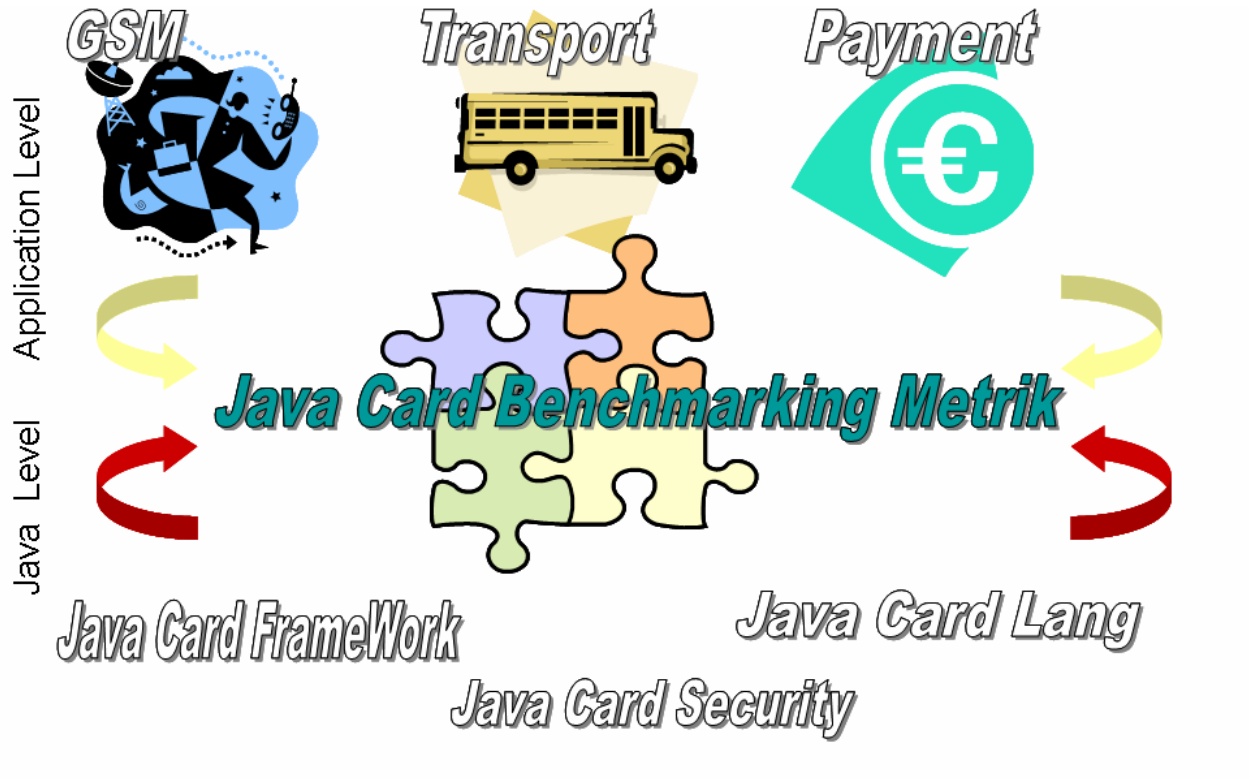


Abbildung 3.4.1. Java Card Benchmarking [private Sammlung von B. Lippmann]

Die Chipkartenfunktionen mit den Unterfunktionen aus dem Innovation Letter [RH01] wurden für diese Arbeit übernommen. Jeder einzelnen, bestehenden Unterfunktion werden entsprechende Java Card Programmiersprachenelemente mit deren Bibliotheken zugeordnet (siehe dazu die Tabelle 3.4.1; Spalte Java Methode und Spalte Bibliothek). Dann werden den Javamethoden die Funktionsklassen zugeordnet. In der letzten Spalte befinden sich die Hardwaremodule, die bei der Abarbeitung der Javamethode beansprucht werden.

Für die verschiedenen Applikationen wurde die Zusammensetzung (die Formel) der Unterfunktionen (der Chipkartenfunktionen) und die dazugehörenden Gewichte beibehalten.

Die Tabelle 3.4.1 (vollständige Tabelle befindet sich im Anhang) stellt den Zusammenhang zwischen Applikationsebene und Javaebene dar.

Applikation Ebene		Java Ebene			
Chipkartenfunktionen und Unterfunktionen	**Funktion Abkürzung**	**Java Methode**	**Bibliothek**	**Funktionsklasse**	**Hardware -Modul**
Checksums/Hash					
CHInitialize	t_{CH_I}				
CHUpdate	t_{CH_U}				
File System					
File Open	t_{FS_O}	getCurrentElementrayFile()			
		getCurrentDidicatedFile()			
Read	t_{FS_R}	getData()		Lesezugriff	EEPROM
Write	t_{FS_W}	updateRecord()		Schreibzugriff	
		putData()			
		writeRecord()			
WritePhysical	t_{FS_WP}	arrayCopyNonAtomic()		Arrays	
		arrayCopy()		Arrays	
Cryptography					
CryptoInitialize	$t_{C\_I\_X}$	getKey()	javacard.security	Sicherheit	
		setKey	javacard.security	Sicherheit	
	X=DES	getInstance()	javacard.security	Sicherheit	DES
		buildKey()	javacard.security	Sicherheit	DES
	X=RSA	getInstance()		Sicherheit	Crypto CP
		getExponent()		Sicherheit	Crypto CP
		getModulus()		Sicherheit	Crypto CP
GenerateRandom	t_{C_R}	generateData()	javacard.security	Sicherheit	
		getInstance()	javacard.security	Sicherheit	
Communication					
ComSend	$t_{COM\_S\_X}$	sendOutGoing()	javacard.framework		
		setOutGoingAndSend()	javacard.framework		
		sendBytes()	javacard.framework		
		sendBytesLong()	javacard.framework		
		setOutGoing()	javacard.framework		
		setOutGoingLenth()	javacard.framework		
ComReceive	$t_{COM\_R\_X}$	setIncomingAndReceive()	javacard.framework		
		getBytes()			
		getBuffer()			
		receiveBytes			
Java					
Java VM	t_{J_B}	install()	javacard.framework	Appletmethoden	
		select()	javacard.framework	Appletmethoden	
		process()	javacard.framework	Appletmethoden	
		deselect()	javacard.framework	Appletmethoden	
Java_Language	t_{J_all}	if…else	java.lang	JC Sprachelemente	
		while	java.lang	JC Sprachelemente	
		switch	java.lang	JC Sprachelemente	
		do	java.lang	JC Sprachelemente	
		Addition	java.lang	Operatoren	CPU
		Multipilkation	java.lang	Operatoren	CPU
		And	java.lang	Operatoren	CPU

Tabelle 3.4.1 Zusammenhang zwischen Applikationsebene und Javaebene

3.5. Vergleich mit EEMBC

In Rahmen der Arbeit stieß man auf die Internetseite www.eembc.org. Die Firma EEMBC bietet ein kommerzielles Benchmarking, wie z. B. für PCs, PDAs und Handys. Die Tests können auf den Geräten durchgeführt werden, und die Testergebnisse werden auch auf demselben Gerät ausgegeben. Das ist allerdings nicht bei Java Cards möglich, weil diese kein visuelles Ausgabefenster besitzen. Die Testergebnisse müssen an ein anderes Gerät übermittelt und dort ausgegeben werden.
Die professionell durchgeführten Vergleiche zeigen jedoch die gleiche Vorgehensweise wie die in Kapitel 3.4 vorgestellte Metrik.

4. Applet Design

Im Abschnitt 2.5 wurde eine kurze Zusammenfassung der Struktur eines Java Card Applet gegeben. Im folgenden Kapitel werden Designregeln vorgestellt. Hierbei werden Regeln für den Entwurf von Java Card Applets eingeführt und diskutiert. Als Erweiterung von bereits bekannten Designregeln werden im folgenden Abschnitt Programmierregeln zur Erstellung eines Java Card Applets, sowie Regeln zur Erstellung eines Benchmarking Java Card Applet angegeben, die es ermöglichen bestimmte Komponenten (CPU, Speichermodul, Kryptocoprozessor) einer Java Card Chipkarte zu analysieren, und die hierbei eine Trennung zwischen der verwendeten Hardware, der Plattform, der Implementierung der Java Virtual Machine auf der Plattform, sowie der ausführenden Java Software Ebene ermöglichen.
Die im Rahmen der Durchführung dieser Arbeit aufgetretenen Schwierigkeiten und Probleme legen jedoch eine geringfügige Ergänzung an einigen Stellen nahe.

4.1. Designregeln für Java Card Applets

Die Designregeln für die Erstellung eines Java Card Applets basieren auf Designregeln für die Erstellung Java Programme.

Wiederverwendbarkeit/Erweiterbarkeit
Die Java Card Applets sollen vollständig oder teilweise wieder verwendet werden. Sie sollen leicht veränderbar und erweiterbar sein.

Wiederholbarkeit
Die Tests eines Java Card Applets sollen unabhängig von der Anzahl der Tests immer die gleichen Messergebnisse liefern. Die Messergebnisse sollen also stabil sein.

Reproduzierbarkeit
Die Messergebnisse eines Java Card Applets sollen unabhängig vom Equipment sein.

Vergleichbarkeit
Ein Java Card Applet soll auf verschiedenen Java Card Plattformen laufen.

Korrektheit
Korrektheit heißt, dass ein Java Card Applet gründlich getestet wurde und keinen Fehler enthält. Man muss auch aufpassen, dass keine logischen Fehler unentdeckt bleiben. Um ein korrektes Java Card Applet zu schreiben, hilft ein guter Programmierstil.

Lesbarkeit
Um die Lesbarkeit eines Java Card Applets zu erhöhen, muss der Entwickler einen Stil entwickeln, indem er ein ordentliches Layout, insbesondere Einrückungen und Leerzeilen, und Kommentare, die die Funktion der nachfolgenden Anweisungen erklären, verwendet.

Effizienz
Einfache Java Card Applets sollen nicht zu lang werden und nicht zu langsam sein.

4.2. Allgemeine Programmierrichtlinien für Java Card Applets

Die allgemeinen Programmierrichtlinien zur Erstellung eines Java Card Applets sind zum Teil die gleichen wie die allgemeinen Richtlinien zur Erstellung eines Javaprogramms.

Klein- und Großschreibung beachten.

Während des Entwurfs eines Java Card Benchmarking Projektes muss der Anwender einen Package- und Appletnamen, sowie einen Package- und Applet AID vergeben. Verwendet man den Sun Java Card Compiler, werden diese Daten in einem *.opt File gehalten [SUN ...]. In den verwendeten Entwicklungsumgebungen sind diese Parameter frei wählbar und leicht editierbar. Nach dem Download eines Applets auf die Karte ist jedoch nur noch der Package und Applet AID im Java Card Manager sichtbar. Ein Konzept zur strukturierten Verwaltung dieser Parameter ist deshalb sehr hilfreich. Hierfür wurde folgendes Konzept verwendet:
Der Projektname, der Packagename und der Appletname müssen immer angegeben werden. Es kann sein, dass ein Projekt mehrere Packages beinhaltet. Es gibt aber immer nur ein Applet pro Package. Ein Package kann, muss aber nicht, ein oder mehrere Javaklassen (Hilfsklassen, z.B. für Exceptions) beinhalten.
Ein gutes Konzept sieht z.B. so aus:

- Der Projektname besteht nur aus großen Buchstaben, z.B. "TEST"
- Der Packagename besteht nur aus kleinen Buchstaben, z.B. "test"
- Im Appletnamen fängt jedes Wort mit großem Buchstaben an, z.B. "Test".

Bei zusammengesetzten Namen werden die jeweiligen Worte mit "_" voneinander getrennt z.B. "RAM_TEST", "ram_test" und "Ram_Test".
Man weiß anhand der Namen, um welchen Bestandteil es sich handelt.
Die Namen sollten auch beschreiben, was das Applet macht.
Die Namen des Projekts, des Packages und des Applets sollten alle immer gleich sein. So findet man schnell die zusammengehörenden Teile.
Methodennamen, Variablennamen und Klassennamen werden klein geschrieben (wie in Java).

Konzept für den Package AID und den Applet AID.

Anders als bei normalen Javaprogrammen besitzen Java Card Applets einen AID. Ein Java Card Applet gehört immer zu einem Package, welches auch einen AID besitzt.
Es ist empfehlenswert, dass für die verschiedenen Java Card Applets verschiedene AIDs vergeben werden. Diese sollen so gewählt werden, dass man anhand des AID weiß, um welche(s) Package(s) und/oder Applet es sich handelt.
Der Applet- und Package AID sollen immer im Quellcode als Kommentar abgespeichert werden, da nicht alle Entwicklungsumgebungen ein *.opt File besitzen.
Die Länge des AIDs ist von SUN empfohlen und bietet jeweils maximal 16 Stellen für den Applet AID und für den Package AID.
Man kann aber von den Vorgaben abweichen und ein eigenes System für die Vergabe von AIDs verwenden.
Die unten vorgestellte AID Systemvergabe wurde bei allen Java Card Applets im Rahmen der Arbeit verwendet. Dieses System hat sich hervorragend bewährt.

Beispiel:
Package AID: 04 03 04 00 01 00 00 00 00 00 EE
Applet AID: 04 03 04 00 01 EE

Der verwendete Applet AID hat eine Länge von 6 Bytes und der Package AID hat eine Länge von 11 Bytes.
Die ersten drei Bytes beim Applet AID und beim Package AID bezeichnen das Datum, an dem das Java Card Applet und Package erstellt wurden. In dem Beispiel ist es der 4-te März 2004.
Die nächsten zwei Bytes bezeichnen die Anzahl der Applets, die ein Entwickler an diesem Tag geschrieben hat. In diesem Beispiel ist es das erste Applet des Entwicklers Erdmann am 4-ten März 2004.
Das letzte Byte bezeichnet den Entwickler, hier Erdmann.
Im Package AID sind die Stellen zwischen der Anzahl der Applets und dem Entwicklerzeichen mit Nullen gefüllt.

Globale und lokale Variablen.

In Java Card Applets werden die globalen Variablen, Felder, Objekte, Arrays und der Appletcode im EEPROM persistent gespeichert. Deswegen sollten alle Objekte und Arrays beim Installieren angelegt werden (bzw. beim ersten select). Persistente Objekte, ihre Variablen und Zustände bleiben von ihrer Erzeugung bis zum Löschen des Applets auf der Java Card erhalten. Jede Verwendung des new- Operators erzeugt ein persistentes Objekt. Zugriffe auf Felder von persistenten Objekten sind atomar. Tritt ein Fehler während des Zugriffs auf (z.B. Stromverlust) behält des Feld seinen alten Wert. Wird ein persistentes Objekt nicht mehr referenziert, könnte es entsorgt werden. Da die JCRE aber typischerweise keinen Garbage-Collector implementiert, verbleiben unreferenzierte persistente Objekte als Datenmüll im EEPROM bis das Applet gelöscht wird.

Dagegen werden die lokalen Variablen und transiente Arrays im RAM flüchtig (transient) abgespeichert. Transiente Objekte sind insofern transient, als dass sie ihren Inhalt bei einem Reset der Karte oder bei einer Deselection des Applets verlieren (werden nicht wiederhergestellt). Die transienten Objekte selber bleiben erhalten. Sie existieren, solange sie referenziert werden. Zugriffe auf transiente Objekte sind nicht atomar. Nicht mehr referenzierte, transiente Objekte belegen weiterhin Speicherbereiche und nur bei der Implementierung eines Garbage-Collectors können sie entsorgt werden.
Man sollte also, wenn möglich, lokale Variable verwenden, da die wesentlich schneller als globale Variable sind (weil sie im RAM liegen). Daher ist es sinnvoll, z.B. temporär genutzte Arrays transient zu realisieren. Man sollte auch so wenige Objekte wie möglich erzeugen, da auf einem Chip relativ begrenzter Speicherplatz zur Verfügung steht.
Wenn man die Werte, die man kopiert, nicht zwischenspeichern will, benutzt man:

Beispiel:

```
  public void Appletmethode1 (APDU apdu){
  …
(*) Util.arrayCopyNonAtomic(Var_name1,(short)0,Var_name2,(short)0,(short)32);
  ...
```

Wenn man die Werte einfach nur kopieren will, und in Kauf nimmt, dass die Werte zwischengespeichert werden benutzt man an Stelle von (*):

Beispiel:

```
…
Util.arrayCopy(Var_name1,(short)0, Var_name2,(short)0,(short)32);
…
```

Andere Klassen

Die anderen Java Files, die zu einem Package gehören, können z.B. benutzerdefinierte Exceptions enthalten. Leider hat sich herausgestellt, dass zusätzliche Klassen den Ablauf verlängern. Deswegen vermeidet man bei kleinen Packages die Verwendung von zusätzlichen Klassen. Bei großen Projekten, wo mehrere Klassen notwendig sind (wie z.B. bei der Server- und Clientkommunikation), fällt die Zeit meist nicht so viel ins Gewicht.

Andere benutzerdefinierte Packages

Andere benutzerdefinierte Packages werden mittels der import-Anweisung angesprochen und wie die API verwendet. Verwendung von anderen Packages verlangsamen genauso wie die anderen Klassen den Ablauf.

Extended Packages

Manche extended Packages sind plattformspezifische Erweiterungen der Java Card API 2.0 z.B. die GSM Lib. Es ist zur Zeit kein sinnvoller Benchmark dieser komplexen Funktionen möglich.

Modifikatoren

Benutzerdefinierte Methoden und Variablen können public, private oder protected (wie in Java) sein.

4.3. Programmierrichtlinien für Benchmarking Java Card Applets

Loop Funktion

Unter der Loop Funktion versteht man, dass eine benutzerdefinierte Methode mehrmals hintereinander abgearbeitet wird. Die Anzahl der Loops ist durch das Skript steuerbar. Man kann z.B. eine Methode zuerst nur ein Mal laufen lassen, um sie dann acht Mal laufen zu lassen, dann 16 Mal usw. Man entscheidet sich bewusst für Loop Funktionen, um die Kommunikationszeit zwischen Karte und Terminal zu ermitteln. Bei nur einem Loop hat man die Kommunikationszeit (Faktor 1) und die Abarbeitungszeit (Faktor 1). Bei z.B. 4096 Loops hat man immer noch die Kommunikationszeit (Faktor 1) und eine ziemlich große Abarbeitungszeit (Faktor 4096). Je mehr Loops man hat, umso geringer ist die Kommunikationszeit im Verhältnis zur Abarbeitungszeit. Bei sehr vielen Loops konvergiert schließlich die Kommunikationszeit gegen null. Die gesamte Abarbeitungszeit wird dann durch die Anzahl der Loops geteilt, womit man die Abarbeitungszeit für eine einmalige Ausführung bestimmt.
Um die Wiederholungen zu ermöglichen, müssen die Input-Stellen aus dem Skript, im Quellcode definiert werden. Das geschieht wie folgt:

```
final static byte LOOP1 = (byte) ISO7816.OFFSET_INS+5;
final static byte LOOP2 = (byte) ISO7816.OFFSET_INS+6;
final static byte LOOP3 = (byte) ISO7816.OFFSET_INS+7;
```

D.h. die 5., 6. und 7. Stelle im APDU Kommando im Skript des folgenden Beispiels ist mit hexadezimalen Zahlen vorbelegt (rot markiert). Die Zahlen werden anschließend multipliziert und das Resultat ergibt die Anzahl der Methodenwiederholungen (Loops). Die maximale Anzahl der Loops ist 255^3 also 16 581 375. Drei Stellen (5-, 6- und 7-te Stelle) stehen zur Verfügung, eine Stelle beinhaltet zwei hexadezimale Zahlen, also maximal FF für eine Stelle (255 dezimal). Da es drei Stellen gibt, ergibt 255*255*255, also 255^3.
Die Multiplikation der Loops geschieht im Quellcode im *for* Block in den jeweiligen Methoden.

Die entsprechende Kommandozeile im Skript sieht so aus:

Beispiel:
0x90 0x01 0x00 0x00 0x03 0x02 0x02 0x02 0x04;

Im Beispiel unten wird ein 256 großes Array vom EEPROM ab der Adresse 5 ins RAM ab der Adresse 8 kopiert. Diese Anweisung wird (dem Skript zu Folge) 8-mal durchgeführt. Die Quell- und Zieladressen ändern sich bei den acht Wiederholungen nicht.

Beispiel:

```
…
 public class Appletname extends Applet {
     final static byte LOOP1 = (byte) ISO7816.OFFSET_INS+5;
     final static byte LOOP2 = (byte) ISO7816.OFFSET_INS+6;
     final static byte LOOP3 = (byte) ISO7816.OFFSET_INS+7;
     …
for (i1=0;i1<buffer[LOOP1];i1++){
    for (i2=0;i2<buffer[LOOP2];i2++){
        for (i3=0;i3<buffer[LOOP3];i3++) {
            Util.arrayCopyNonAtomic(EEdata1,(short)5,RAMdata1,(short)8,(short)256);
            // Anweisungsabarbeitung
         }
    }
 }
```

Dummy_Null Funktion

Ein anderes Mittel, um die Kommunikationszeit zwischen Karte und Terminal zu ermitteln, ist eine Dummy_Null Funktion in den Quellcode einzubauen und deren Laufzeit zu testen.
Die Dummy_Null Funktion enthält drei Loops, die keine Anweisung beinhalten. Es findet nur die Kommunikation zwischen der Karte und dem Terminal statt. Daraus kann man die Kommunikationszeit ermitteln, da die Abarbeitungszeit null ist.

Beispiel:

```
for (i1=0;i1<buffer[LOOP1];i1++){
    for (i2=0;i2<buffer[LOOP2];i2++){
        for (i3=0;i3<buffer[LOOP3];i3++) {
            … // keine Anweisungsabarbeitung
         }
    }
 }
```

Delta T-Funktion

Diese Methode enthält zwei Funktionen, die Loop Funktion (1) und Dummy_Null Funktion (2). Wenn man die gesamte Abarbeitungszeit der Loop Funktion von der gesamten Abarbeitungszeit der Dummy_Null Funktion abzieht, bekommt man nur eine Abarbeitungszeit für die jeweilige Anweisung.

Beispiel:
(1)
Get Data from Terminal ()
```
for (i1=0;i1<buffer[LOOP1];i1++){
    for (i2=0;i2<buffer[LOOP2];i2++){
        for (i3=0;i3<buffer[LOOP3];i3++) {
            … // Anweisungsabarbeitung
        }
    }
}
```
Send Data to Terminal ()

(2)
Get Data from Terminal ()
```
for (i1=0;i1<buffer[LOOP1];i1++){
    for (i2=0;i2<buffer[LOOP2];i2++){
        for (i3=0;i3<buffer[LOOP3];i3++) {
            … // keine Anweisungsabarbeitung
        }
    }
}
```
Send Data to Terminal ()

Kaskadierte Instruktionen (Instruction Cascading)

Eine kaskadierte Instruktion beinhaltet einen Anweisungsblock. Der Anweisungsblock beinhaltet mehrere gleich aufgebaute Anweisungen, die hintereinander abgearbeitet werden.
Im Beispiel werden 256 lange Arrays des Typs short vom EEPROM ins RAM kopiert. Die Quell- und Zieladressen sind bei jeder Anweisung anders.

Beispiel:
```
Util.arrayCopyNonAtomic(EEdata1,(short)0,RAMdata1,(short)1,(short)256);
Util.arrayCopyNonAtomic(EEdata1,(short)1,RAMdata1,(short)2,(short)256);
Util.arrayCopyNonAtomic(EEdata1,(short)2,RAMdata1,(short)3,(short)256);
Util.arrayCopyNonAtomic(EEdata1,(short)3,RAMdata1,(short)4,(short)256);
Util.arrayCopyNonAtomic(EEdata1,(short)4,RAMdata1,(short)5,(short)256);
Util.arrayCopyNonAtomic(EEdata1,(short)5,RAMdata1,(short)6,(short)256);
Util.arrayCopyNonAtomic(EEdata1,(short)6,RAMdata1,(short)7,(short)256);
Util.arrayCopyNonAtomic(EEdata1,(short)7,RAMdata1,(short)8,(short)256);
```

Die kaskadierte Instruktion kann durch eine Loop Funktion ersetzt werden. Die kaskadierte Instruktion verhindert allerdings das Caching der Java Card Virtual Machine (genauer beschrieben in Kapitel 6.1).

Wrapper Class

Der Vollständigkeit halber soll hier eine Klasse, die Wrapper Class genannt werden, die bislang noch auf theoretischen Überlegungen basiert, d.h., die noch nicht praktisch angewendet wird. Die Idee ist, dass ein Applet (die Wrapper Class) mehrere andere Applets (Testklassen) beinhaltet, die durch die Wrapper Class in einem Zug automatisch abgearbeitet werden, so dass das separate Laden der einzelnen Testklassen entfällt (siehe Abbildung 4.3.1).
Falls eine der Testklassen nicht bearbeitet werden kann, wird sie ausgelassen, ohne die anderen Testklassen zu beeinflussen, oder sogar die ganze Wrapper Class zu stoppen. Die Kommunikation zwischen Wrapper Class und Testklassen könnte über das Shareable Interface stattfinden, da alle Klassen, die dieses Interface realisieren, nur diejenigen Methoden zum sharen freigibt, die durch dieses Interface vorgegeben werden. Somit wird die Firewall der Applets umgangen.

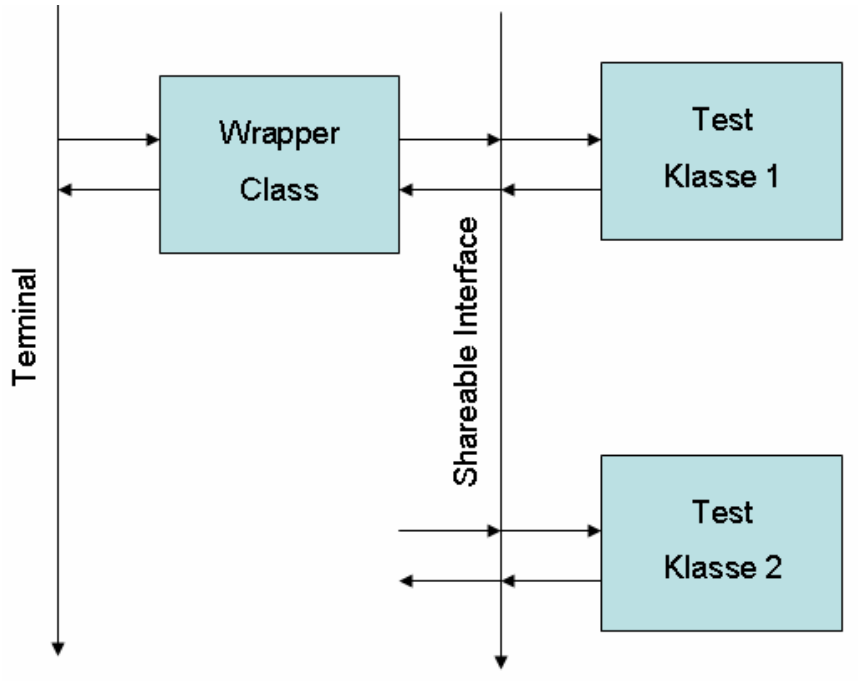


Abbildung 4.3.1 Wrapper Class

Dummy Funktion

Die Dummy Funktion ist eine andere Funktion als die Dummy_Null Funktion, auch wenn die Namen ganz ähnlich sind. Die Dummy Funktion kann, muss aber nicht zwingend so aufgebaut sein wie eine Dummy_Null Funktion. Die beiden Funktionen werden für unterschiedliche Zwecke benutzt. Die Dummy Funktion wird zuerst aufgerufen, also bevor der Test Code aufgerufen wird (siehe Abbildung 4.3.2). Die Aufgabe dieser Funktion besteht darin, die Kommunikation zwischen dem Terminal und dem Java Card Betriebssystem zu synchronisieren. D.h. alle Initialisierungen finden hier statt. So werden alle danach folgenden Funktionen ohne zusätzlichen Zeitverlust für die Initialisierung bearbeitet. Das garantiert, dass es keine Verzögerungen gibt. Die Dummy_Null Funktion dagegen soll die reine Abarbeitungszeit ermitteln und wird daher am häufigsten zuletzt aufgerufen.

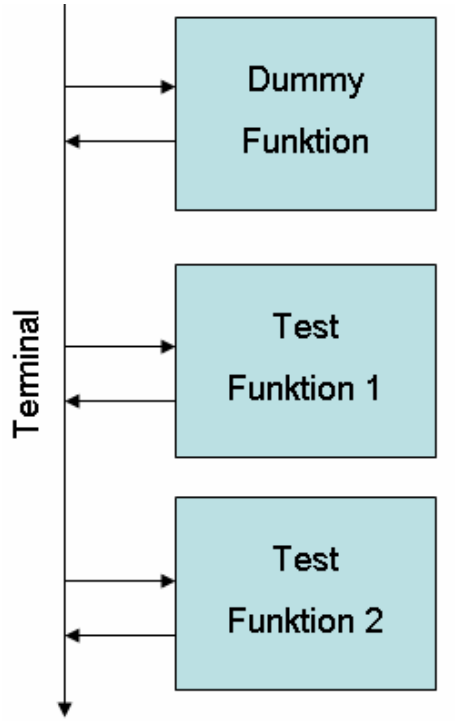


Abbildung 4.3.2 Dummy Funktion

5. Messungen

Nachdem nun die Vorraussetzungen geschaffen und präsentiert worden sind, wie die Messungen durchzuführen sind, um sinnvolle, vergleichbare Ergebnisse zu erhalten, soll in diesem Kapitel hauptsächlich der Messplatzaufbau beschrieben werden

5.1. Messplatzbeschreibung

Die Tabelle 5.1.1 zeigt eine Übersicht der verwendeten Komponenten zur Entwicklung und zum Testen von Java Card Applets.

Messsystem	Rechner	HW	BS	IDE	Kartenleser
Messsystem 1	A1	800 MHz, 512 MB RAM	Windows 2000	Aspects	Towitoko seriell, Omnikey USB
Messsystem 2	Laptop	656 MHz, 512 MB RAM	Windows 2000	Aspects	Towitoko seriell, Omnikey USB
Messsystem 3	Laptop	656 MHz, 512 MB RAM	Windows 2000	SCard	Towitoko seriell, Omnikey USB
-	Netzrechner	330 MHz, 192 MB RAM	Windows 2000	-	-

Tabelle 5.1.1 Vorhandene Testrechner und deren Eigenschaften

Der Messplatz bestand aus insgesamt vier Rechnern. Wie die Tabelle 5.1.1 zeigt, wurden drei davon nur für die Entwicklung und zum Testen der Java Card Applets benutzt. Als Betriebssystem war Windows 2000 installiert. An diesen Rechnern waren verschiedene Kartenleser angeschlossen, wie z. B. der Kartenleser von Towitoko mit serieller Schnittstelle oder der Kartenleser von Omnikey mit USB Schnittstelle. Auf jedem Rechner war eine Entwicklungsumgebung (siehe dazu Kapitel 2.11) installiert, wie z.B. Aspects Developer, eine Entwicklungsumgebung der englischen Firma Aspects, mit der man eine Vielzahl von Java Cards testen kann

und SCard, ein Eigenentwicklungsprogramm der Firma Infineon Technologies, mit dem man alle Karten testen kann. Man hatte absichtlich Aspects Developer auf zwei Rechner installiert, um zu zeigen, dass es keinen Unterschied gab, wenn man ein und dasselbe Applet auf zwei unterschiedlich schnellen Rechnern getestet hat. Die Ergebnisse haben gezeigt, dass bei einer hinreichend performanten CPU (ab 500 MHz) ein Einfluss der verwendeten Computerhardware zu vernachlässigen ist.
Die Hardwareausstattung der Rechner war unterschiedlich. So lag z. B. der Hauptspeicher (RAM) zwischen 195 - 512 MB und die Taktfrequenz der CPU zwischen 330-800 MHz. Die einzelnen Entwicklungssoftwarepakete wurden auf separaten PCs installiert, um eine Beeinflussung der Software durch andere, einzelne Komponenten zu vermeiden. Dieser Messplatzaufbau hatte vor allem die Vorteile, dass z.B. Störeffekte durch eine automatische Pfadsetzung oder durch die Verwendung unterschiedlicher Versionsnummern der Java Card Virtual Machine bei den PCs ausgeschlossen werden konnte. Auf dem vierten Rechner wurden alle Messungen dokumentiert, die Testergebnisse in Excel-Tabellen eingetragen und ausgewertet. Dieser Rechner hatte zudem einen Internetzugang für Recherchen und das Versenden von Emails.

Alle Messungen sollten unabhängig von Messprogrammen, Kartenlesern, Betriebssystemen und Rechnereigenschaften sein und gleiche Ergebnisse für ein und dieselbe Karte liefern. Um das zu zeigen, wurden vor allem, wie in Kapitel 6 beschrieben, mehrere Beweismessungen durchgeführt. Es wurden sowohl Messungen mit mehreren Kartenlesern mit gleichem Messsystem als auch Messungen mit verschiedenen Messprogrammen und mit gleichen Kartenlesern durchgeführt. Es wurden auch Messungen gemacht, bei denen eine Karte mit verschiedenen Messprogrammen und mit verschiedenen Kartenlesern getestet wurde. All diese Kombinationen haben vergleichbare Resultate geliefert. Aufgrund dieser Messungen konnte man feststellen, dass sowohl die Kartenleser als auch die Messsysteme auf die Ergebnisse keinen Einfluss haben.

Die Abbildung 5.1.1 zeigt die Zeitdifferenzen bei der Ausführung eines Applets mit gleichen Kommandos bei zwei Rechnern (A1 und Laptop mit Aspects) mit unterschiedlichen Betriebssystemen und Rechnereigenschaften. Es hat sich herausgestellt, dass fast 38% der Zeitunterschiede bei der Ausführung von Kommandos eine Abweichung von +5 msec aufweisen. Um die 90% aller Zeitunterschiede bewegen sich zwischen ±15 msec. Maximal lagen die Zeitunterschiede bei ±25 msec. Daher wurde für die gesamten, späteren Messungen die Fehlerdifferenz von 50 msec gewählt. Dieser Wert wurde hinreichend spezifiziert und nicht mehr in Betracht gezogen.

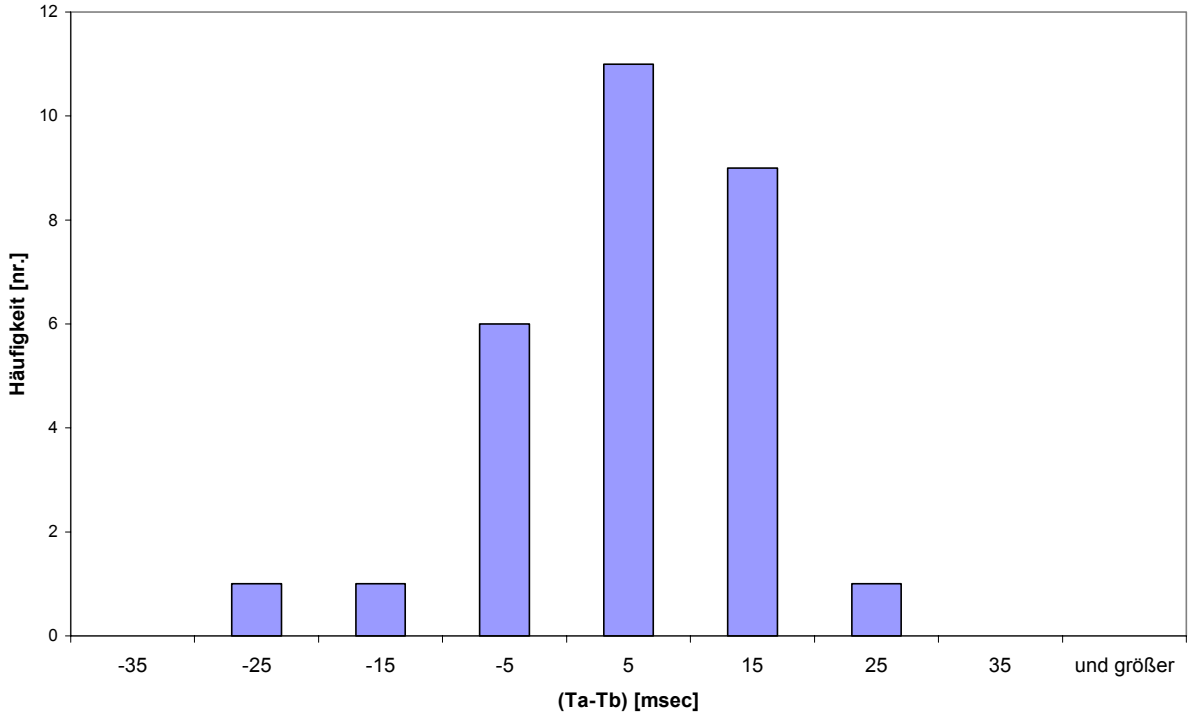


Abbildung 5.1.1 Die Zeitdifferenzen bei unterschiedlichen Betriebssystemen und Rechnereigenschaften

5.2. Zeitmessung

In Abbildung 5.2.1 wird der gesamte Kommunikationsablauf dargestellt. Auf einem Rechner ist eine Entwicklungsumgebung installiert. Mit dem Programm wird ein Applet auf die Karte geladen. Aus den verschiedenen Applets, die auf der Karte geladen sind, wird ein Applet ausgewählt. Eine bestimmte Funktion (Kommando) wird ausgeführt. Die Zeitspanne Time_1 besteht aus folgenden Komponenten: ein bestimmtes Kommando (z.B. das Addieren von zwei Zahlen) wird in Form eines APDU Kommandos über die PC/SC Schnittstelle zum Kartenleser geschickt. Hierzu wird ein Timer-Flag_1 in der PC Software unmittelbar vor dem Abschicken des APDU gesetzt. Über die PC/SC Treiber Software wird das Kommando an den Kartenleser übermittelt. Das APDU Kommando erreicht im nächsten Abschnitt die Karte und wird dort ausgeführt. Das APDU wird von der Karte interpretiert und die entsprechenden Codesequenzen werden aufgerufen. Erst dann finden z.B. in der CPU die eigentlichen Berechnungen (Addiere zwei Zahlen) statt. Das entspricht dem Zeitfenster Time_5. Das Resultat wird in einem Antwort APDU kodiert und zum Kartenleser zurückgeschickt. Über die PC/SC Schnittstelle gelangt die Nachricht zur Entwicklungsumgebung. Die Entwicklungsumgebung setzt Timer-Flag_2 und blendet die Antwort mit der dazugehörigen Zeitdifferenz (Time_1) ein. Alle durchgeführten Messungen haben den Zeitwert Time_1. Die gesamte Kommunikation für das Benchmarking von Java Card ist jedoch nicht von Interesse.

Für das Benchmarking von Java Card ist Time_5 der zu untersuchende Parameter. Man versucht also die Kommunikationszeit auszuschließen. Das wird mit Hilfe der Loop Funktion, der Dummy Funktion usw. (siehe auch Kapitel 4.3) erreicht.

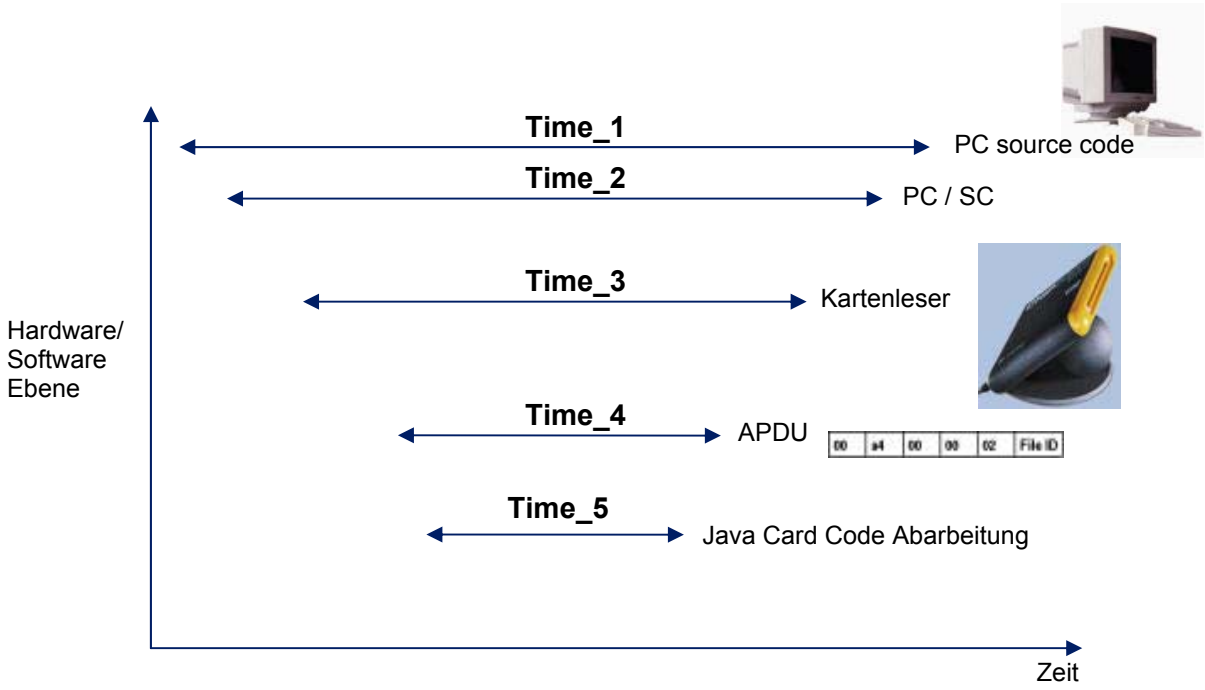


Abbildung 5.2.1 Zeitmessung

6. Ergebnisse des Java Card Benchmarking

Im Rahmen dieser Arbeit wurden im Kapitel 6 die vorhandenen Karten mit drei Messsystemen und mit zwei Kartenlesern, nämlich Towitoko (abgekürzt T) und Omnikey (abgekürzt O) untersucht. Die drei Messsysteme (siehe auch Tabelle 5.1.1) sind:

- Ein Rechner A1 mit installiertem Aspects Developer und mit zwei angeschlossenen Kartenlesern; die Bezeichnungen für dieses Messsystem sind (T), wenn der Towitoko Kartenleser angeschlossen und (O), wenn der Omnikey Kartenleser angeschlossen ist.
- Ein Laptop mit installiertem Aspects Developer und mit zwei angeschlossenen Kartenlesern; die Bezeichnungen für dieses Messsystem sind (T^2), wenn der Towitoko Kartenleser angeschlossen und (O^2), wenn der Omnikey Kartenleser angeschlossen ist.
- Ein Laptop mit dem Programm SCard und mit zwei angeschlossenen Kartenlesern; die Bezeichnungen für dieses Messsystem sind (T^3), wenn der Towitoko Kartenleser angeschlossen und (O^3), wenn der Omnikey Kartenleser angeschlossen ist.

Durch die Variation der verwendeten Messumgebungen, d.h. durch die Variation der Ansteuersoftware und der Chipkartenleserhardware wurde die Unabhängigkeit der Benchmarking Ergebnisse der Messumgebungen nachgewiesen. Ein Applet, das auf einer bestimmten Karte mit verschiedenen Messsystemen und mit verschiedenen Kartenlesern getestet wurde, sollte somit immer gleiche Ergebnisse liefern.

Im folgenden Abschnitt werden die im Rahmen der Diplomarbeit erstellten Java Cards Applets beschrieben. Die erhaltenen Messergebnisse werden dokumentiert und ausgewertet.

Die im Kapitel 3 beschriebene Metrik soll hier angewendet werden. Die einzelnen Namenspfade auf einem Spinnennetz werden abgearbeitet. Für Schreib- und Lesezugriffe werden das Eeprom_Test Applet und das Eeprom_Test_FB Applet getestet. Für den Namenspfad Sicherheit wird das DES Applet und für den Namenspfad Java Card Sprachelemente wird das Control_Logik Applet getestet. Die restlichen Namenspfade (Appletmethoden und Objektmethoden, Arrays, Operatoren, Java Card System, Visa Open Platform) wurden im Rahmen dieser Diplomarbeit nicht abgearbeitet.

6.1. Hardware

Die nachfolgenden Applets liefern die Ergebnisse für den Namenspfad Schreib- und Lesezugriffe.

Eeprom_Test Applet

Mit diesem Applet werden die Hardwarekomponenten, das EEPROM und das RAM getestet. Hier werden die Schreib- und Lesezugriffe auf dem EEPROM und dem RAM untersucht. Mit Hilfe dieses Applets sollen, wenn möglich, auch die Seitengröße und die Größe des I/O Buffers (siehe Abbildung 6.1.1) ermittelt werden.

Das Modell eines Datenspeichers beruht auf folgenden Annahmen:

- die Daten [0 bis x Byte] werden über den Datenbus in den I/O Buffer geschrieben
- aus dem Daten-Buffer werden sie in eine zeilen/spalten-orientierte Matrix geschrieben, jede Zelle entspricht 1 Bit
- Daten werden zeilenweise ins Zellenfeld geschrieben
- die Anzahl der geschriebenen [0 bis y] Bits bestimmt die Seitengröße (Pagegröße), d.h. die Seitengröße wird durch das physikalische Layout des Speichermoduls bestimmt
- die anderen Basiskomponenten (HV Control, Timing Control, Adressbus, Dekoder) werden in diesem Modell nicht berücksichtigt

- **Annahme:** 1) I/O Buffergröße = Seitengröße (ist technisch gesehen naheliegend)
  2) Die Programmierzeit hat einen festen Wert, der durch physikalische Eigenschaften der Zelle bestimmt wird.

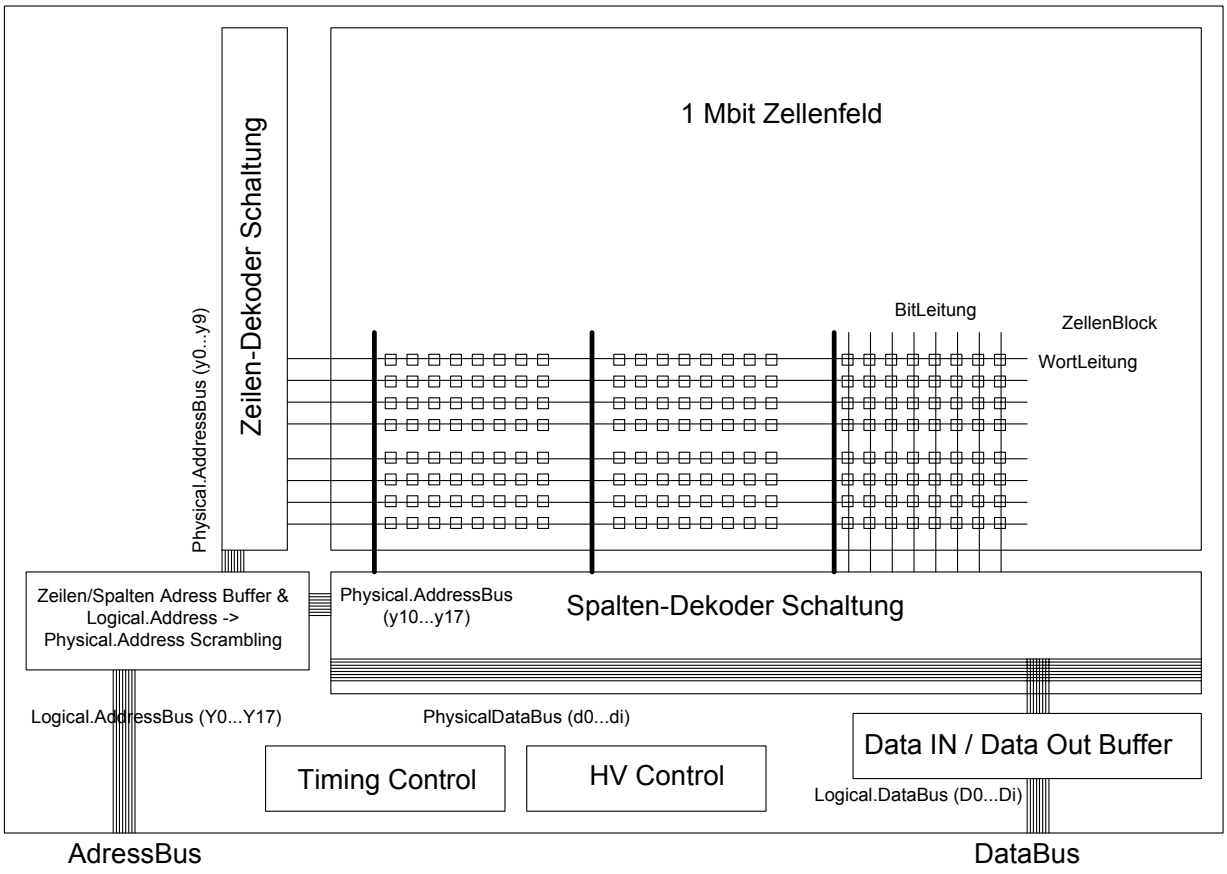


Abbildung 6.1.1 Hardwaremodell eines Datenspeichers [private Sammlung von B. Lippmann]

Wie bereits in Kapitel 4.2 beschrieben, kann man die Datenstruktur im EEPROM als globale Variable initialisieren, wohingegen transiente Variablen im RAM der Chipkarte erstellt werden.

Das Eeprom_Test Applet beinhaltet drei Methoden: EEPROM-EEPROM, RAM-EEPROM und Dummy_Null (siehe dazu auch Kapitel 4.3). Die erste Methode kopiert ein Array vom EEPROM ins EEPROM, und die zweite Methode kopiert ein Array vom RAM ins EEPROM. Die Methode Dummy_Null soll die Kommunikationszeit bestimmen. Sie besteht nur aus einem Rumpf. Die Methode wird über das APDU an eine Karte geschickt, und da in diesem Fall keine Berechnung stattfindet, sendet die Karte gleich eine Antwort zurück.
Das für die zwei ersten Methoden benötigte Array beinhaltet eine variable Anzahl an Bytes (4, 8, 16 und 32). Wie viele Bytes gesendet werden, bestimmt eine Variable, die bei jedem Kommando den Wert vom Skript übernimmt. Die jeweiligen Kommandos werden 1-, 8- und 64-mal wiederholt. Auch die Anzahl der Wiederholungen (Loops) sind aus dem dazugehörigen Skript zu übernehmen.

Es wurden folgende Karten mit verschiedenen Messsystemen und mit verschiedenen Kartenlesern getestet:

- Mit dem ersten Messsystem und mit zwei Kartenlesern wurden die Karten A, C, E, F, H und I getestet. Karte G kann man nicht auf diesem Messsystem testen, da nicht alle Parameter dieser Karte frei verfügbar sind. Somit kann diese Karte vom Messsystem nicht angesprochen werden.
- Mit dem zweiten Messsystem wurden die Karte E und die Karte H mit beiden Kartenlesern und die Karte A mit dem Towitoko Kartenleser getestet.
- Mit dem dritten Messsystem wurden die Karte G mit beiden Kartenlesern und die Karte E mit dem Towitoko Kartenleser getestet.

(1) Das Eeprom_Test Applet wurde auf oben genannten Karten mit drei verschiedenen Messsystemen und mit zwei verschiedenen Kartenlesern getestet.
Die Abbildung 6.1.2 zeigt einen Ausschnitt der Messungen. Es wurden zwei Kommandos, nämlich EEPROM-EEPROM und RAM-EEPROM aufgezeichnet. Diese Kommandos kopieren 32 Byte vom EEPROM oder vom RAM auf das EEPROM. Die beiden Methoden wurden 64-mal wiederholt. Die restlichen Kommandos (also alle möglichen Kombinationen für das Kopieren von 4, 8, 16 und 32 Bytes, mit 1, 8 und 64 Loops) liefern äquivalente Ergebnisse.
Die Gesamtzeit für die ausgewählten Kommandos kann man aus der Abbildung 6.1.2 ablesen. Die beste Karte in diesem Vergleich ist die Karte E. Mit drei Messsystemen und mit zwei Kartenlesern erreicht sie sowohl bei dem Kommando EEPROM-EEPROM (immer 160 msec) als auch bei dem Kommando RAM-EEPROM (die Zeit liegt zwischen 130 und 150 msec) die schnellste Ausführungszeit. Die Karte braucht für das Kommando EEPROM-EEPROM maximal 30 msec mehr als für das Kommando RAM-EEPROM. Zu den mittleren Karten gehören die Karten A, C, H und I. Deren Zeiten liegen bei dem Kommando EEPROM-EEPROM zwischen 290 und 591 msec. Die Zeiten bei dem Kommando RAM-EEPROM liegen zwischen 190 und 591 msec. Bei den Karten A und C sieht man, dass die Zeit, die die Karten für das Kopieren vom EEPROM auf das EEPROM brauchen, deutlich größer ist als die Zeiten für das Kopieren vom RAM auf das EEPROM. Bei den übrigen Karten ist der Unterschied nicht so deutlich. Die Karten G und F waren in dem Vergleich am langsamsten. Sie brauchten zwischen 661 und 812 msec bei dem Kommando EEPROM-EEPROM und zwischen 651 und 811 msec bei dem Kommando RAM-EEPROM. Diese zwei Karten haben fünfmal mehr Zeit als die Karte E gebraucht.

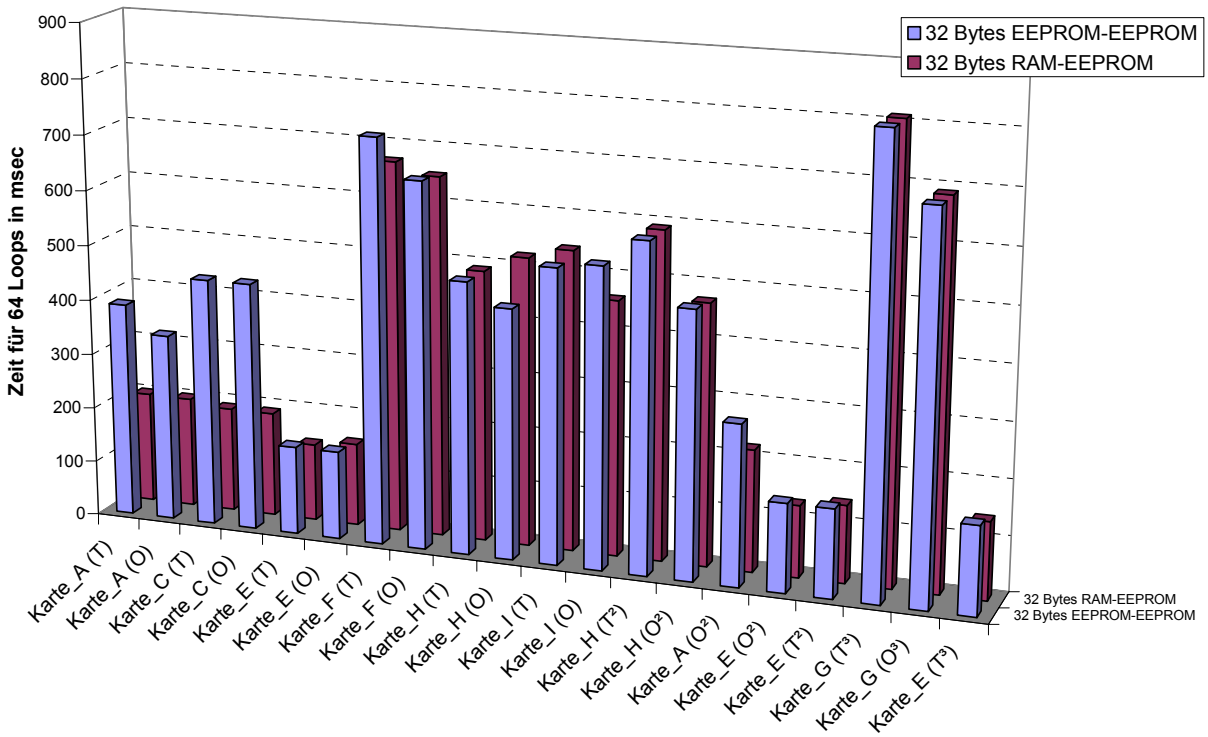


Abbildung 6.1.2 Das Eeprom_Test Applet, Ergebnisse aller Karten

Aufgrund der unterschiedlichen Zugriffszeiten der Speichermodule EEPROM und RAM erwartet man, dass das Kopieren vom EEPROM auf das EEPROM länger dauert als das Kopieren vom RAM auf das EEPROM. Gemäß diesen Erwartungen haben sich aber nur die Karten A, C und H verhalten. Bei der Karte E sind keine Zeitunterschiede zu sehen. Es war also nicht von Bedeutung, ob man die Daten vom RAM oder vom EEPROM auf das EEPROM kopiert. Bei den Karten F, H, I und G sind die Zeitunterschiede zwischen dem Kopieren vom EEPROM auf das EEPROM und vom RAM auf das EEPROM entweder gleich groß, oder das Kopieren vom EEPROM auf das EEPROM ist schneller um 50 msec als das Kopieren vom RAM auf das EEPROM, oder das Kopieren vom RAM auf das EEPROM ist um ca. 50 msec schneller als das Kopieren vom EEPROM auf das EEPROM. Dies entspricht einer angenommenen Fehlertoleranz von 50 msec. Bei diesen Karten kann man davon ausgehen, dass das Kopieren von Daten fast immer gleich lang dauert, unabhängig davon, ob man die Daten vom EEPROM oder vom RAM auf das EEPROM kopiert.

(2) Abbildung 6.1.3 zeigt wie eine Karte (Karte E), mit einem Towitoko Kartenleser und mit drei verschiedenen Messsystemen getestet wird. Hier wurden neun Kommandos getestet, wobei jeweils 4, 8, 16 und 32 Byte vom EEPROM bzw. vom RAM auf das EEPROM kopiert wurden. Zusätzlich wurde ein Dummy_Null Kommando ausgeführt. All diese Kommandos wurden nur einmal wiederholt.

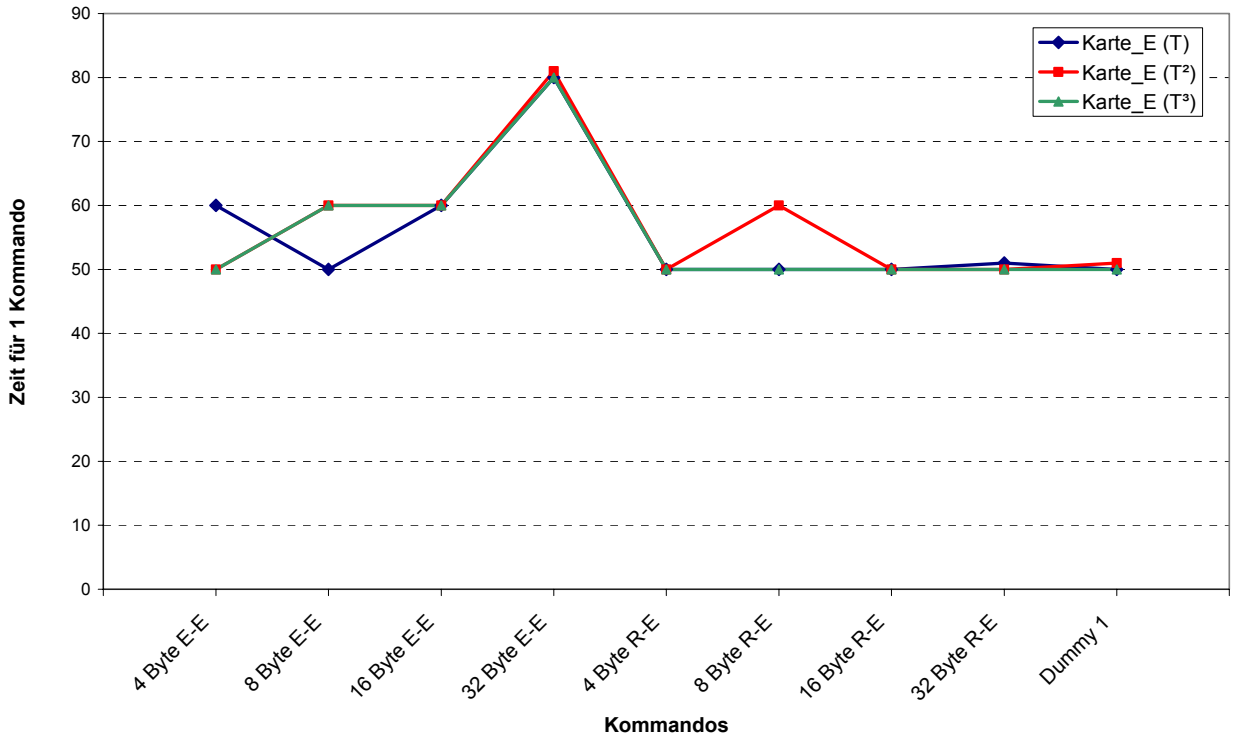


Abbildung 6.1.3 Eeprom_Test Applet, eine Karte, ein Kartenleser, drei Messsysteme

Diese Messungen haben ergeben, dass es keinen Unterschied zwischen den drei Messsystemen, die eine verschiedene Softwareausstattung haben, gibt. Die gemessenen Zeitunterschiede liegen nur bei 10 msec (siehe Kapitel 5.1).

(3) In Abbildung 6.1.4 wird eine Karte (Karte E) mit zwei verschiedenen Kartenlesern (Towitoko und Omnikey) mit einem Messsystem getestet. Alle Kommandos (EEPROM-EEPROM mit 4, 8, 16 und 32 Bytes, RAM-EEPROM mit 4, 8, 16 und 32 Bytes, Dummy_Null) wurden jeweils 1-, 8- und 64-mal wiederholt und ausgewertet.

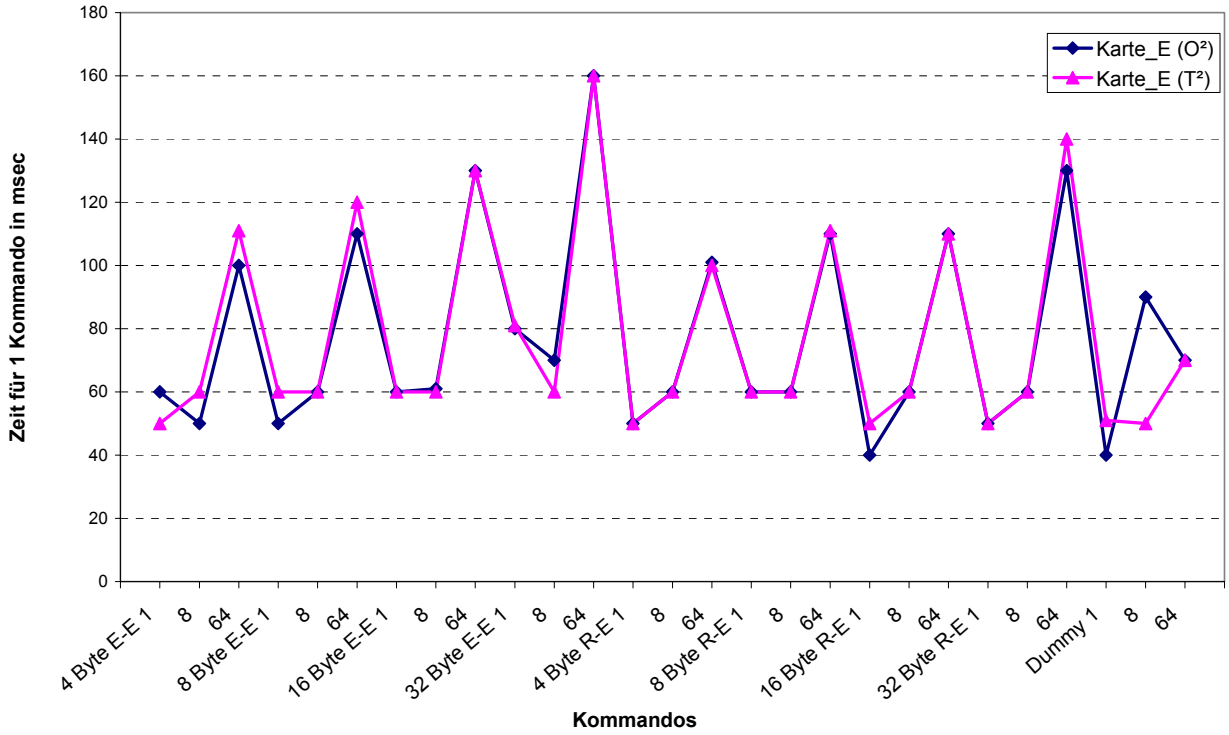


Abbildung 6.1.4 Eeprom_Test Applet, eine Karte, zwei Kartenleser, ein Messsystem

Diese Messungen haben die Unabhängigkeit der Kartenleser, die mit einem Messsystem getestet wurden, gezeigt. Wie man sehen kann, liegen die Zeitunterschiede bei 10 msec, oder wie in einem Fall bei 40 msec.

Die Messungen in Abbildung 6.1.3 und Abbildung 6.1.4 belegen somit die bereits in Kapitel 5.1 beschriebene Unabhängigkeit von Software- und Hardwarekomponenten.

(4) Ziel der nachfolgenden Messungen war es, die Seitengröße (siehe Abbildung 6.1.1) zu ermitteln. Es wurden Bytes (von 4 bis 124 Bytes) 1-, 8-, 16, 32-, 64-, 128-, 256-, 496-, 992-, 1984 und 4096-mal vom EEPROM auf das EEPROM hintereinander kopiert. Alle verfügbaren Karten wurden getestet.

Abbildung 6.1.5 zeigt die Ergebnisse bei der Karte E. Bei 8 und 4096 Loops wurde ein Effekt beobachtet. Man hat gesehen, dass bei 8 Loops eine Stufenfunktion deutlich zu sehen ist (Stufen sind mit Pfeilen markiert). Bei steigender Anzahl von Loops wird die Stufenfunktion flacher und schließlich bei 4096 Loops fast linear. Das bedeutet, dass auf diese Weise die Ermittlung der Größe einer EEPROM-Seite bei der Karte E nicht möglich war.

Unter gleichen Voraussetzungen wurde auch die Karte I getestet. Wie die Abbildung 6.1.6 zeigt, bilden sich bei ansteigender Anzahl von Loops regelmäßige Stufen (Stufen sind mit Pfeilen markiert) der Größe 16 Byte. Besonderes bei 4096 Loops liegt die Vermutung nahe, dass die Größe einer EEPROM-Seite 16 Byte groß ist. Jeder fünfte Punkt der Gerade erweist eine Steigung. Da ein Punkt 4 Byte bezeichnet, ergibt das sechzehn Bytes.

Das beobachtete Absinken der Stufen konnte durch folgende Modellannahme erklärt werden. Das Caching der Java Card Virtual Machine verursacht, dass die Stufenfunktionen bei steigender Anzahl an Loops immer flacher werden. Die JCVM kopiert nicht mehrmals die gleiche Anzahl der Bytes an die gleiche Stelle. Sie überprüft lediglich, ob sich die zu kopierenden Daten an der erwünschten Stelle befinden. Wenn dies der Fall ist, hat es keinerlei Auswirkungen. Der Programmiervorgang findet nicht statt. Die Steigung ist daher proportional zum Array_Copy Befehl ohne Programmierzyklus. Deswegen konnte man anstatt einer Loop Funktion die Kaskadierte Instruktion benutzen (siehe Kapitel 4.3). Dadurch unterbindet man das Caching der Java Card Virtual Machine.

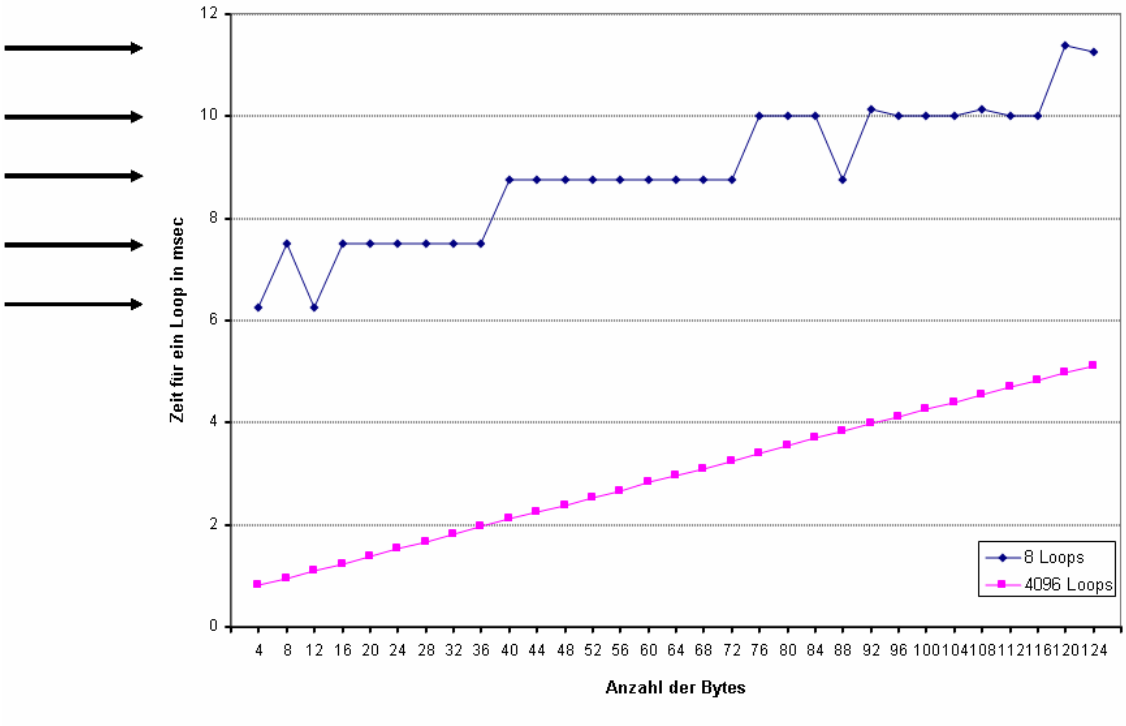


Abbildung 6.1.5 Eeprom_Test, Karte _E, EEPROM-EEPROM

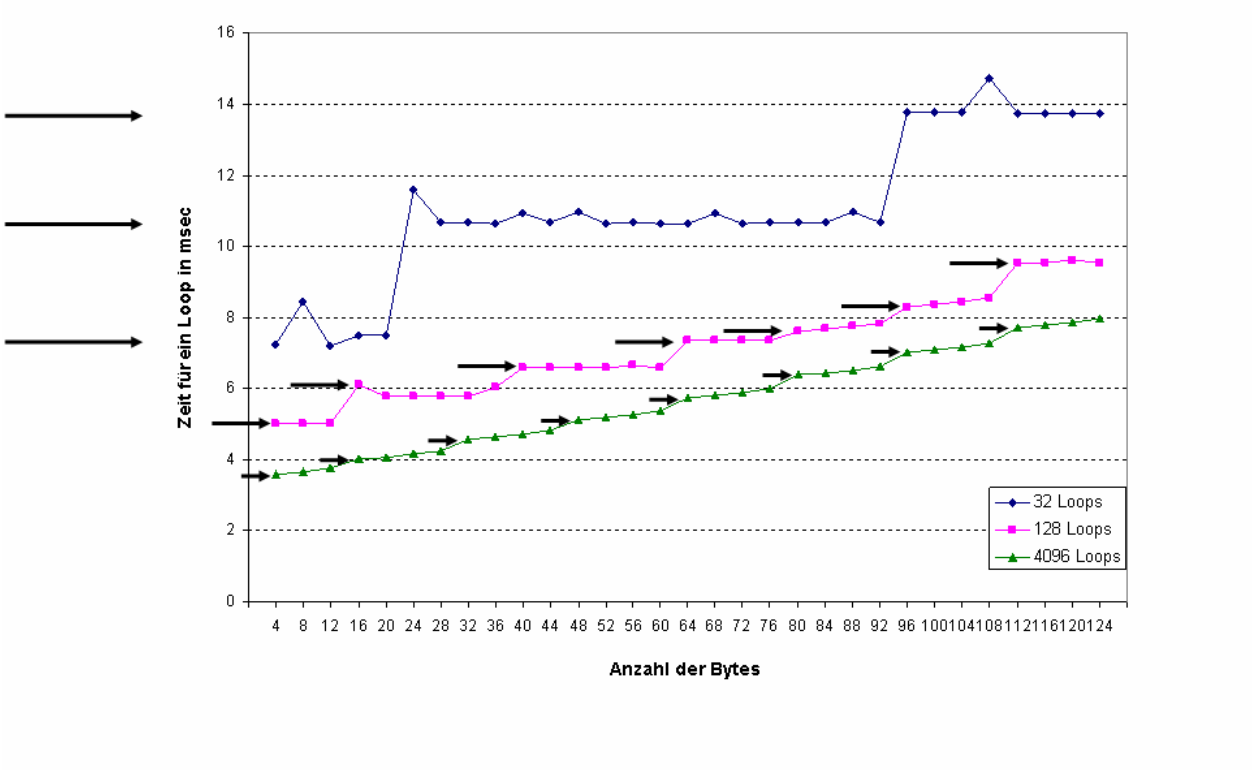


Abbildung 6.1.6 Eeprom_Test, Karte _I, EEPROM-EEPROM

Aus diesen Messungen geht hervor, dass man nicht bei jeder Karte auf genannter Weise die Seitengröße ermitteln kann. Nur bei der Karte I konnte mit hoher Sicherheit festgestellt werden, dass die Seitengröße 16 Byte groß ist. Bei den anderen Karten haben die Messungen keine Hinweise auf die Seitengröße gegeben.

Eeprom_Test_FB Applet

Das Eeprom_Test_FB Applet unterscheidet sich vom vorherigen Applet in folgenden Punkten: die Bytelänge kann über 2 Bytes bestimmt werden, und die Anzahl der Loops ist anders gewählt. Die Arrays bestehen aus 1, 128, 256 oder 512 Bytes. Die Anzahl der Loops ist in diesem Applet 1, 8, 16 und 32. Die Methoden sind identisch zum vorherigen Applet.
Um das Problem des Cachings (wie es im Eeprom_Test aufgetreten ist) zu umgehen, wurde in diesem Applet der zweite Ansatz, nämlich das Kopieren von großen Arrays, verwendet. Das Caching findet im RAM statt, große Arrays können aber nicht im RAM gecachet werden. Man hat folgende Ergebnisse erhalten. Nur vier Karten (die Karten A, C, F und I) waren in der Lage große Arrays zu kopieren. Die restlichen Karten (die Karten E, G und H) konnten nicht getestet werden. Die Fehleranalyse ergab, dass es nicht möglich war, das Applet mit einem großen globalen Array auf diese Karten zu laden. Der Grund war vermutlich ein zu geringer Platz auf dem RAM oder/und dem EEPROM. Diese Vermutung wurde durch folgenden Versuch bestätigt. Im Applet wurden kleinere globale Arrays (128 bzw. 256 Byte) erzeugt.

Dieses leicht veränderte Applet konnte ohne Probleme auf die restliche Karten geladen und ausgeführt werden. Der Versuch noch größerer Array (1024 Bytes) auf die Karten zu laden und auszuführen ist bei allen Karten gescheitert.

(1) Abbildung 6.1.7 zeigt die Ausführungszeiten für sechs Kommandos. Die Kommandos wurden einmal wiederholt. Es wurden jeweils 128, 256 und 512 Bytes vom EEPROM auf das EEPROM oder vom RAM auf das EEPROM kopiert. Es wurden vier Karten mit einem Kartenleser (Omnikey) und mit einem Messsystem getestet.
Die Karte A braucht für das Kopieren vom EEPROM auf das EEPROM bei 128 Bytes 341 msec, bei 256 Bytes 621 msec und bei 512 Bytes 1162 msec. Die Ausführungszeit steigt je mehr Bytes vom EEPROM auf das EEPROM kopiert werden. Anders sieht es für das Kommando RAM auf EEPROM aus. Hier sind die Werte für das Kopieren von 128 Bytes 80 msec, für 256 Bytes 80 msec und für 512 Bytes 90 msec. Sehr ähnlich sieht die Situation bei der Karte C aus. Die Werte liegen für das Kommando EEPROM-EEPROM für 128 Bytes bei 370 msec, für 256 Bytes bei 580 msec und für 512 Bytes bei 1101 msec. Die Werte für das Kommando RAM-EEPROM sind entsprechend 90, 71, 91 msec.
Bei den Karten F und I steigen die Werte für beide Kommandos mit ansteigender Anzahl von Bytes ganz langsam an. Die Ausführungszeiten für die beiden Kommandos bei gleicher Anzahl von Bytes sind fast gleich. Es gibt also keinen großen Unterschied ob man bei den Karten F und I von EEPROM oder von RAM auf EEPROM die gleiche Anzahl von Bytes kopiert.

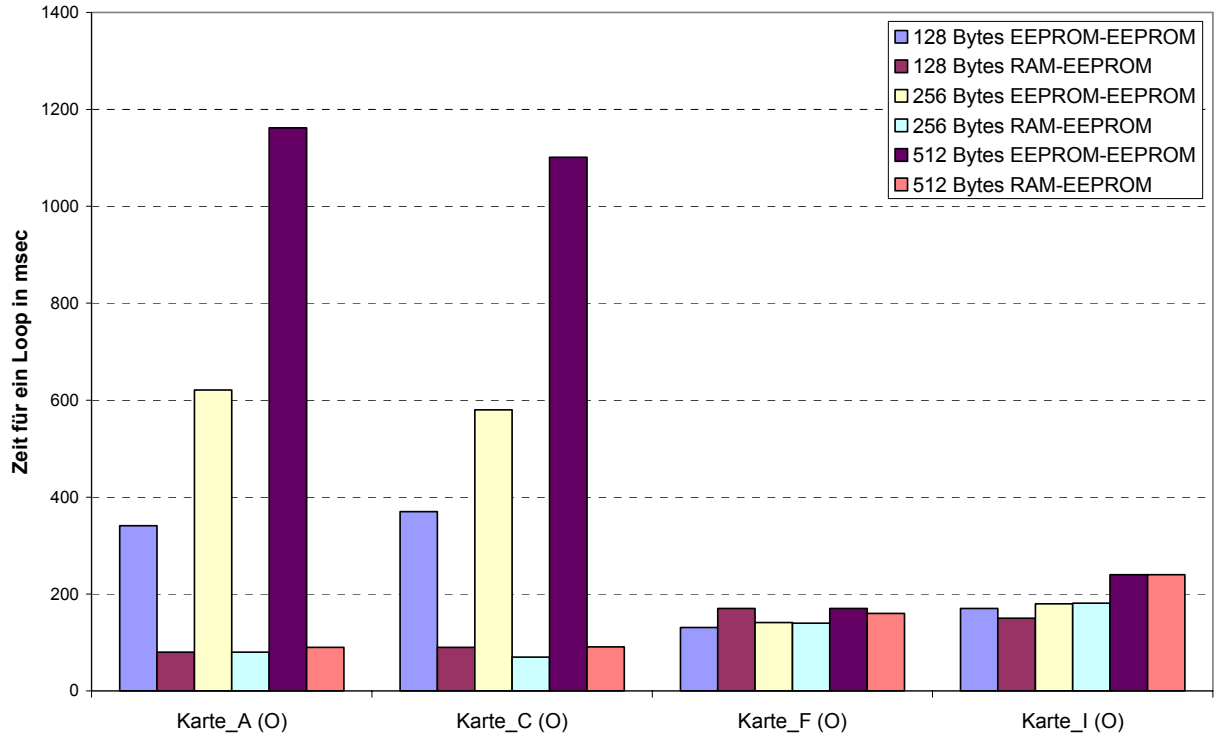


Abbildung 6.1.7 Eeprom_Test_FB Applet, Gesamtzeit bei einem Loop
(2) Die Abbildung 6.1.8 zeigt die gleichen Karten und die gleichen Kommandos wie Abbildung 6.1.7, aber die Kommandos wurden nicht einmal sondern 32-mal wiederholt. Hier sieht man deutlich, dass die Gesamtzeit mit ansteigender Anzahl von Bytes sowohl bei der Methode EEPROM-EEPROM als auch bei der Methode RAM-EEPROM steigt.

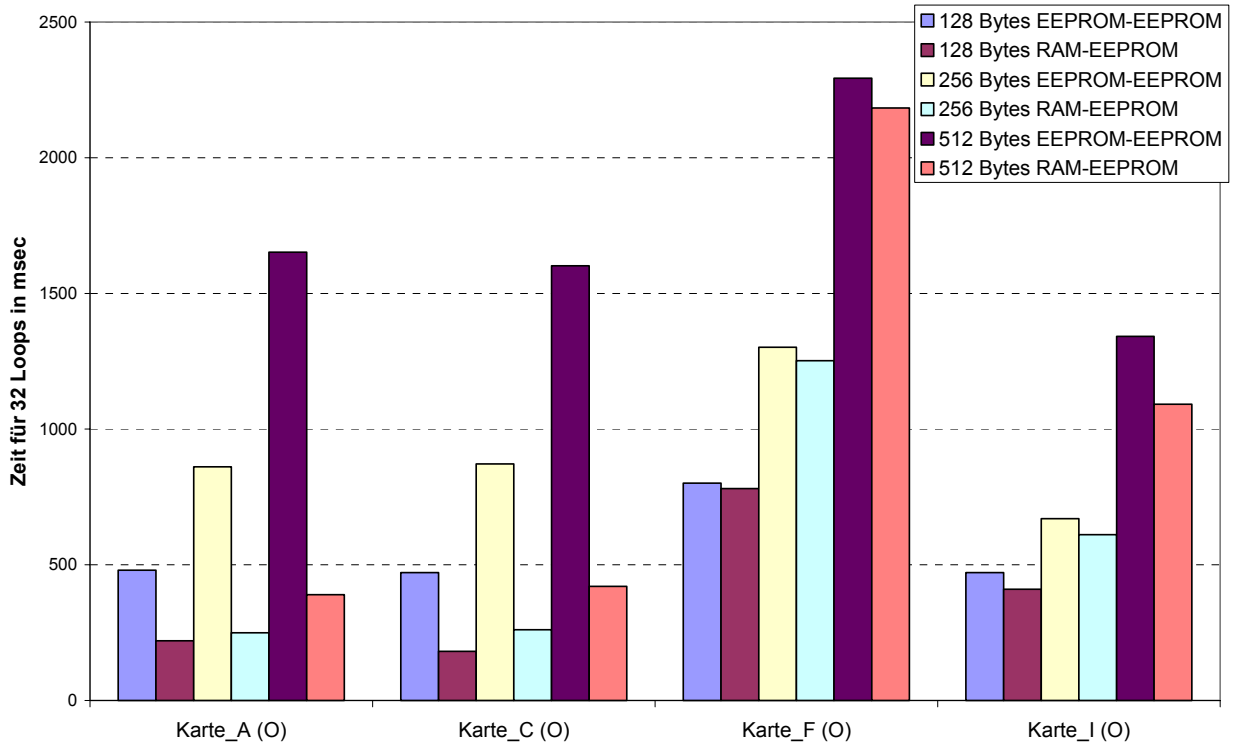


Abbildung 6.1.8 Eeprom_Test_FB Applet, Gesamtzeit bei 32 Loops

Das Eeprom_Test_FB Applet konnte nicht auf allen Karten getestet werden, da der vorhandene freie Speicherplatz auf dem RAM und dem EEPROM nicht für eine Installation des Applets reicht. Aus den erfolgreich durchgeführten Messungen geht deutlich hervor, dass bei größeren Mengen an Daten das Kopieren vom RAM auf das EEPROM deutlich schneller ist als das Kopieren vom EEPROM auf das EEPROM. Diese Erkenntnis gilt für alle getesteten Karten. Bei dem ersten getesteten Eeprom_Test Applet konnte diese Erkenntnis nicht eindeutig erlangt werden. Dies kann vielleicht an den geringeren Messzeiten liegen. Die Messungen ergaben nicht wie erwartet eine Stufenfunktion, da die untersuchten Karten nicht in der Lage waren, mehr als 512 Byte große Array zu kopieren.

6.2. Sicherheit

Das nachfolgende Applet liefert die Ergebnisse für den Namenspfad Sicherheit.

DES Applet

Das DES Applet für die DES-Verschlüsselung und Entschlüsselung dient zur Messung der „Performance“ des Kryptocoprozessors für den DES Algorithmus. Es implementiert die DES-Kryptofunktionen, sowie eine Dummy Funktion zur Messung des PC/SC und Java Card bedingten Overheads (Versenden des APDU Kommandos, etc.). Für die TripleDES-Kryptofunktion werden nur zwei DES-Schlüssel generiert und verwendet.

Dieses Applet enthält folgende Kommandos: Generieren eines DES-Schlüssel für die DES-Kryptofunkion (Put_DES_Key), Generieren von zwei DES-Schlüsseln für die TripelDES-Kryptofunkion (Put_3DES2_Key), DES-Verschlüsseln von 8 Bytes (DES_Encript_8_Bytes), DES-Entschlüsseln von 8 Bytes (DES_Decript_8_Bytes), DES-Verschlüsseln von 16 Bytes (DES_Encript_16_Bytes), DES-Entschlüsseln von 16 Bytes (DES_Decript_16_Bytes), DES-Verschlüsseln von, anders als oben, 16 Bytes (DES_Encript*_16_Bytes) und DES-Entschlüsseln von, anders als oben, 16 Bytes (DES_Decript*_16_Bytes).

Es wurden folgende Karten mit verschiedenen Messsystemen und mit verschiedenen Kartenlesern getestet:

- Mit dem ersten Messsystem und mit zwei Kartenlesern wurden die Karten A, C, E und I getestet. Karte H wurde nur mit dem Omnikey Kartenleser getestet.
- Mit dem zweiten Messsystem wurden Karte A, H und I getestet. Diese Karten wurden mit dem Omnikey Kartenleser getestet.
- Mit dem dritten Messsystem und dem Omnikey Kartenleser wurden Karte G getestet.
- Karte F konnte mit keinem Messsystem und/oder Kartenleser getestet werden.

(1) Die Abbildung 6.2.1 zeigt die Messungen vom DES Applet. Es wurden oben beschriebene Karten getestet.
Die Karten A, C und E brauchen für den gesamten Appletablauf um die 1000 msec. Alle anderen Karten brauchen dafür ganz unterschiedliche Zeiten. Die Karte I braucht 1500 msec, die Karte G 2500 msec und die Karte H um die 5000 msec.

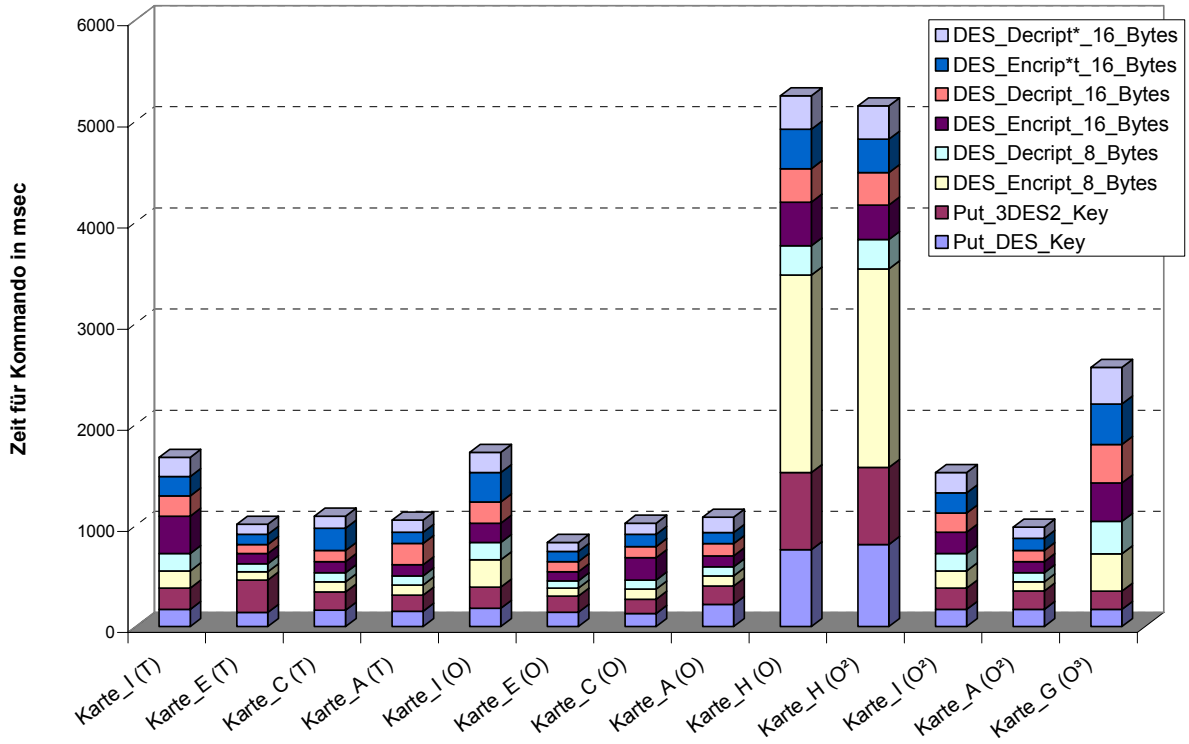


Abbildung 6.2.1 DES Applet, alle Karten

(2) In Abbildung 6.2.2 sind die absoluten Werte der jeweiligen Kommandos dargestellt. Offensichtlich kann man aber aus diesen Werten nicht allgemeingültig bestimmen, welches Kommando relativ zu den anderen Kommandos das zeitintensivste ist. Die Zeiten haben in den meisten Fällen dieselbe Größenordnung. Die vorkommenden Zeitdifferenzen unterscheiden sich höchstens um den Faktor 2 bis 3.

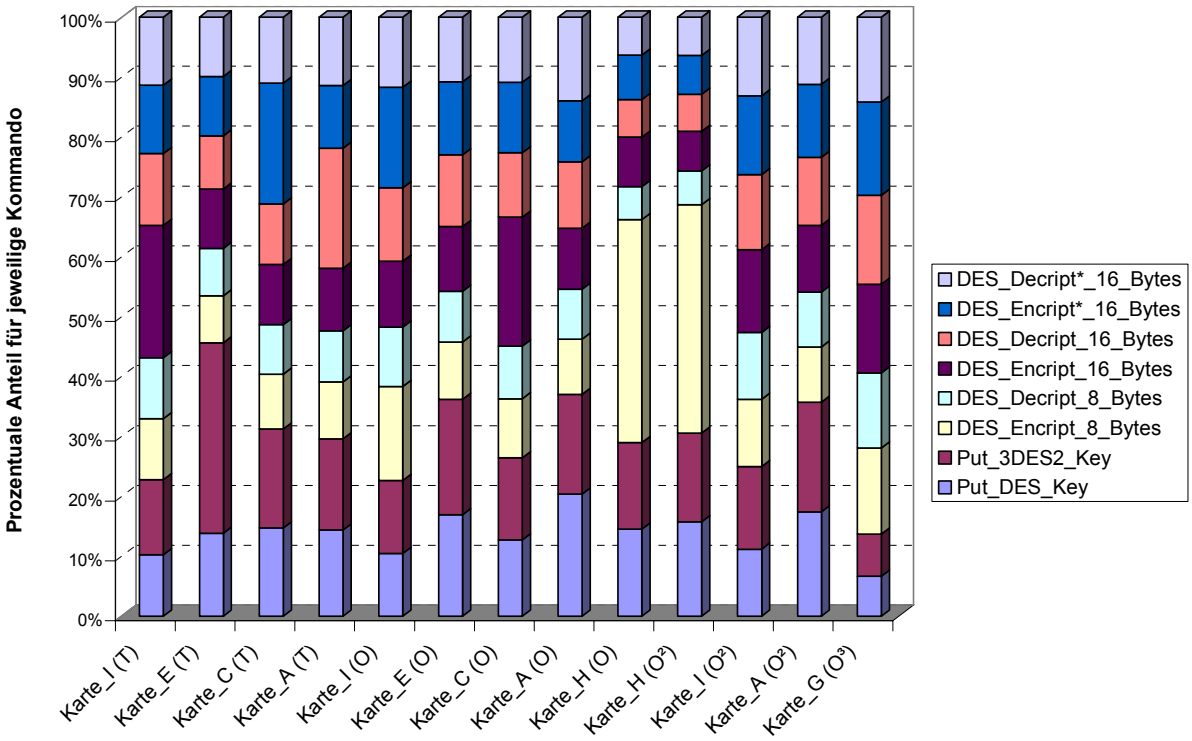


Abbildung 6.2.2 DES Applet, alle Karten

Aus den Abbildungen 6.2.1 und 6.2.2 wird die Schnelligkeit des Kryptocoprozessors ermittelt.

(3) Es wurden fünf Karten mit dem Omnikey Kartenleser und mit einem Messsystem getestet. Nur sechs Kommandos wurden ausgewertet.
Die Abbildung 6.2.3 zeigt, dass die Karte H bei jedem Kommando die langsamste ist. Diese braucht für das Kommando DES_Encript_8_Bytes die meiste Zeit, nämlich 1953 msec. Für die Kommandos Put_DES_Key und Put_3DES2_Key braucht die Karte jeweils 761 msec. Bei dieser Karte braucht die Entschlüsselung in jedem Fall weniger als die Verschlüsselung. Bei den vier anderen Karten braucht jedes Kommando weniger als 300 msec. Die Karte A benötigt für Put_DES_Key und Put_3DES2_Key jeweils um die 200 msec und für die anderen Kommandos zwischen 150 und 90 msec, die Entschlüsselung und die Verschlüsselung benötigen fast gleiche Zeiten.
Ganz ähnlich verhält sich die Karte E. Die ersten zwei Kommandos brauchten um die 150 msec, für die Verschlüsselung zwischen 101 und 80 msec und für die Entschlüsselung zwischen 100 und 70 msec. Die Karten C und I brauchten für Put_DES_Key und Put_3DES2_Key jeweils fast gleiche Zeiten, aber bei mindestens einer Verschlüsselung benötigten sie fast doppelt so viel Zeit als bei der dazugehörigen Entschlüsselung.

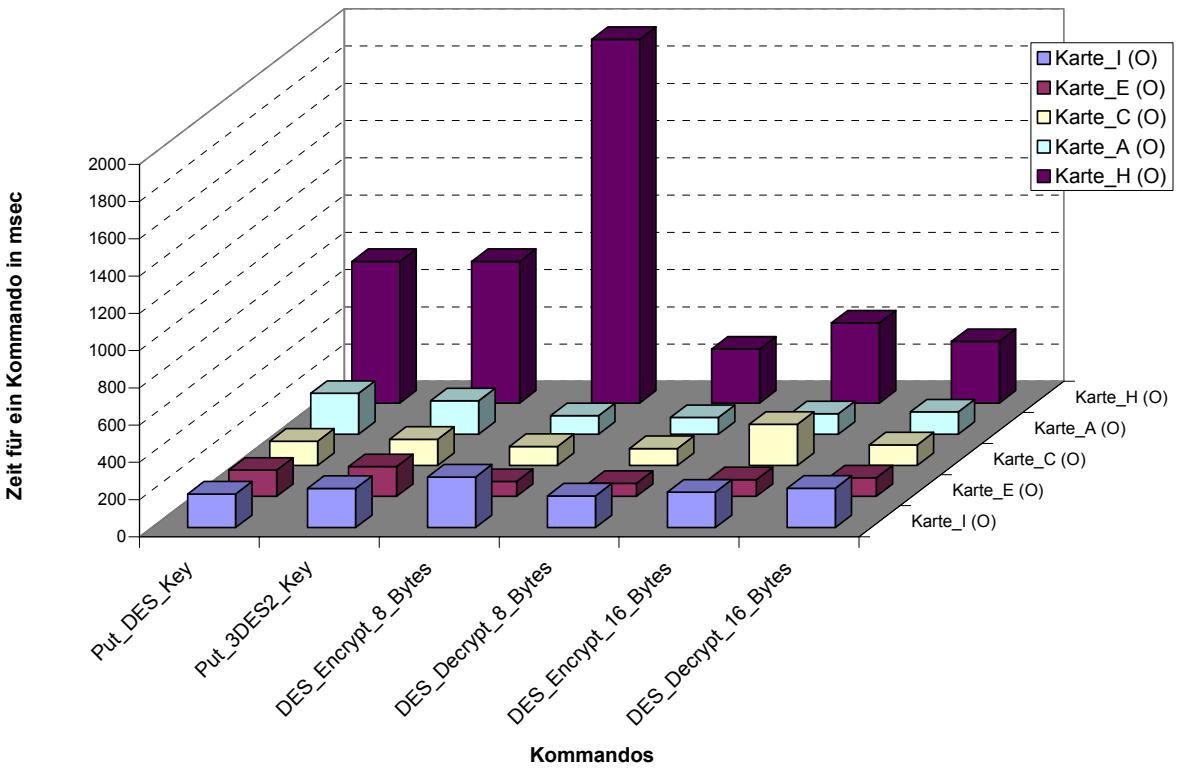


Abbildung 6.2.3 DES Applet

Aus diesen Messungen gehen die genaueren Ausführungszeiten der jeweiligen Kommandos hervor.

Aus den gesamten Messungen kann man die Abarbeitungszeiten von Ver- und Entschlüsselung für 8 und 16 Bytes ableiten. Diese Zeiten liegen im Bereich 100 bis 1900 msec, was allerdings nicht mit den Angaben aus den Spezifikationen der einzelnen Firmen [P00] übereinstimmt. Dort sind die Zeiten für die Abarbeitung von 64 Bytes angegeben und liegen zwischen 100 und 200 µsec (1 msec = 1000 µsec). Die Vermutung liegt nah, dass die in dieser Arbeit gemessenen Zeiten nicht reine DES Abarbeitungszeiten sind. Die gemessenen Zeiten enthalten sowohl die Abarbeitungszeit von DES als auch die Kommunikationszeit zwischen Karte und Terminal und zusätzlich noch die Verwaltungszeiten der Java Card Virtual Machine.

6.3. Java Card Virtual Machine und Java Card Sprachelemente

Das Control_Logik Applet liefern die Ergebnisse für den Namenspfad Java Card Sprachelemente.

Control_Logik Applet

Dieses Applet testet die Java Card Sprachelemente. Es wurden die Control Statements getestet wie: If _Else, For, Break_Continue, While, Switch, Do_While und Ternary.

Das Applet enthält für jedes Sprachelement eine Funktion und zusätzlich eine Dummy_Null Funktion (die in diesem Applet End heißt). Mit dem Applet soll getestet werden, wie viel Zeit die einzelnen Javakarten für die Abarbeitung der jeweiligen Control Statements brauchen. Die Kommandos werden 1-, 8- und 64-mal hintereinander wiederholt.

Es wurden folgende Karten mit verschiedenen Messsystemen und mit verschiedenen Kartenlesern getestet:

- Mit dem ersten Messsystem und mit zwei Kartenlesern wurden die Karten A, C, E und I getestet. Karte H wurde nur mit dem Omnikey Kartenleser getestet.
- Mit dem zweiten Messsystem wurden die Karte A, H und I getestet. Diese Karten wurden mit dem Omnikey Kartenleser getestet.
- Mit dem dritten Messsystem und dem Omnikey Kartenleser wurde dei Karte G getestet.
- Karte F konnte mit keinem Messsystem und/oder Kartenleser getestet werden.

(1) Es wurden alle Karten mit zwei Kartenlesern und mit drei Messsystemen getestet. In Abbildung 6.3.1 wurde das Kommando While bei allen Karten ausgewertet. Das Kommando wurde 1-, 8- und 64-mal wiederholt. In diesem Vergleich sind die Karten E, C und A die schnellsten, die Karte I lag bei 64 Loops mit Faktor fünf zur Karte E in der Mitte, und die Karten F,G und H waren bei 64 Loops mit ca. Faktor 20 zur Karte E am langsamsten.

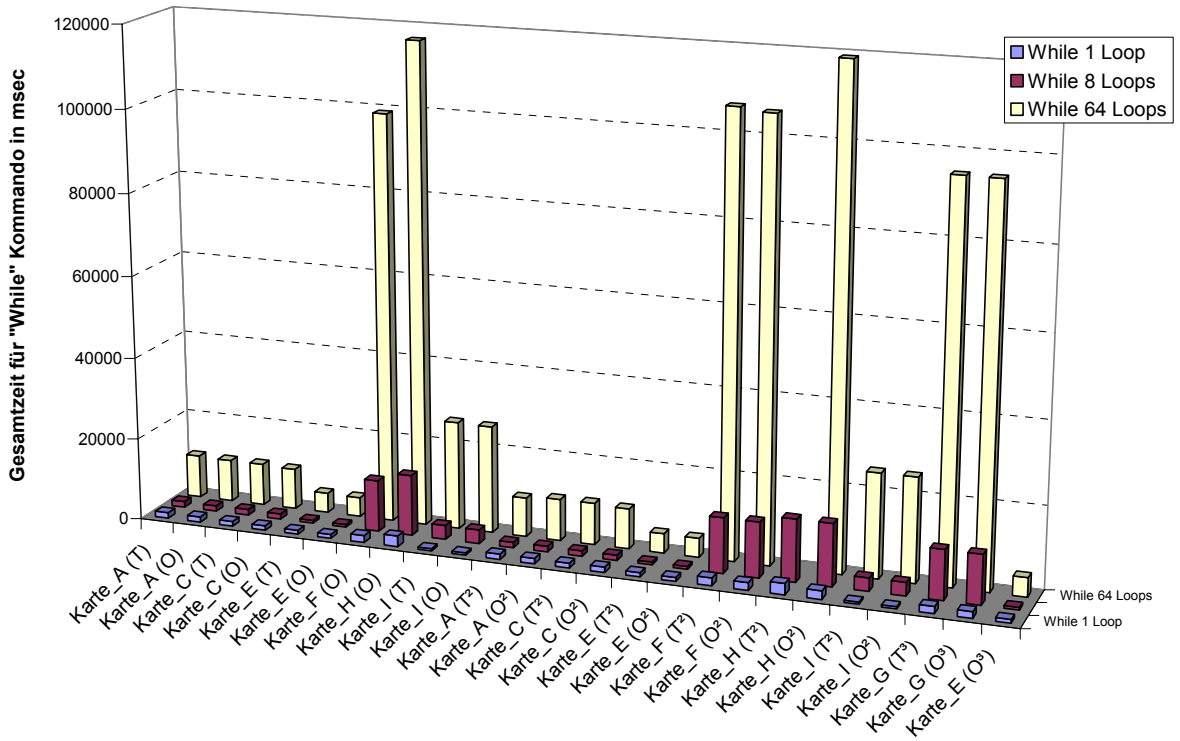


Abbildung 6.3.1 Control_Logik Applet, alle Karten

(2) Abbildung 6.3.2 präsentiert eine Einschränkung der Messergebnisse auf einem Messsystem (Messsystem zwei) und zwei Kartenlesern. Alle Karten wurden mit einem Messsystem und mit zwei Kartenlesern getestet. Es wurde nur das Kommando „While“ bei 1, 8 und 64 Loops ausgewertet.

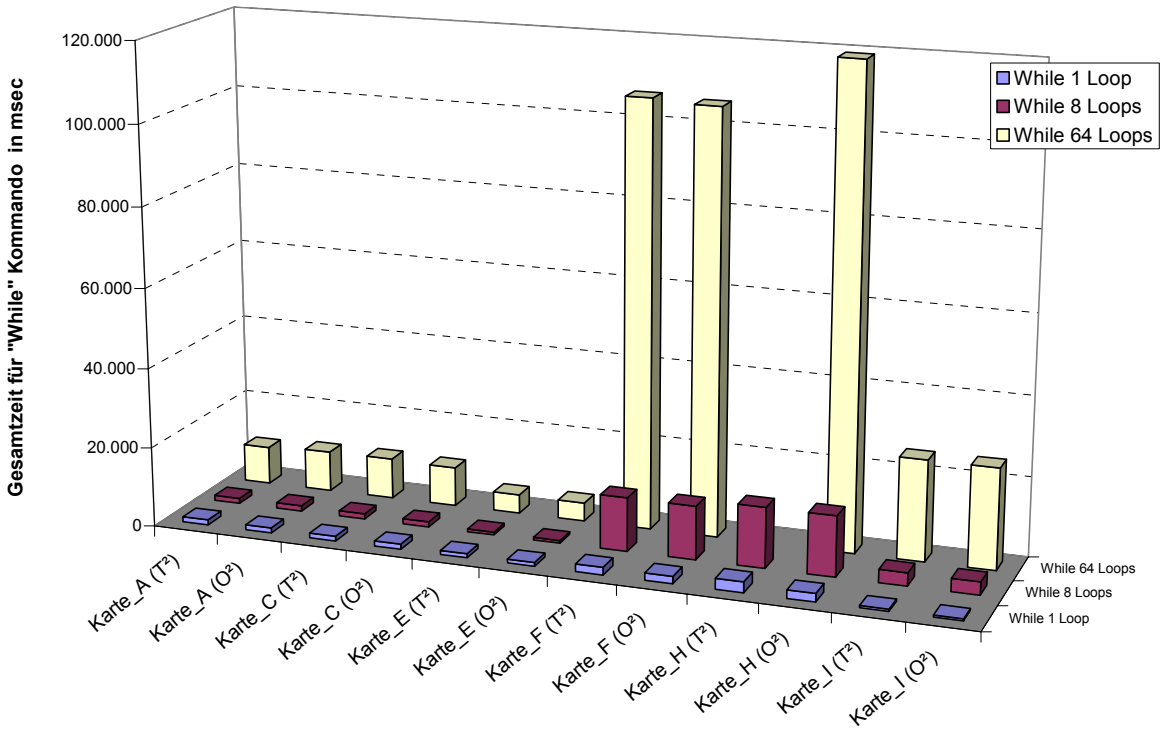


Abbildung 6.3.2 Control_Logik Applet, ein Messsystem und zwei Kartenleser

Aus den Abbildungen 6.3.1 und 6.3.2 geht zum wiederholten Mal hervor, dass eine Karte bei allen drei unabhängigen Messsystemen und mit zwei Kartenlesern immer dasselbe Ergebnis liefert.

(3) Alle Kommandos wurden 1-, 8- und 64-mal wiederholt. Es wurde zwei Karten (Karte I und Karte E) mit einem Messsystem (Messsystem eins) und einem Kartenleser (Omnikey) getestet. Die Abbildung 6.3.3 zeigt die Gesamtzeiten für alle Kommandos mit 1, 8 und 64 Loop(s). Beide Karten weisen die gleiche Tendenz auf. Je mehr ein Kommando wiederholt wird, desto mehr Zeit braucht es für die Abarbeitung. Das Kommando End braucht am wenigsten Zeit. Hier wird nur die reine Kommunikation zwischen der Karte und dem Terminal getestet. Bei End wird die Karte angesprochen. Sobald dies geschieht, antwortet das Terminal, das dies passiert ist. Die Kommandos If_Else und Ternary, die nicht nur Zeit für die Kommunikation, sondern auch für die Abarbeitung der Kommandos benötigen, haben zwar einen größeren Zeitaufwand als das Kommando End, dieser ist aber trotzdem relativ gering. Ternary ist eine Art If_Else Anweisung. Diese ist aber meist schneller als das If_Else. Dazu ein kleines Beispiel:

Beispiel:
(a > b) ? 1 : 0); // in Quellcode

Wenn a größer ist als b, dann gilt 1, sonst gilt 0. // wörtliche Erklärung

Bei der Karte I sind die Kommandos While, Do_While und Switch, diejenigen mit den längsten Ausführungszeiten. Bei der Karte E brauchen die Kommandos Switch und Break_Continue die meiste Zeit, die Kommandos While und Do_While benötigen einen mittleren Zeitbedarf.

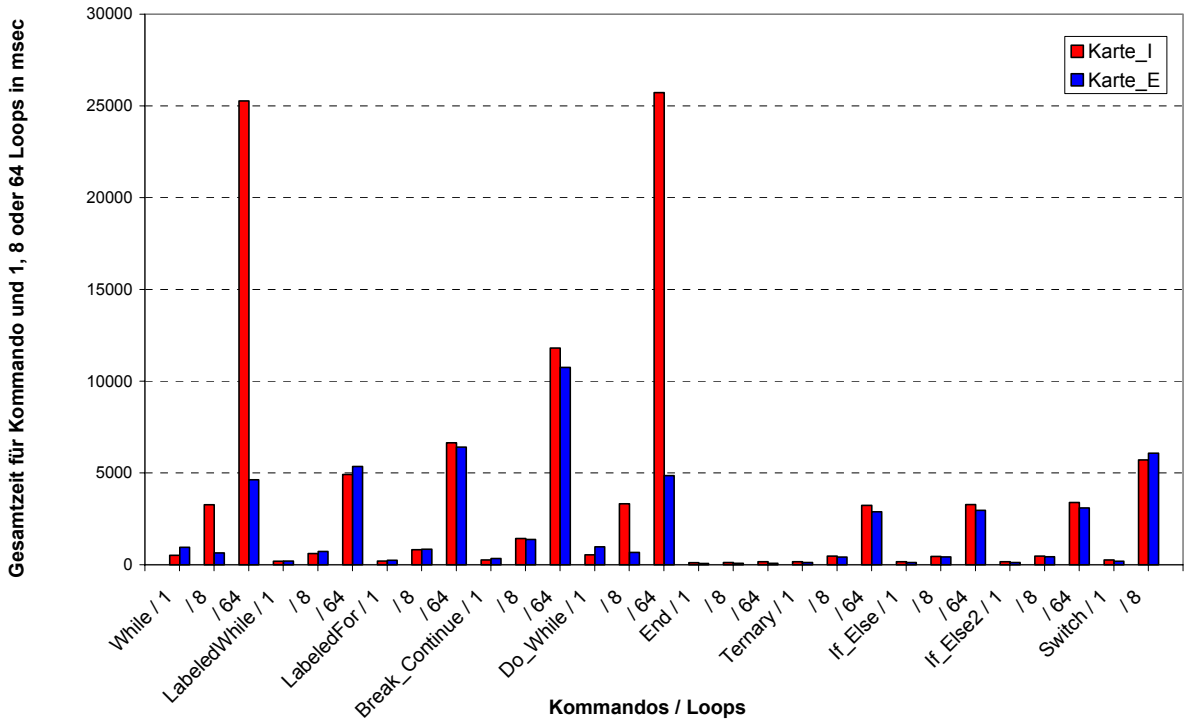


Abbildung 6.3.3 Control_Logik Applet, zwei Karten, alle Kommandos, alle Loops

Aus der Messung in Abbildung 6.3.3 geht hervor, dass die Kommandos If_Else und Ternary am wenigsten Zeit brauchen. Andere Kommandos dagegen brauchen bei verschiedenen Karten unterschiedlich viel Ausführungszeit. Aus den Messungen kann man ableiten, dass alle Kommandos, die While enthalten, bei allen Karten sehr viel Zeit brauchen im Vergleich zu If_Else.

(4) Alle Kommandos wurden 8-mal wiederholt. Es wurde zwei Karten (Karte I und Karte D) mit einem Messsystem und einem Kartenleser (Omnikey) getestet.
Die Abbildung 6.3.4 zeigt die genaueren Durchschnittszeiten bei 8 Loops für alle Kommandos. Die beiden Karten sind bei acht Kommandos gleichschnell, nur bei While und Do_While ist die Karte E um den Faktor drei schneller. Das Kommando Switch war bei beiden Karten am langsamsten, die Zeiten lagen zwischen 720 und 750 msec. Das schnellste Kommando war End. Bei Karte E waren die Zeiten der Kommandos, die While enthalten (While, Labeled_While und Do_While) fast alle gleich. Auch die Zeiten des If_Else Kommandos und Ternary waren fast identisch. Bei der Karte I waren zwar die Kommandos While und Do_While genauso schnell, aber das Kommando Labeled_While war dreimal schneller als While oder Do_While.

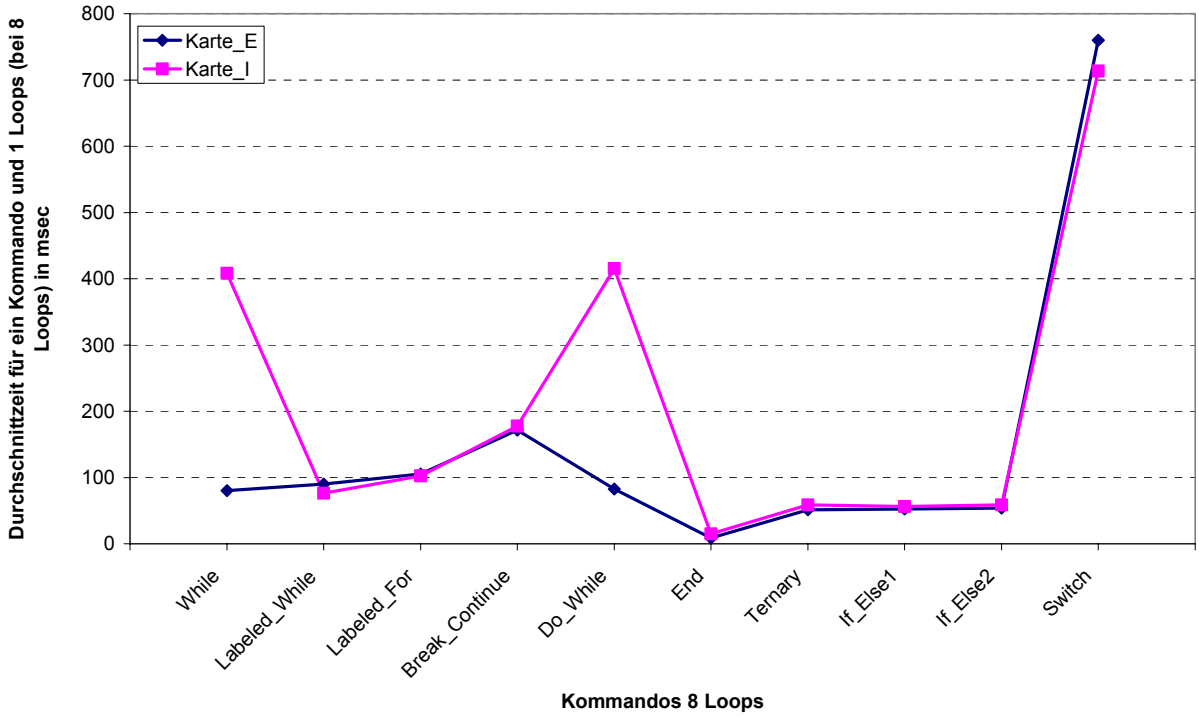


Abbildung 6.3.4 Control_Logik Applet, zwei Karten, alle Kommandos, ein Loop

Die Abbildung 6.3.4 macht deutlich, dass bei allen Kommandos, die 8-mal wiederholt wurden, die Ausführungszeiten bei den zwei Karten nur beim Kommando While und Do_While deutlich unterschiedlich waren. Für alle anderen Kommandos brauchten die Karten fast dieselbe Zeit. Man kann davon ausgehen, dass diese Kommandos in der Praxis nicht mehr als 8-mal hintereinander ausgeführt werden.

6.4. Applikation

Das nachfolgende Applet liefert keine Ergebnisse für das Spinnennetz. Das Purse Applet ist ein Beispiel für eine Applikation, die für eine Geldkarte benutzt werden kann. Diese Applikation beschränkt sich nur auf eine minimale Funktionalität.

Purse Applet

Dieses Applet beinhaltet elementare Bankoperationen wie die Abfrage des maximalen Kontostands (Maximum_Balance), die Abfrage des aktuellen Kontostands (Balance), die Abfrage der PIN beim Einzahlen oder Abheben von Geld (Verify_Debit_PIN oder Verify_Credit_PIN), Einzahlen oder Abheben vom Geld (Debit oder Credit), (Verify_Admin_PIN), (Update_Debit_PIN) und (Select_CM).

Es wurden folgende Karten mit dem ersten Messsystem und mit zwei Kartenlesern getestet:

- Mit dem ersten Messsystem und mit zwei Kartenlesern wurden die Karten A, C, E, F und I getestet. Karte H wurde nur mit dem Omnikey Kartenleser getestet.
- Mit dem dritten Messsystem und dem Omnikey Kartenleser wurde die Karte G getestet, jedoch nicht vollständig, da die Kommandos Debit, Credit und Select_CM nicht getestet werden konnten.

(1) Das Purse Applet wurde auf oben genannten Karten mit zwei bzw. einem Kartenleser(n) (Towitoko und Omnikey) mit zwei Messsystemen getestet. Es wurden alle Kommandos (Methoden, Operationen), die sich in der Legende befinden hintereinander einmal abgefragt.

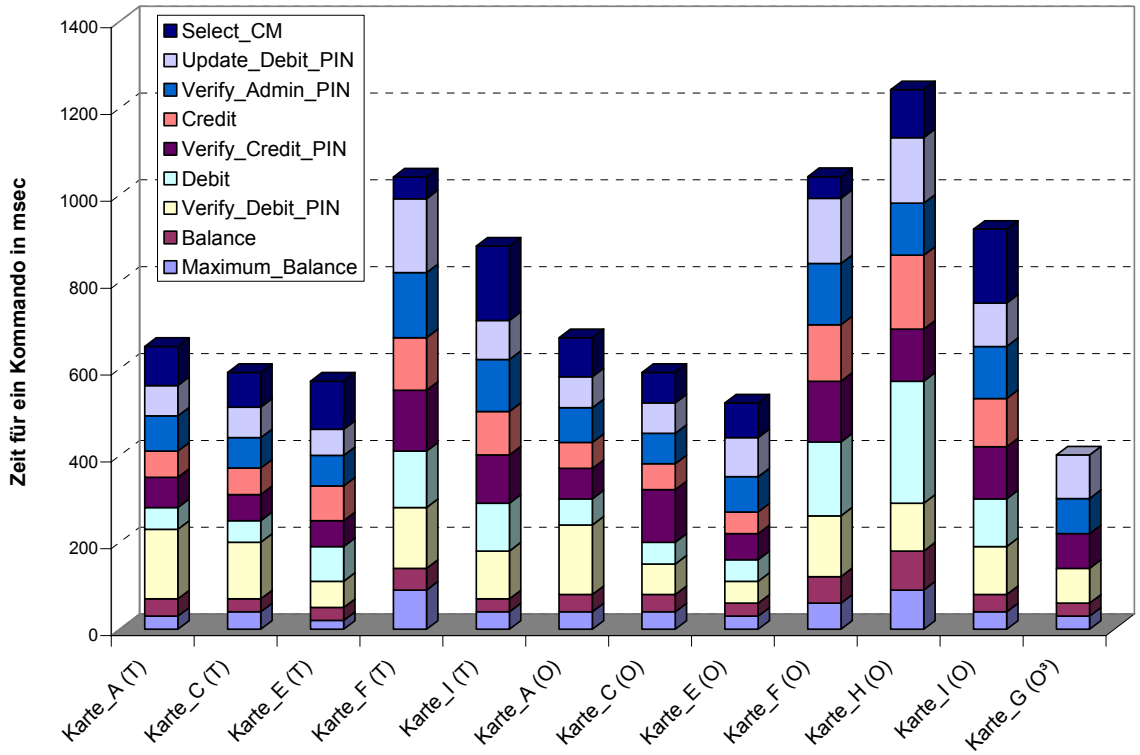


Abbildung 6.4.1 Purse Applet, alle getestete Karten

Die Abbildung 6.4.1 zeigt die Resultate. Die Karten A, C und E sind die schnellsten. Sie haben für die gesamte Appletabarbeitung um die 600 msec gebraucht. Mittelmäßig hat die Karte I abgeschnitten. Sie hat insgesamt ca. 850 msec gebraucht. Am längsten brauchten die Karten F und H. Sie haben um die 1000 bis 1200 msec für den kompletten Appletablauf benötigt. Aus den Messungen kann man folgern, dass auch bei Tests der gesamten Applikation die Kartenleser und die Messsysteme keinen Einfluss auf die Ergebnisse haben.

(2) Die Abbildung 6.4.2 zeigt die genaueren Ausführungszeiten bei jedem einzelnen Kommando. Die Karten A, C, E, F, H und I wurden mit dem Omnikey Kartenleser mit einem Messsystem getestet. Für das Kommando Maximum_Balance haben die Karten A, C, E und I 20 bis 40 msec, und die Karten F und H 60 bis 90 msec gebraucht. Beim Kommando Balance liegen die Werte wie beim Maximum_Balance im selbem Bereich. Bei den Kommandos Verify_Debit_PIN und Verify_Credit_PIN konnten interessante Werte beobachtet werden. Bei den Karten E, F, H und I sind die Werte bei Verify_Debit_PIN und Verify_Credit_PIN fast gleich und liegen für die Karten H und I zwischen 110 und120 msec, für die Karte F bei 140 msec und für die Karte E bei 50-60 msec. Bei den Karten A und C ist einer der Werte doppelt so hoch wie der andere Wert. Wenn also das Kommando Verify_Credit_PIN 60 msec benötigt, so benötigt das Kommando Verify_Debit_PIN 130 msec (siehe Karte C). Zu einem ähnlichen Effekt kommt es bei den Kommandos Debit und Credit bei der Karte H. Der Wert bei dem einen Kommando ist um 110 msec höher als bei dem anderem. Bei allen andern Karten sind die Werte für Debit und Credit fast gleich. Auch die beiden Kommandos Verify_Admin_PIN und Update_Debit_PIN liefern pro Karte fast dieselben Werte (Unterschied zwischen zwei Werten liegt bei 10 bis 30 msec). Für Kommando Select_CM braucht die Karte I 170 msec, die restlichen Karten brauchen für das Kommando zwischen 50 und 110 msec.

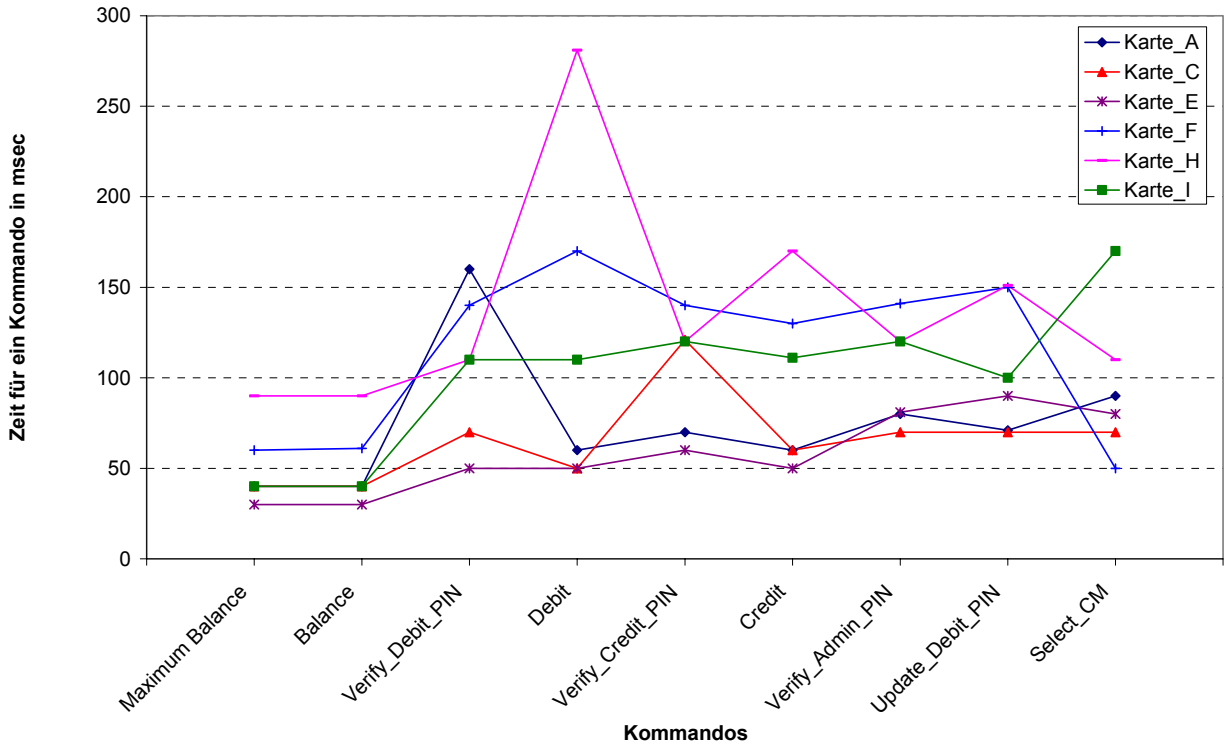


Abbildung 6.4.2 Purse Applet, sechs Karten, mit einem Kartenleser und einem Messsystem

Aus diesen Messungen kann man für jede Karte die einzelnen Ausführungszeiten für die jeweiligen Kommandos ablesen. Bei allen Karten wird die Abfrage vom Kontostand am schnellsten bearbeitet. Für alle anderen Kommandos brauchen die unterschiedlichen Karten unterschiedlich viel Zeit. Die Karte E ist auch bei der Abarbeitung von ganzen Applikation die beste.

6.5. Strommessungen

Die Ergebnisse aus den Kapiteln 6.1, 6.2, 6.3 und 6.4 zeigen, dass die Karte E die Spitzenposition belegt. Um alle Testergebnisse zu vergleichen, muss man davon ausgehen, dass alle Karten mit gleicher interner CPU-Frequenz arbeiten. Diese Parameter sollen über die Strommessungen bestimmt werden. Die Stromprofile eines beliebigen Mikrokontroller hängen mit den verarbeiteten Daten zusammen [K98], [KJJ98/I], [KJJ98/II], [HL02].
Die Strommessungen wurden auf einem digitalen Oszilloskop vom Typ WAVEMASTER der Firma LECROY mittels einer Kartenlesereigenentwicklung für Laboranwendungen von Infineon Technologies durchgeführt. Die Strommessungen wurden mit dem Eigenentwicklungsprogramm „showtrace2d“ von Infineon Technologies ausgewertet. Es wurden vier Karten getestet.
Normalerweise wird für die Abarbeitung eines Maschinenbefehls eine gewisse Anzahl an Takten gebraucht (siehe Abbildung 6.5.1). Die bekannte 8051 CPU benötigt z.B. 12 Taktzyklen zur Abarbeitung eines Maschinenbefehls [kei]. In weiteren Implementierungen sind kurze Ausführungszeiten bekannt [dol].

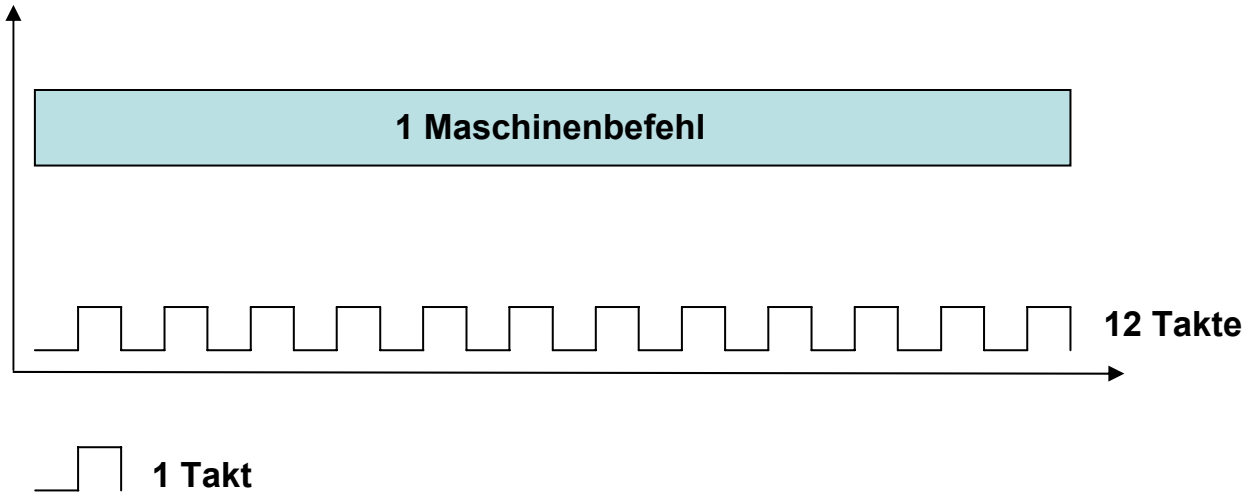


Abbildung 6.5.1 Anzahl der Takte pro Maschinenbefehl

(1) Abbildung 6.5.2 und Abbildung 6.5.3 zeigen den Stromverbrauch für die Abarbeitung eines Kommandos bei vier Karten: Karte A, Karte E, Karte H und Karte I. Die unterste Stromverbrauchskurve gehört der Karte A, diese Karte verbraucht am wenigsten Strom, die zweitunterste Stromkurve gehört der Karte H, die drittunterste der Karte E und die oberste der Karte I.

Stromverbrauch: Karte A < Karte H < Karte E< Karte I

Das „X Range“ Zusatzfenster beschreibt die x Achse. „First Sample“ beschreibt den ersten Messpunkt ab dem die Daten angezeigt werden. „Num Sample“ beschreibt die Anzahl der Messpunkte, die ab First Sample angezeigt werden. „Step Size“ bezeichnet die Anzahl der Messpunkte, um welche Num Sample vergrößert oder verkleinert wird. In allen Abbildungen hat das „Step Size“ den Wert eins.
In Abbildung 6.5.2 werden 1000 Messpunkte und in Abbildung 6.5.3 100000 Messpunkte vom Start (0 Wert) aus angezeigt. Die steigenden und fallenden Flanken sind in der Abbildung 6.5.2 deutlich im Stromprofil zu sehen. Bei den Flanken wird der Strom um ein Vielfaches (Faktor 3-5) mehr verbraucht als zwischen den Flanken. Die Ursache ist das Übersprechen vom externen Clock Signal. Eine Taktfrequenz beginnt immer mit einer fallenden Flanke. Man sieht, dass bei den Karten A, H und I der Stromverbrauch zwischen den Flanken gleichmäßig ist.

Abbildung 6.5.2 Stromverbrauch alle Karten (1)

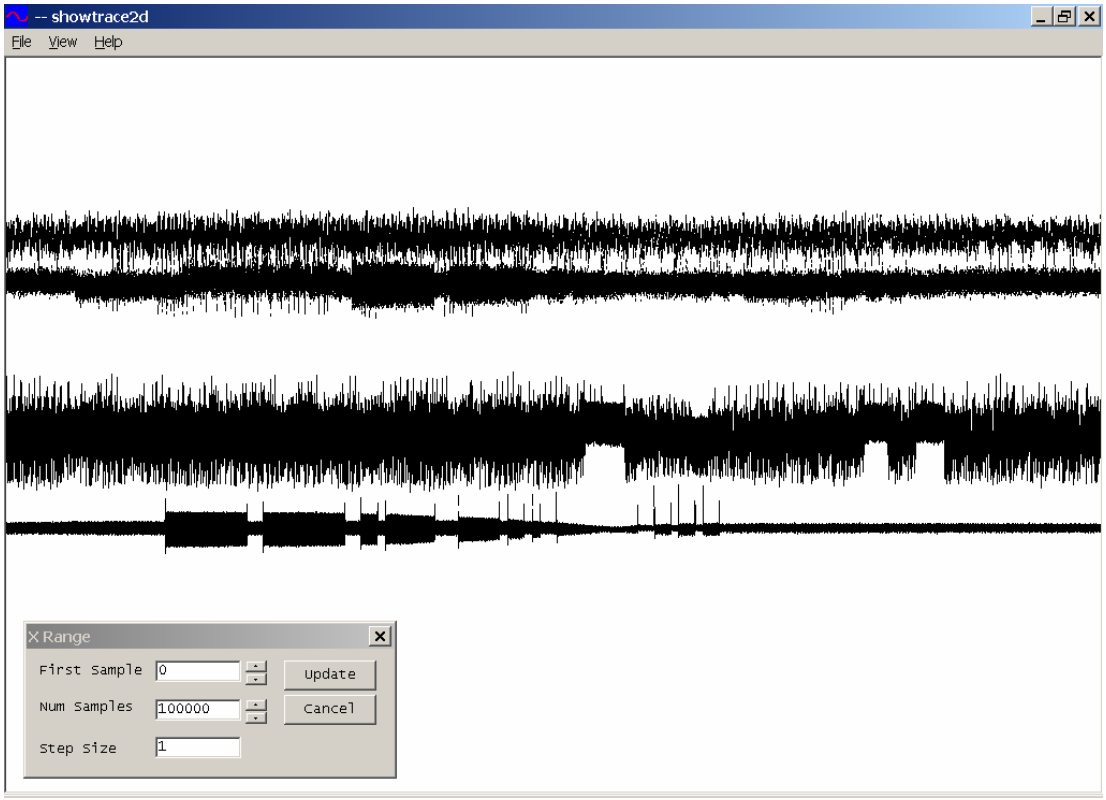


Abbildung 6.5.3 Stromverbrauch alle Karten (2)

(2) In den vier nacheinander folgenden Abbildungen (Abbildung 6.5.4, Abbildung 6.5.5, Abbildung 6.5.6, Abbildung 6.5.7) werden die Karten H, A, E und I genauer betrachtet. Es sind jeweils zwei Kurven zu sehen. Die oberste beschreibt die Taktfrequenz und die unterste den Stromverbrauch. Es werden 500 Messpunkte vom Startpunkt aus angezeigt.
Bei den Karten H und A sind die fallenden bzw. steigenden Flanken deutlich zu sehen. Bei der Karte I sind die Flanken viel kleiner als bei den Karten H und A.
Bei der Karte E sind die Flanken kleiner und zwischen den Flanken befinden sich andere Subpeaks, die fast so groß sind wie die Flanken.

Abbildung 6.5.4 Stromverbrauch und Takt bei Karte H

Abbildung 6.5.5 Stromverbrauch und Takt bei Karte A

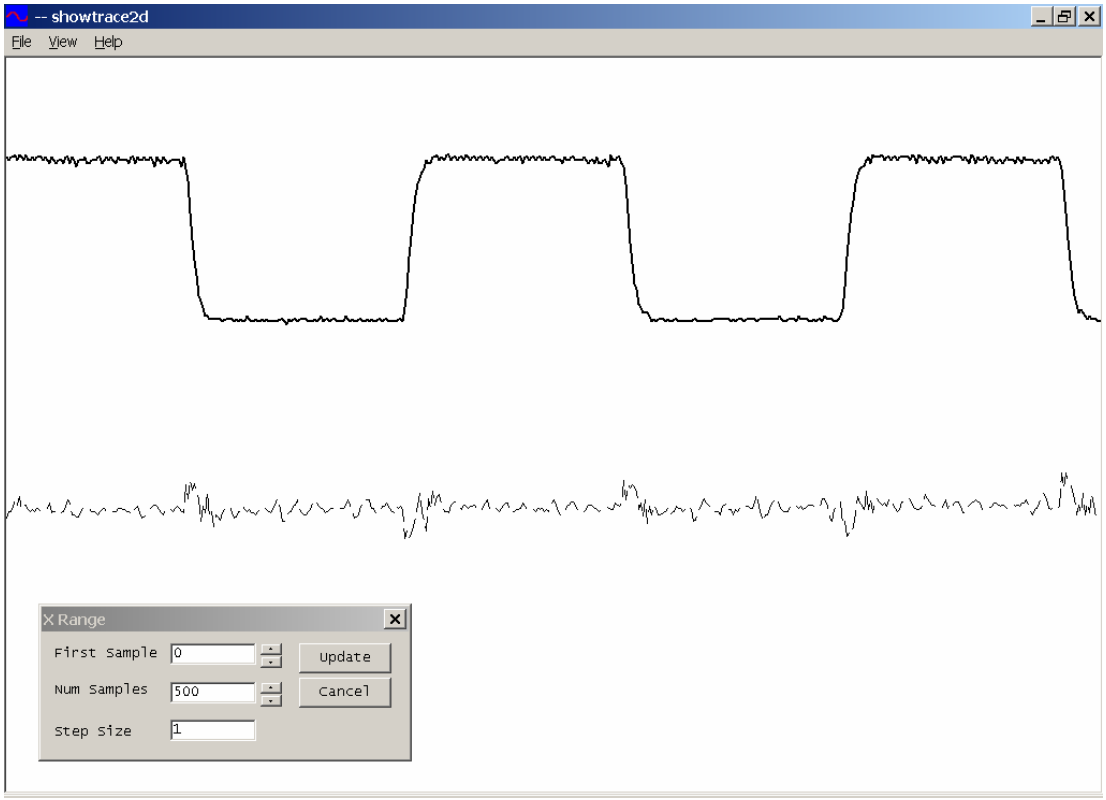


Abbildung 6.5.6 Stromverbrauch und Takt bei Karte E (1)

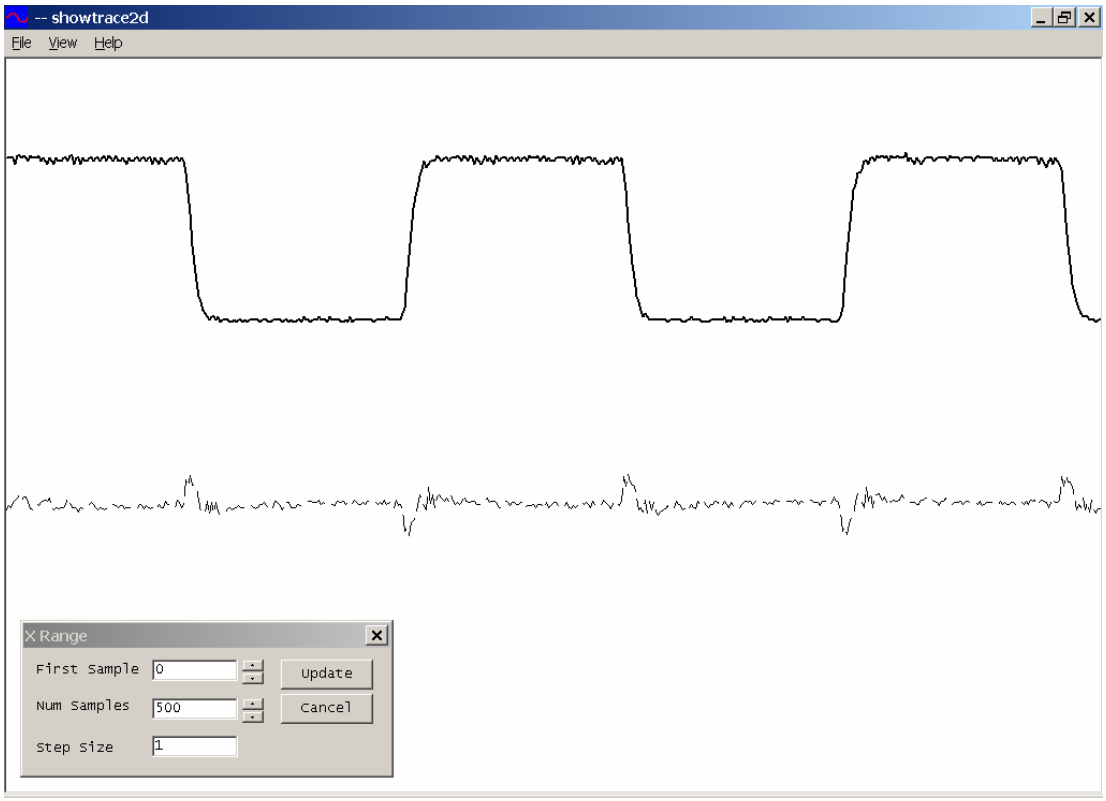


Abbildung 6.5.7 Stromverbrauch und Takt bei Karte I (1)

(3) Um das Ungewöhnliche bei der Karte E näher zu betrachten, wird die Anzahl der angezeigten Messpunkte auf 400 reduziert, und ein zufälliger Messpunkt (hier 3920000), ab dem die 400 Messpunkte angezeigt werden, gewählt. Die Abbildung 6.5.8 zeigt das oben genannte Szenario. Man sieht deutlich, dass sich jeweils zwischen einer fallender und jeweils einer steigenden Flanke drei zusätzliche Subpeaks befinden. Das deutet darauf hin, dass diese Karte mit einem Vielfachen der externen Frequenz arbeitet. Die gesamte Frequenz könnte also aus dem Achtfachen der externen Frequenz bestehen. Da die externe Frequenz bei dieser Messung 2,5 MHz war, hat die gesamte Frequenz bei der Karte E 20 MHz (wegen 8 * 2,5 MHz).
Zum Vergleich hat man die gleichen Parameter mit einer anderen Karte (Karte I) verwendet, wie in Abbildung 6.5.9 zu sehen ist.

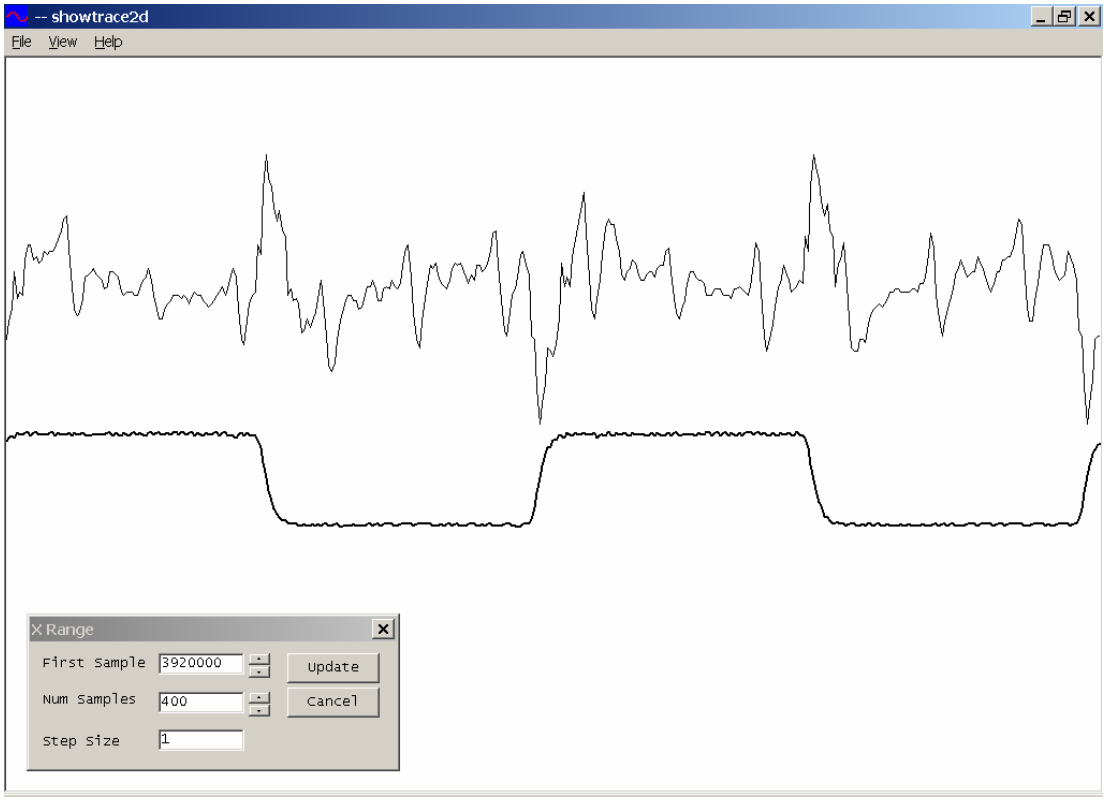


Abbildung 6.5.8 Stromverbrauch und Takt bei Karte E (2)

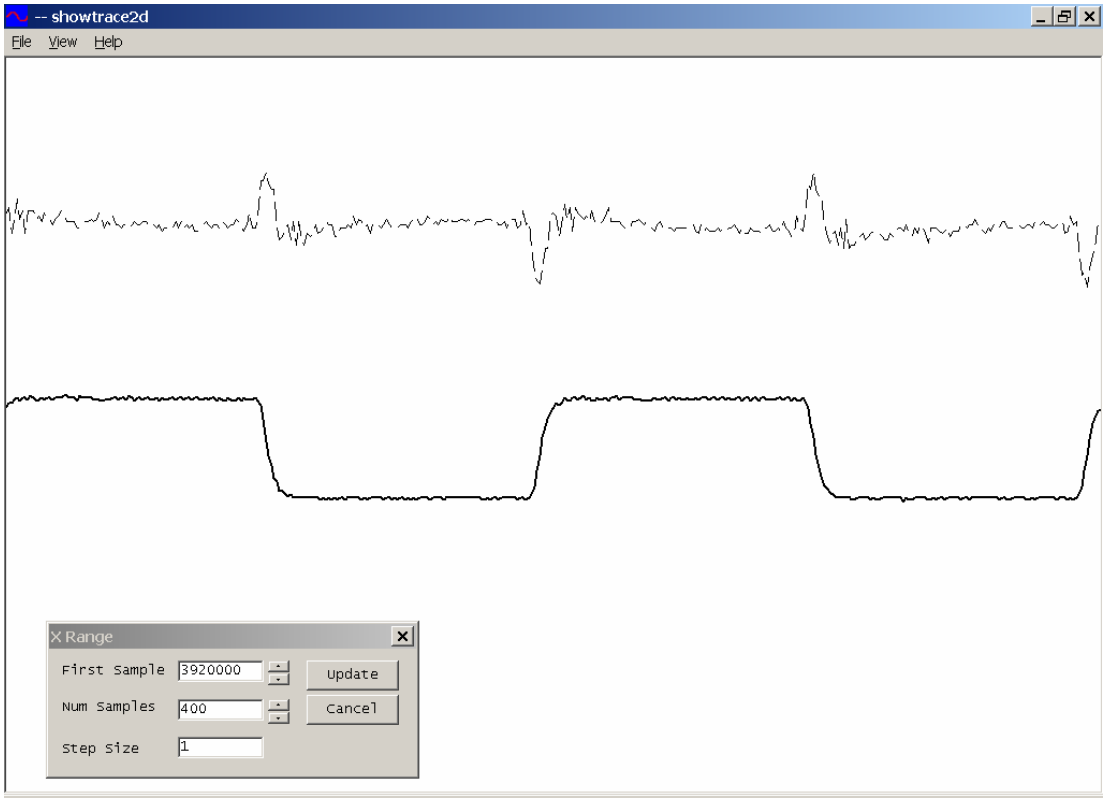


Abbildung 6.5.9 Stromverbrauch und Takt bei Karte I (2)

(4) In Abbildung 6.5.10 wird das Zeitverhältnis zwischen Stromverbrauch, Takt und I/O bei der Karte I (alle andere gemessene Karten liefern äquivalente Messergebnisse) angezeigt. Die erste Kurve beschreibt den Stromverbrauch, die zweite den Takt und die dritte die I/O Kommunikation. Bis zur ersten fallenden Flanke beim I/O bearbeitet die Karte ein Kommando, ab dann sendet sie eine Antwort. Man kann das auch beim Stromverbrauch deutlich sehen. Während ein Kommando abgearbeitet wird, ist der Stromverbrauch groß, wenn die Karte eine Antwort schickt und auf weitere Befehle wartet, ist der Stromverbrauch kleiner. Nur bei einer fallenden oder steigenden Flanke beim I/O steigt auch der Stromverbrauch als Folge des Übersprechens.

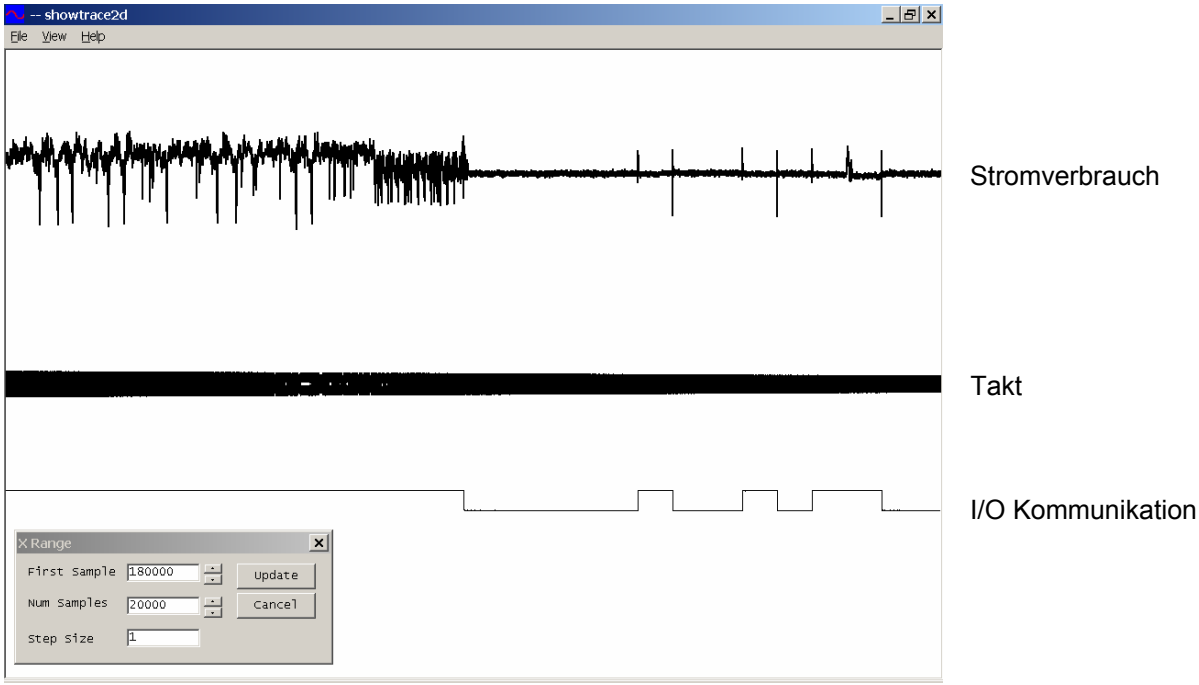


Abbildung 6.5.10 Stromverbrauch, Takt und I/O bei Karte I

6.6. Zusammenfassung der Ergebnisse

Aus all den Messungen geht hervor:

- nicht jedes Applet kann auf eine beliebige Karte geladen und/oder dort ausgeführt werden
- es können nicht beliebig große Arrays auf jede Karte geladen werden, da nicht bei allen Karten genügend freier Speicherplatz zur Verfügung steht
- beim Kopieren von großen Arrays ist das Kopieren vom RAM auf das EEPROM viel schneller als das Kopieren vom EEPROM auf das EEPROM
- es war nur bei der Karte I möglich die Seitengrößen vom EEPROM zu bestimmen, bei allen anderen Karten war es nicht möglich
- die Leistung des Kryptocoprozessors kann ermittelt werden
- es kann getestet werden, wie viel Zeit die jeweiligen Java Card Sprachelemente benötigen
- die Ausführungszeiten von einem Applet und/oder Kommando sind von den Kartenlesern und/oder Messsystemen unabhängig, die Zeiten sind fast identisch
- durch die Strommessungen kann festgestellt werden, dass die schnellste Karte zusätzlich mit einem Vielfachen der externen CPU-Frequenz arbeitet, woraus folgt, dass diese Karte eigentlich immer schneller sein sollte als die Karten, deren CPU nur mit einer externen Frequenz arbeiten. Als weitere, technische Bewertung sollte man die Ergebnisse mit der erhaltenen CPU-Frequenz skalieren, bzw. den Stromverbrauch berücksichtigen. Aus der Anwendersicht ist die Karte E die schnellste.

In dieser Arbeit wurden drei Funktionsklassen (Schreib-/Lesezugriffe, Java Card Sprachenelemente, Sicherheit) abgearbeitet und die Ergebnisse wurden für jede Karte in einem Spinnennetz aufgetragen.

Die **Karte A** erreicht bei der Funktionsklasse Schreib-/Lesezugriffen 66,8 Punkte, bei der Funktionsklasse Java Card Sprachenelemente 58 Punkte und bei der Funktionsklasse Sicherheit 88,4 Punkte.

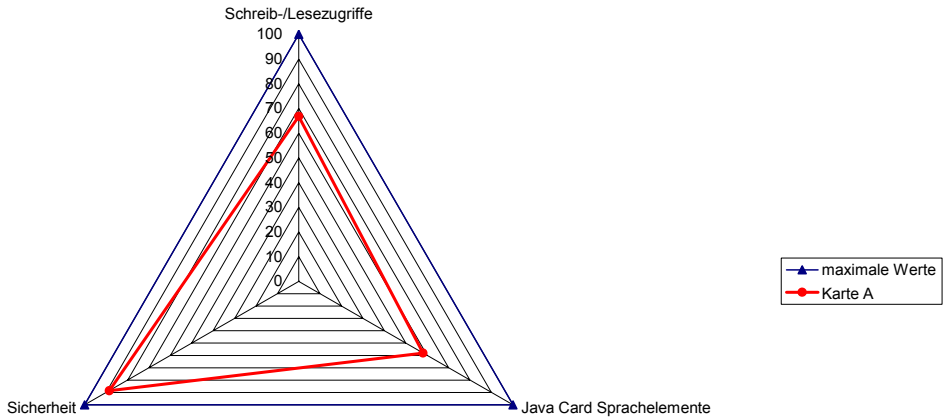


Abbildung 6.6.1 Ergebnisse bei der Karte A

Die **Karte C** erreicht bei der Funktionsklasse Schreib-/Lesezugriffe 66,7 Punkte, bei der Funktionsklasse Java Card Sprachenelemente 64 Punkte und bei der Funktionsklasse Sicherheit 87,2 Punkte.

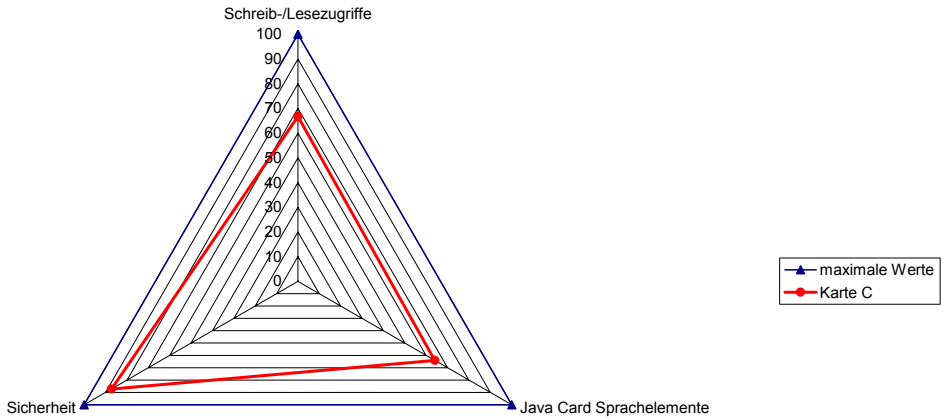


Abbildung 6.6.2 Ergebnisse bei der Karte C

Die **Karte E** erreicht bei der Funktionsklasse Schreib-/Lesezugriffe 100 Punkte, bei der Funktionsklasse Java Card Sprachenelemente 77,8 Punkte und bei der Funktionsklasse Sicherheit 100 Punkte.

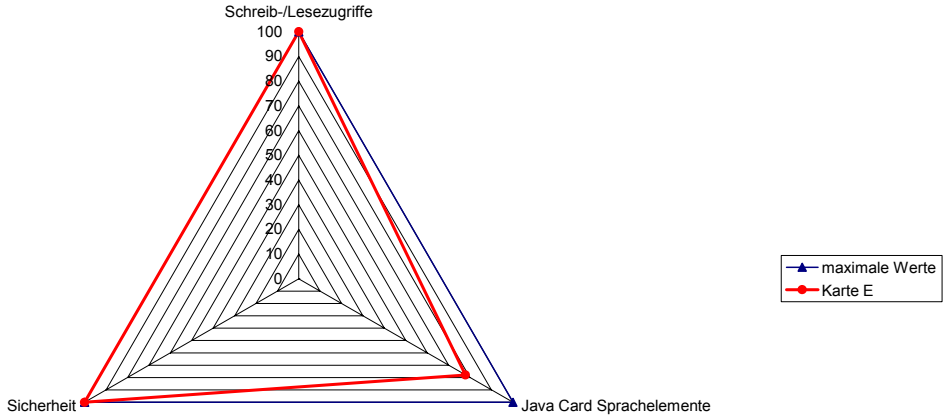


Abbildung 6.6.3 Ergebnisse bei der Karte E

Die **Karte F** erreicht bei der Funktionsklasse Schreib-/Lesezugriffe 27 Punkte, bei der Funktionsklasse Java Card Sprachenelemente 39,7 Punkte und bei der Funktionsklasse Sicherheit 0 Punkte (d.h. diese Klasse konnte nicht getestet werden, siehe dazu Kapitel 6.2).

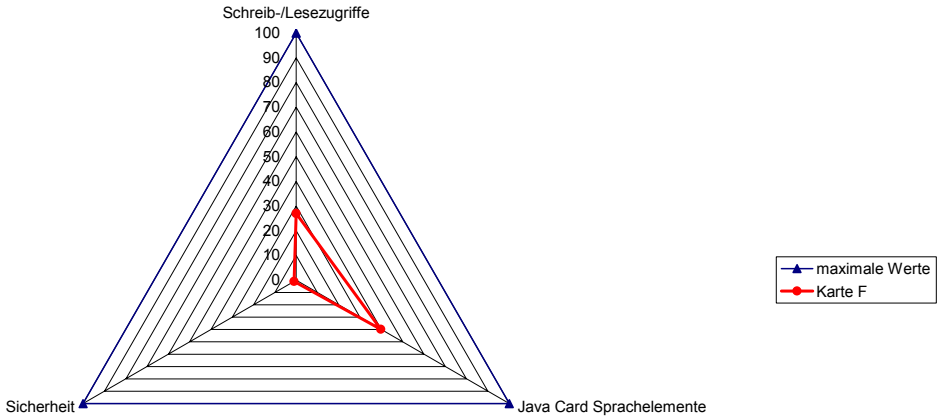


Abbildung 6.6.4 Ergebnisse bei der Karte F

Die **Karte G** erreicht bei der Funktionsklasse Schreib-/Lesezugriffe 24,2 Punkte, bei der Funktionsklasse Java Card Sprachenelemente 12,1 Punkte und bei der Funktionsklasse Sicherheit 35,9 Punkte.

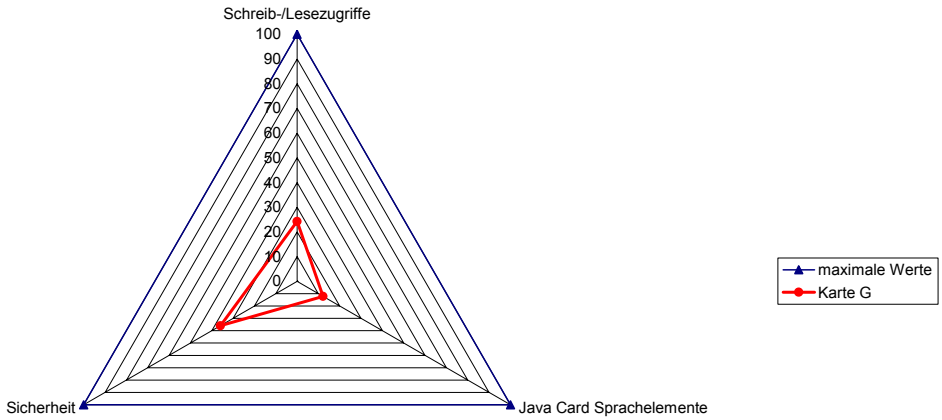


Abbildung 6.6.5 Ergebnisse bei der Karte G

Die **Karte H** erreicht bei der Funktionsklasse Schreib-/Lesezugriffe 28,2 Punkte, bei der Funktionsklasse Java Card Sprachenelemente 18,3 Punkte und bei der Funktionsklasse Sicherheit 17,9 Punkte.

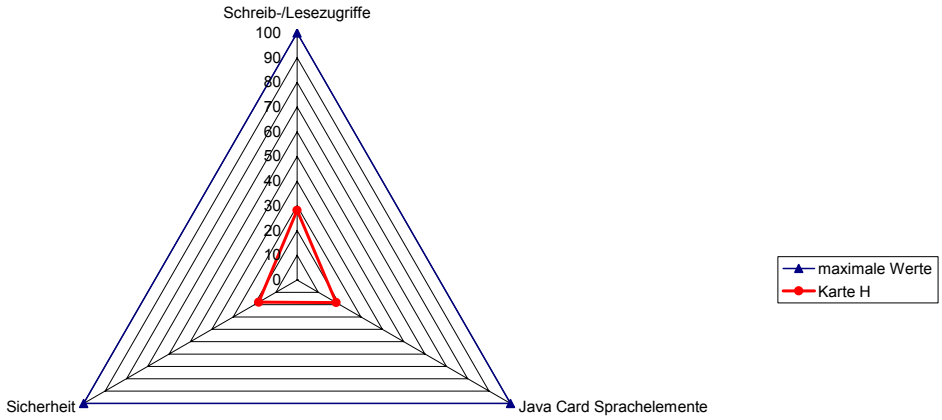


Abbildung 6.6.6 Ergebnisse bei der Karte H

Die **Karte I** erreicht bei der Funktionsklasse Schreib-/Lesezugriffe 33,5 Punkte, bei der Funktionsklasse Java Card Sprachenelemente 100 Punkte und bei der Funktionsklasse Sicherheit 56,2 Punkte.

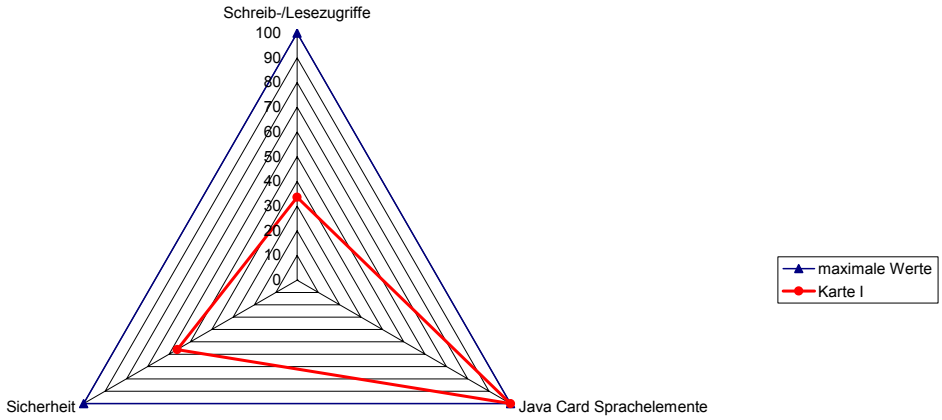


Abbildung 6.6.7 Ergebnisse bei der Karte I

In Abbildung 6.6.8 wurden von allen Karten die Ergebnisse eingetragen.

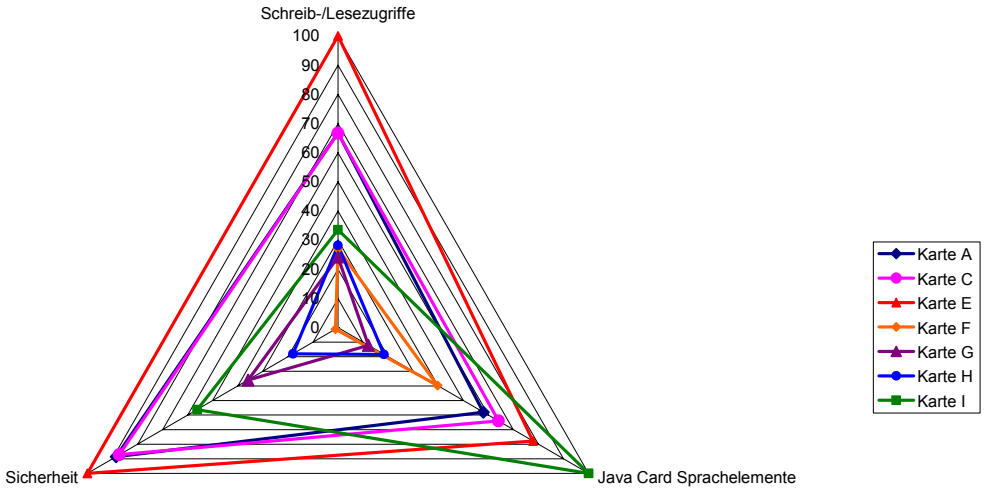


Abbildung 6.6.8 Ergebnisse bei allen Karten

Aus der Abbildung 6.6.8 kann man entnehmen, dass die Karte E die beste Java Card ist. Die Karten A und C liefern auch gute Resultate. Karte I liegt im mittleren Bereich. Sie ist bei der Funktionsklasse Java Card Sprachelemente die beste. Die Karten F und G liegen im unteren Bereich, die erste liefert bei Funktionsklasse Java Card Sprachelemente deutlich bessere Ergebnisse als bei Funktionsklasse Schreib-/Lesezugriffe, die Funktionsklasse Sicherheit konnte auf der Karte nicht getestet werden. Die Karte G liefert bei Funktionsklasse Sicherheit bessere Resultate als bei den anderen Funktionsklassen. Die Karte H hat in der gesamten Bewertung am schwächsten abgeschnitten.

7. Ausblick

Oben wurden die Ergebnisse des Benchmarking von Java Cards präsentiert und zusammengefasst. Dazu wurde in Kapitel 3 eine Metrik erstellt, um ein vernünftiges Vergleichsmaß zu erhalten. Dabei wurde auch zugleich eine Methodik vorgestellt, wie eine Metrik entwickelt wird, indem Richtlinien genannt wurden, was für Eigenschaften eine gute Metrik erfüllen muss. In der Metrik werden die Funktionsklassen mit ihren Elementen bestimmt. Diese Elemente werden dann mittels Applets getestet. Die Applets wiederum folgen bei ihrer Implementierung bestimmten Designregeln, damit z.B. ein auf eine Karte geladenes Applet ressourcensparend ist, und damit dieses Applet auch dasjenige Element testet, welches gerade getestet werden soll. Die Designregeln, die in dieser Arbeit verwendet worden sind, setzen sich aus allgemeinen Regeln zur Erstellung von Java Card Applets und selbst erstellten Richtlinien für Applets, die zum Benchmarking von Java Cards benutzt werden, zusammen.
Diese Regeln erschlagen nicht das ganze Spektrum der Möglichkeiten, um Applets zu erstellen. Daher wird vorgeschlagen, sich für weitere Arbeiten, die sich mit dem Benchmarking von Java Cards befassen, weitere Regeln und Richtlinien zur Erstellung von Benchmarking Java Card Applets zu überlegen.
Da in dieser Arbeit nur drei der acht Funktionsklassen getestet und ausgewertet, und somit ins Spinnennetz eingetragen worden sind, wäre es notwendig dies im Rahmen zukünftiger Arbeiten zu diesem Thema zu vervollständigen. Diese Ausarbeitung ist als Einführung in diese Thematik zu verstehen, so dass nicht alle Funktionsklassen ausgewählt worden sind.

Anhang

Applikation Ebene		Java Ebene			
Chipkartenfunktionen und Unterfunktionen	**Funktion Abkürzung**	**Java Methode**	**Bibliothek**	**Funktionsklasse**	**Hardware -Modul**
Checksums/Hash					
CHInitialize	t_{CH_I}				
CHUpdate	t_{CH_U}				
CHFinalize	t_{CH_F}				
File System					
File Open	t_{FS_O}	getCurrentElementrayFile()			
		getCurrentDidicatedFile()			
File Close	t_{FS_C}				
Read	t_{FS_R}	getData()		Lesezugriff	EEPROM
		arrayCompare()		Lesezugriff	EEPROM
Write	t_{FS_W}	updateRecord()		Schreibzugriff	
		putData()			
		writeRecord()			
Write Though	t_{FS_WT}				
Write TearingSave	t_{FS_WTS}				
WritePhysical	t_{FS_WP}	arrayCopyNonAtomic()		Arrays	
		arrayCopy()		Arrays	
		arrayFillNonAtomic()		Arrays	
Cryptography					
CryptoInitialize	$t_{C\_I\_X}$	getKey()	javacard.security	Sicherheit	
		setKey	javacard.security	Sicherheit	
		init()	javacard.security	Sicherheit	
		getAlgorithm()	javacard.security	Sicherheit	
	X=DES	getInstance()	javacard.security	Sicherheit	DES
		buildKey()	javacard.security	Sicherheit	DES
	X=RSA	getInstance()	javacard.security	Sicherheit	Crypto CP
		getExponent()	javacard.security	Sicherheit	Crypto CP
		getModulus()	javacard.security	Sicherheit	Crypto CP
		setExponent()	javacard.security	Sicherheit	Crypto CP
		setModulus()	javacard.security	Sicherheit	Crypto CP
CryptUpdate	$t_{C\_U\_X}$	update()	javacard.security	Sicherheit	
CryptFinalize	$t_{C\_F\_X}$				
GenerateRandom	t_{C_R}	generateData()	javacard.security	Sicherheit	
		getInstance()	javacard.security	Sicherheit	
GeneratePrime	t_{C_P}				
Communication					
ComInitialize	$t_{COM\_I\_X}$				
ComSend	$t_{COM\_S\_X}$	sendOutgoing()	javacard.framework		

		setOutgoingAndSend()	javacard.framework		
		sendBytes()	javacard.framework		
		sendBytesLong()	javacard.framework		
		setOutgoing()	javacard.framework		
		setOutgoingLenth()	javacard.framework		
		setOutgoingNoChaning()	javacard.framework		
ComReceive	$t_{COM\_R\_X}$	setIncomingAndReceive()	javacard.framework		
		getBytes()			
		getBuffer()			
		getShort()	javacard.framework		
		receiveBytes			
		getProtocoll()			
Biometric					
(not avialable)					
Java					
Java VM	t_{J_B}	install()	javacard.framework	Appletmethoden	
		select()	javacard.framework	Appletmethoden	
		process()	javacard.framework	Appletmethoden	
		deselect()	javacard.framework	Appletmethoden	
		selectingApplet()	javacard.framework	Appletmethoden	
		register()	javacard.framework	Appletmethoden	
Java_Language	t_{J_all}	if…else	java.lang	JC Sprachelemente	
		while	java.lang	JC Sprachelemente	
		switch	java.lang	JC Sprachelemente	
		do	java.lang	JC Sprachelemente	
		Addition	java.lang	Operatoren	CPU
		Multipilkation	java.lang	Operatoren	CPU
		And	java.lang	Operatoren	CPU
		Or	java.lang	Operatoren	CPU
		Not	java.lang	Operatoren	CPU
		Equels	java.lang	Operatoren	CPU
		byteShift	java.lang	Operatoren	CPU
Multipplication					
(not avialable)					
Extras					
		makeTransientArray	javacard.framework	Arrays	
		makeTransientByteArray	javacard.framework	Arrays	
		makeTransientObjectArray	javacard.framework	Arrays	
		makeTransientShortArray	javacard.framework	Arrays	
		makeShort()	javacard.framework		

Abkürzungen

AID	Application Identifier
APDU	Application Protocol Data Unit
API	Application Programming Interface
ATR	Answer To Reset
CAP	Converted Applet
CLA	(Application) Class
CPU	Central Processing Unit
DES	Data Encryption Standard
EEPROM	Electrically Erasable Programmable Read Only Memory
ENC	Encryption
EXP	Java Card Export File
FCI	File Control Information
ID	Identifier
IDE	Integrated Developer Environment
INS	Instruction
ISO	International Standardisation Organization
JCA	Java Card Assembler
JCAPI	Java Card Application Programming Interface
JCRE	Java Card Runtime Environment
JCVM	Java Card Virtual Machine
Lc	(Data) Length Command
Le	(Data) Length Expected
NPU	Numeric Processing Unit (Kryptocoprozessor)
OCF	Open Card Framework
OP	Open Platform
OS	Operating System
P1/P2	Parameter 1 / Parameter 2 of the command header
PC/SC	Personal Computer/Smart Card
PIN	Personal Identification Number
RAM	Random Access Memory
RID	Registered Application Provider Identifier
ROM	Read Only Memory
RSA	Rivest, Shamir and Adlemann Asymmetric Algorithm
RST	Reset
SCR	Script File, Skript Datei
SW1/SW2	Status Word 1 / Status Word 2
TPDU	Transport Protocol Data Unit
VM	Virtual Machine
VOP	Visa Open Platform

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[C00] Zhiqun Chen: *"Java Card Technology for Smart Cards "*, Addison-Wesley, 2000

[dol] www.dolphin.fr/flip/logic/8051/logic_8051_overview.html

[DS95] Dallas Semiconductor: "*Secure Microprocessor Chip*", Dallas Semiconductor, 1995
http://sub.chipdoc.ru/pdf/Dallas_Sem/5002fp.pdf?fid=12

[G02] Frank Gräber: "*Chipkarten-Schutzmaßnahme*", Card-Forum, 2002

[G03] Gemplus "*Physical Security of Smart Cards: Test Procedure Proposal*", Gemplus, 2003,
http://www.ncits.org/tc_home/b105htm/b105Doc2003/N03-143-Sec-v.01.pdf

[GJ98] Scott B. Guthery, Timothy M. Jurgensen: *"Smart Card Developer's Kit"*, Macmilian Technical Publishing, Indianapolis, Indiana, 1998

[HL02] Peter Hofreiter, Dr. Peter Laackmann: „*Elektromagnetische Abstrahlung*", Card-Forum, 2002

[HLL03] Peter Hofreiter, Dr. Peter Laackmann, Dimitri Lyebyedyev: „*Spike- und Glitchangriffe gegen Security Controller*", Card-Forum, 2003

[HNSSS02] Uwe Hansmann, Martin S. Nicklous, Thomas Schäck, Achim Schneider, Frank Seliger: *"Smart Card Application Development Using Java"*, Springer, 2002

[ISO/IEC 7816 -1] International Standardisation Organization: "Identification cards – Integrated circuit(s) cards with contacts, Part 1: Physical characteristics", 1998

[ISO/IEC 7816 -2] International Standardisation Organization: "Identification cards – Integrated circuit(s) cards with contacts, Part 2: Dimensions and location of the contacts", 1999

[ISO/IEC 7816 -3] International Standardisation Organization: "Identification cards – Integrated circuit(s) cards with contacts, Part 3: Electronic signals and transmission protocols", 1997

[ISO/IEC 7816 -4] International Standardisation Organization: “Identification cards – Integrated circuit(s) cards with contacts, Part 4: Inter-industry commands for interchange”, 1995

[ISO/IEC 7816 -5] International Standardisation Organization: “Identification cards – Integrated circuit(s) cards with contacts, Part 5: Numbering system and registration procedure for application identifiers”, 1994

[ISO/IEC 7816 -6] International Standardisation Organization: “Identification cards – Integrated circuit(s) cards with contacts, Part 6: Inter-industry data elements”, 2001

[ISO/IEC 7816 -7] International Standardisation Organization: “Identification cards – Integrated circuit(s) cards with contacts, Part 7: Interindustry commands for Structured Card Query Language (SCQL)”, 1999

[ISO/IEC 7816 -8] International Standardisation Organization: “Identification cards – Integrated circuit(s) cards with contacts, Part 8: Security related interindustry commands”, 1999

[ISO/IEC 7816 -9] International Standardisation Organization: “Identification cards – Integrated circuit(s) cards with contacts, Part 9: Enhanced interindustry commands”, 2000

[ISO/IEC 7816 -10] International Standardisation Organization: “Identification cards – Integrated circuit(s) cards with contacts, Part 10: Electronic signals answer to reset for synchronous cards”, 1999

[ISO/IEC 7816 -11] International Standardisation Organization: “Identification cards – Integrated circuit(s) cards with contacts, Part 11: Personal verification through biometric methods”, 2000

[ISO/IEC 7816 -15] International Standardisation Organization: “Identification cards – Integrated circuit(s) cards with contacts, Part 15: Cryptographic Information Application”, 2001

[ISO 10 202-1] International Standardisation Organization: “Financial Transaction Cards –Security Architecture of Financial Transaction Systems using Integrated Circuit Cards, Part 1: Card life cycle”, 1991

[ISO 10 202-2] International Standardisation Organization: “Financial Transaction Cards –Security Architecture of Financial Transaction Systems using Integrated Circuit Cards, Part 2: Transaction process”, 1996

[ISO 10 202-3] International Standardisation Organization: “Financial Transaction Cards –Security Architecture of Financial Transaction Systems using Integrated Circuit Cards, Part 3: Cryptographic key relationship”, 1998

[ISO 10 202-4] International Standardisation Organization: “Financial Transaction Cards –Security Architecture of Financial Transaction Systems using Integrated Circuit Cards, Part 4: Secure Application modules”, 1996

[ISO 10 202-5] International Standardisation Organization: “Financial Transaction Cards –Security Architecture of Financial Transaction Systems using Integrated Circuit Cards, Part 5: Use of algorithms”, 1998

[ISO 10 202-6] International Standardisation Organization: “Financial Transaction Cards –Security Architecture of Financial Transaction Systems using Integrated Circuit Cards, Part 6: Card holder verification”, 1994

[ISO 10 202-7] International Standardisation Organization: “Financial Transaction Cards –Security Architecture of Financial Transaction Systems using Integrated Circuit Cards, Part 7: Key Management”, 1998

[ISO 10 202-8] International Standardisation Organization: “Financial Transaction Cards –Security Architecture of Financial Transaction Systems using Integrated Circuit Cards, Part 8: General principles and overview”, 1998

[ISO/IEC 10 536 -1] International Standardisation Organization: “Identification cards Contactless integrated circuit(s) cards – Closecoupled cards, Part 1: Physical characteristics”, 2000

[ISO/IEC 10 536 -2] International Standardisation Organization: “Identification cards – Contactless integrated circuit(s) cards – Closecoupled cards, Part 2: Dimensions and location of coupling areas”, 1995

[ISO/IEC 10 536 -3] International Standardisation Organization: “Identification cards – Contactless integrated circuit(s) cards – Closecoupled cards, Part 3: Electronic signals and reset procedures”, 1996

[ISO/IEC 10 536 -4] International Standardisation Organization: “Identification cards – Contactless integrated circuit(s) cards – Closecoupled cards, Part 4: Answer to reset and transmission protocols”, 1997

[ISO/IEC 14 443 -1] International Standardisation Organization: “Identification cards – Contactless integrated circuit(s) cards – Proximity cards, Part 1: Physical characteristics”, 2000

[ISO/IEC 14 443 -2] International Standardisation Organization: “Identification cards – Contactless integrated circuit(s) cards – Proximity cards, Part 2: Radio frequency power and signal interface”, 2001

[ISO/IEC 14 443 -3] International Standardisation Organization: “Identification cards – Contactless integrated circuit(s) cards – Proximity cards, Part 3: Initialization and anticollision”, 2001

[ISO/IEC 14 443 -4] International Standardisation Organization: “Identification cards – Contactless integrated circuit(s) cards – Proximity cards, Part 4: Transmission protocol”, 2001

[ISO/IEC 15 6931-1] International Standardisation Organization: “Identification cards – Contactless integrated circuit(s) cards – Vicinity cards, Part 1: Physical characteristics”, 2000

[ISO/IEC 15 6931-2] International Standardisation Organization: “Identification cards – Contactless integrated circuit(s) cards – Vicinity cards, Part 2: Air interface and initialization”, 2000

[ISO/IEC 15 6931-3] International Standardisation Organization: “Identification cards – Contactless integrated circuit(s) cards – Vicinity cards, Part 3: Anticollision and transmission protocol”, 2001

[ISO/IEC 15 6931-4] International Standardisation Organization: “Identification cards – Contactless integrated circuit(s) cards – Vicinity cards, Part 4: Extendend command set and security features”, 1996

[K98] Paul Kocher: „*Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems*”, http://www.cryptography.com/timingattack/paper.html, 1998

[kei] www.keil.com/dd/docs/datashts/intel/ism51.pdf

[KJJ98/I] Paul Kocher, Joshua Jaffe, Benjamin Jun: “*Cryptography Research Q&A on Differential Power Analysis*”, http://www.cryptography.com/dpa/qa/index.html, 1998

[KJJ98/II] Paul Kocher, Joshua Jaffe, Benjamin Jun: "*Introduction to Differential Power Analysis and Related Attacks"*, http://www.cryptography.com/dpa/technical/index.html, 1998

[P00] Philips Semiconductor: *„Short Form Specification",* Philips Semiconductor, http://www.semiconductors.philips.com /acrobat/other/identification/sfs051310.pdf, 2000

[RE02] Wolfgang Rankl, Wolfgand Effing: *"Handbuch der Chipkarten"*, Hanser, 2002

[RH01] Stefan Rüping, Harald Hewel: "*Innovation Letter: Benchmarking*", Infineon Technologies, 2001

[S98] Thomas A. Standfish: „*Data structures in Java*", Addison-Wesley, 1998

[Sun1] Sun Microsystems Inc.: "*Java Card 2.0 Application Programming Interface*", Sun Microsystems Inc., 1998

[Sun2] Sun Microsystems Inc.: "*Java Card 2.0 Language Subset and Virtual Machine Specification*", Sun Microsystems Inc., 1998

[Sun3] Sun Microsystems Inc.: "*Java Card Applet Developer's Guide*", Sun Microsystems Inc., 1998

[Sun4] Sun Microsystems Inc.: "*Java Card 2.2 Development Kit User's Guide*", Sun Microsystems Inc., 2002

[Sun5] Sun Microsystems Inc.: "*Java Card 2.2 Virtual Machine Specification*", Sun Microsystems Inc., 2002

[Sun6] Sun Microsystems Inc.: "*Java Card 2.2 Runtime Environment (JCRE) Specification*", Sun Microsystems Inc., 2002