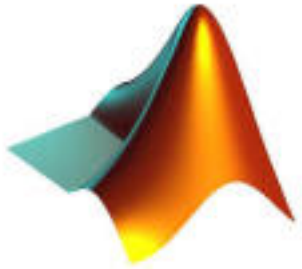


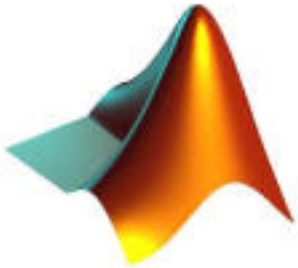
NT-Seminarvortrag am 12. Nov. 2002

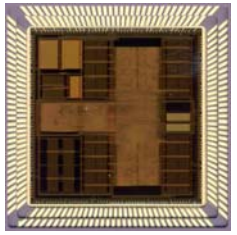
Coding-Style Richtlinien für compilierbares Matlab

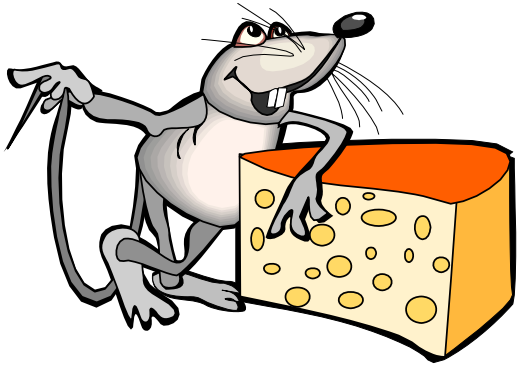
von Gordon Cichon

Inhalt

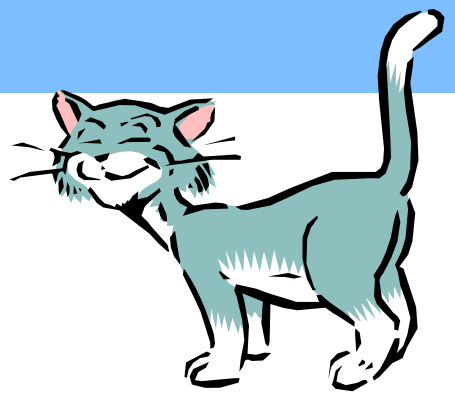
- Überblick über MOUSE Compiler
- Propagation von
  - Konstanten
  - Typen
  - Vektor-Größen (Shape)
- Parallelisierung
  - Vektorisierung
  - Do‘s and Don‘ts
  - Ein einfaches Rezept
- Die DVB-T Teststrecke







CATS



CATS (Marcus Bronzel)

(Concepts for Application Tailored Signal processors)

- integrierte, computerunterstützte Prozessorentwurfsplattform
- Entwicklung applikationsspezifischer Prozessoren
- angepaßte Akzeleratoren
- integrierte Softwareentwicklungsplattform

MOUSE (Teilprojekt von CATS)

(Matlab Optimized Universal Signal-processing Environment)

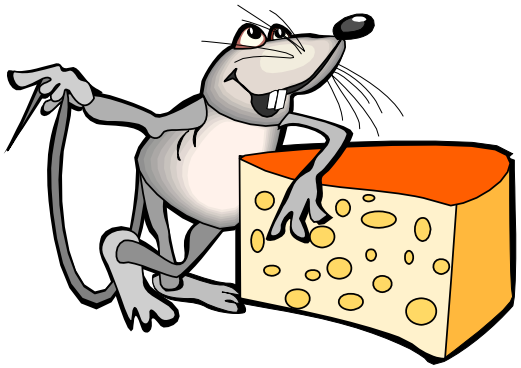
MOUSE

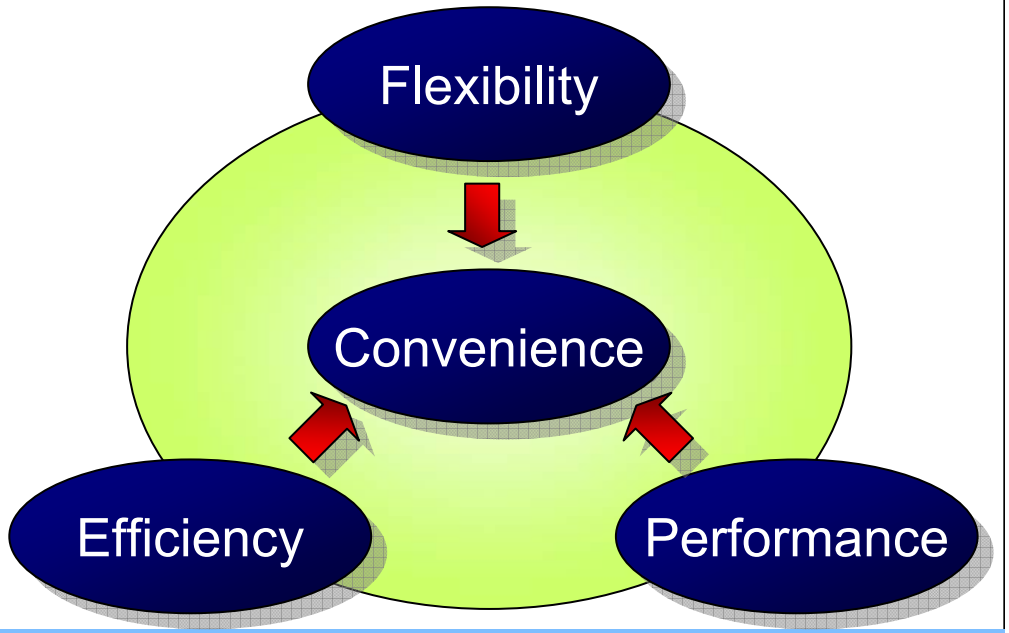
The ***MOUSE*** project is dedicated to provide

- an easily programmable platform (Matlab)

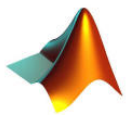
that combines

- the cost and power efficiency of an DSP with
- the flexibility of a FPGA and
- the performance of an ASIC.





Compiler Architecture



Matlab Source

C Source

Matlab

Octave

Frontend

Vectorize Optimizations

Intermediate representation

Run as C

Utrecht Centerpart

Dresden Centerpart

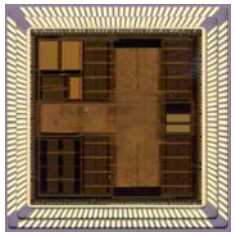
Optimizations

Intermediate representation

Run as C

Backend:
- instruction selection
- scheduler
- register allocation
- code compression

Binary program



Processor

Der MOUSE-Compiler

Leistungen von MOUSE

- Compilierung von Matlab Programmen zu DSP-Code
- Eingeschränkter Support von intrinsischen Funktionen
- Parallelisierung
- Tradeoff zwischen Implementierungsaufwand und Performance

Anwendungen

- Performanceabschätzung von Signalverarbeitungsalgorithmen
- Implementierung von Systemen auf dem M5 DSP

Randbedingungen

- Rumpf der Sprachkonstrukte von Matlab, wie in Octave (http://www.octave.org)
- Vordefinierte Funktionen müssen als Matlab Quellcode vorliegen
- Code sollte vektorisierbar sein

Das Octave Frontend

OpenSource Matlab Clone

- Interpreter
- Dient MOUSE als Compiler-Frontend
- Kann ohne User-Interface laufen (=> Standalone Programme)
- Beherrscht nicht alle intrinsischen Funktionen von Matlab (s. Dokumentation)
- Keine Matlab-Klassen
- Erweiterte Syntax (Inkrement und Dekrement, Short-Circuit Boolean, flexibler Zeilenumbruch)

Dokumentation: http://www.octave.org/doc/octave_toc.html

Implementierungsaufwand und Performance

Philosophie: Jedes gültige Matlab-Programm sollte auf dem M5-DSP laufen

Aber: Es gibt gravierende Unterschiede in der Performance:

- Datentyp-Erkennung
- Dynamische Speicherverwaltung
- Parallelisierung

Propagation von Information

Im Programm steht:

- a = 5
- b = fread (da_conv, 128, “short”)
- c = [a ; b]
- d = c + 5.0;
- e = zeros (a,2)
- f = 2*b-1;

Lesen vom DA-Konverter

Schlecht: Vermischung von bekannten und unbekannten Werten

Compiler leitet ab:

- a: Typ: int8, Wert: 5
- b: Typ: vector of int16, Shape: 128x1 Wert: unbekant
- c: Typ: vector of int16, Shape: 6x1 Wert: unbekannt
- d: Typ: vector of int16, Shape 6x1 Wert: unbekannt
- e: Typ: vector of bool, Shape 5x2, Wert: 0
- f: Typ: vector of int16, Shape: 128x1, Wert unbekannt

for α in 1:128
f(α) = 2*b(α) -1

Propagation von Information (Forts.)

Im Programm steht:

- if a == 5
  g = 17
  else
  g = [1+5i 3i ; 17 59+3i]
  end
- if c(1) == 5
  h = 17
  else
  h = [1+5i 3i ; 17 59+3i]
  end

Compiler leitet ab:

- g: Typ: int8, Wert: 17
- h: Typ: matrix of complex, Shape: unbekannt

Folge der Vermischung von bekannten und unbekannten Werten

Datentyp-Erkennung

Falls keine andere Information vorliegt, muß der Compiler für jede Variable den allgemeinsten Datentyp annehmen:

- komplexe Gleitkommazahlen mit doppelter Genauigkeit

Wie erhält der Compiler weitere Information:

- Durch die Initialisierung und Benutzung der Variablen im Programm
- Explizite Spezifikation in einer separaten Datei

Initialisierung von Werten

- Compiler stellt einen Initialwert bei der Initialisierung von Variablen fest:

Datentyp	Initialisierung	Einlesen mittels fread
Boolean (1 bit)	x = 0 oder x = 1	
Integer (1, 2, 4 byte)	x = 15	char, short, int, long
Float (4, 8 byte)	x = 2.5	float, double
Complex (8, 16 byte)	x = 1+1i	

- Abbildung von Gleitkomma auf Festkomma:
  Testläufe des Programms mit realen Daten ergeben Werteprofil für Variablen (Beschreibung s. http://www.iss.rwth-aachen.de/Projekte/Tools/FRIDGE/fridge.html)

Propagation von Variablentypen

- Datentyp wird durch aritmetische Operationen propagiert.
- Charakteristische Operationen:

Ergebnistyp	Operation
Boolean	Vergleich (==, ~=, <, >, etc.), Logik (&, |)
Integer	floor, ceil
Float	/, real, imag
Complex	sqrt

- Information über Dimension von Vektoren und Matrizen (Shape) wird ebenfalls propagiert

Dynamische Speicherverwaltung

- Statische Speicherverwaltung zur Compile-Zeit wesentlich effizienter als dynamische Speicherverwaltung auf DSP.
- Bitte Matlab-Dokumentation beachten:

Getting Started

Preallocation

If you can't vectorize a piece of code, you can make your `for` loops go faster by preallocating any vectors or arrays in which output results are stored. For example, this code uses the function `zeros` to preallocate the vector created in the `for` loop. This makes the `for` loop execute significantly faster.

```
r = zeros(32,1);
for n = 1:32
    r(n) = rank(magic(n));
end
```

`R = zeros (32,1);`
Legt die Dimension von r fest

Without the preallocation in the previous example, the MATLAB interpreter enlarges the `r` vector by one element each time through the loop. Vector preallocation eliminates this step and results in faster execution.

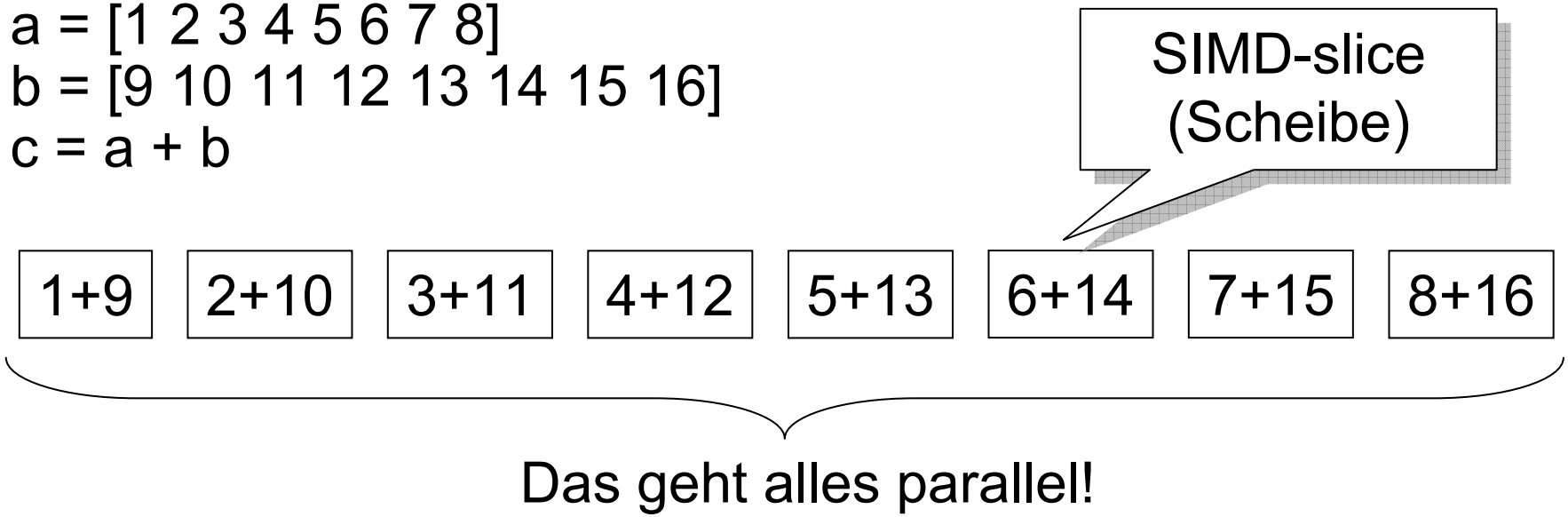
Vectorization | Function Handles

Dynamische Speicherverwaltung (Teil 2)

- Speicher im DSP ist knapp
- Unbenutzte Variablen können mit **`clear`** vorzeitig freigeben werden.

Parallelisierung

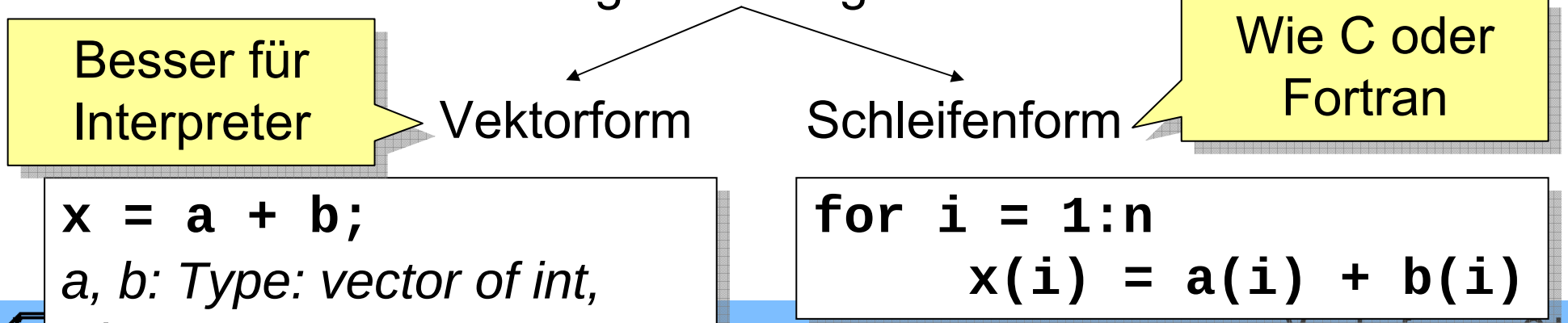
- Der M5-DSP ist eine sog. SIMD-Maschine (single instruction, multiple data)
- D.h. der Prozessor kann eine gleichartige Rechenoperation of mehrere Elemente eines Vektors gleichzeitig ausführen
- a = [1 2 3 4 5 6 7 8]
  b = [9 10 11 12 13 14 15 16]
  c = a + b



- Parameter **s** = Anzahl der slices, bei uns 8 oder 16

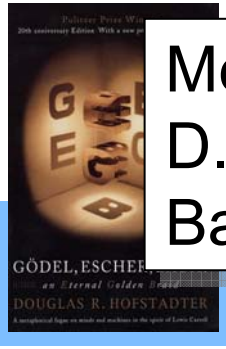
Wie nutze ich die Parallelität im Programm?

- Bei Schleifen muß der gleiche Code mehrmals hintereinander ausgeführt werden.
- Falls der Code bei jedem Durchlauf mit jeweils verschiedenen Daten arbeitet, werden jeweils s Iterationen der Schleife gleichzeitig parallel auf den s slices durchgeführt.
- Dieses Vorgehen des Compilers heißt Vektorisierung.
- Erfunden wurde es in den 1970er Jahren von Herren Allen und Kennedy (Fa. Cray).
- Für den Vektorisierer gleichwertig:



Was muß der Compiler dabei tun?

- Um eine Schleife vektorisieren zu können, muß der Compiler **beweisen** können, daß keine Datenabhängigkeiten zwischen den Instanzen des Schleifenrumpfes bestehen.
- Der **Beweis** muß geführt werden, daß für eine sog. diophantische Gleichung über die Indexausdrücke keine Lösung existiert.
- Z.B.: Für **x(k)=x(k+1)+1** muß bewiesen werden, daß die Gleichung $k_i = k_j + 1$ keine Lösung unter der Nebenbedingung $k_i < k_j$ besitzt.
- Gemäß Gödel bzw. Turing kann eine solche Gleichung nur für Spezialfälle gelöst werden. Z.B. ist **x(z^n)=x(a^n+b^n)** genau dann vektorisierbar, falls die Fermatsche Vermutung wahr ist.
- Also: Bitte erwartet nicht zu viel vom Compiler!



Mehr zu diesem Thema:
D.Hofstadter, Gödel-Escher-Bach, Basic Books, NewYork, 1979

Wie kann ich den Compiler unterstützen?

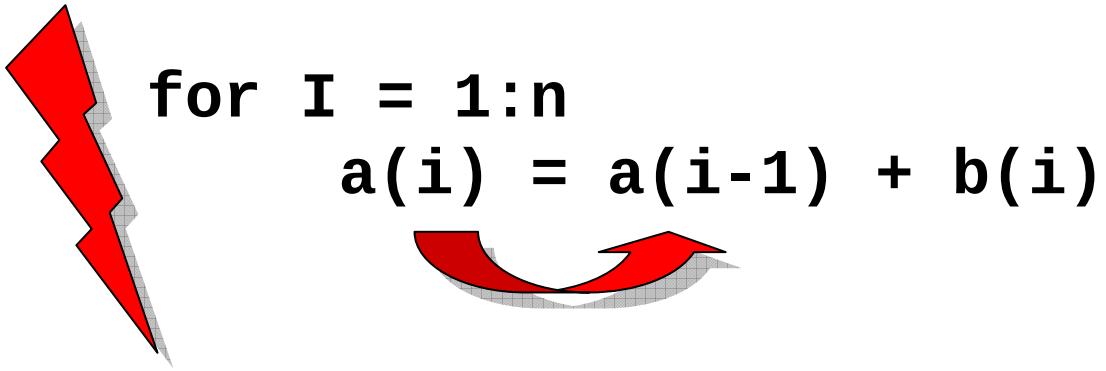
- Besonders sicher sind elementweise Arithmetische Operationen: +, -, .*, ./ usw.
- Beim Schreiben von Schleifen
  - Aussuchen einer Variable (z.B. i), über die Vektorisiert werden soll.
  - Auf möglichst einfache Indexausdrücke achten:
    Am besten ist *offset*±i, wobei *offset* ein beliebiger Ausdruck ist,
    der vor dem Schleifenstart berechnet werden kann.
  - Vorteilhaft ist, wenn die Anzahl der Schleifendurchläufe durch die Zahl der slices **s** teilbar ist. Ansonsten muß der Compiler zwei Schleifen daraus machen: eine schnelle SIMD-Schleife, und eine langsame normale Schleife, die den Rest Durchläufe abarbeitet.
  - Möglichst lange Vektoren verarbeiten. (Aber nicht zu lange: der DSP wird nur etwa 64K-256KByte Speicher haben)

Was sollte nach Möglichkeit vermieden werden?

- Lange, geschachtelte Schleifen mit komplizierten

  Nebenbedingunen und Seiteneffekten. Je einfacher, desto besser.

- Skalarprodukte sind schwierig zu parallelisieren.
- Keine sonstigen Aktivitäten in der Schleife
  (z.B. noch irgendetwas mitzählen)
- Schleifen mit linearen Ketten von Datenabhängigkeiten:



Eine einfache Parallelisierungstechnik

- Z.B. FIR $$y_k = \sum_{i=0}^{I} b_i x_{k-i}$$
- Aufschreiben des Algorithmus als Tabelle:

Parallelität →

Zeit ↓

$b_I x_0$	$b_I x_1$	$b_I x_2$	…	$b_I x_{N-I-2}$	$b_I x_{N-I-1}$	$b_I x_{N-I}$
$b_{I-1} x_1$	$b_{I-1} x_2$	$b_{I-1} x_3$	…	$b_{I-1} x_{N-I-1}$	$b_{I-1} x_{N-I}$	$b_{I-1} x_{N-I+1}$
$b_{I-2} x_2$	$b_{I-2} x_3$	$b_{I-2} x_4$	…	$b_{I-2} x_{N-I}$	$b_{I-2} x_{N-I+1}$	$b_{I-2} x_{N-I+2}$
…	…	…	…	…	…	…
$b_2 x_{I-2}$	$b_2 x_{I-1}$	$b_2 x_I$	…	$b_2 x_{N-4}$	$b_2 x_{N-3}$	$b_2 x_{N-2}$
$b_1 x_{I-1}$	$b_1 x_I$	$b_1 x_{I+1}$	…	$b_1 x_{N-3}$	$b_1 x_{N-2}$	$b_1 x_{N-1}$
$b_0 x_I$	$b_0 x_{I+1}$	$b_0 x_I$	…	$b_0 x_{N-2}$	$b_0 x_{N-1}$	$b_0 x_N$
$\Sigma: y_I$	**$\Sigma: y_{I+1}$**	**$\Sigma: y_{I+2}$**	**…**	**$\Sigma: y_{N-2}$**	**$\Sigma: y_{N-1}$**	**$\Sigma: y_N$**

Eine einfache Parallelisierungstechnik (Teil 2)

- Umwandlung der Tabelle in eine Schleife

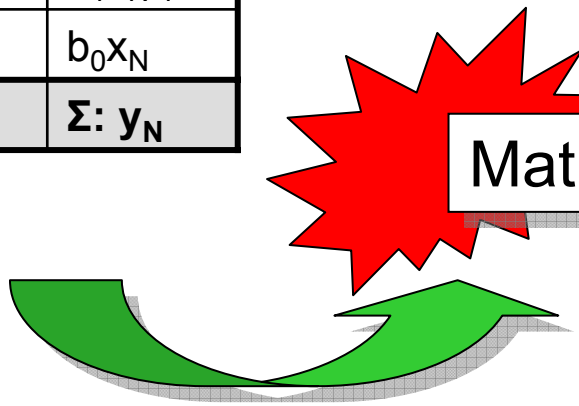
$b_I x_0$	$b_I x_1$	$b_I x_2$	…	$b_I x_{N-I-2}$	$b_I x_{N-I-1}$	$b_I x_{N-I}$
$b_{I-1} x_1$	$b_{I-1} x_2$	$b_{I-1} x_3$	…	$b_{I-1} x_{N-I-1}$	$b_{I-1} x_{N-I}$	$b_{I-1} x_{N-I+1}$
$b_{I-2} x_2$	$b_{I-2} x_3$	$b_{I-2} x_4$	…	$b_{I-2} x_{N-I}$	$b_{I-2} x_{N-I+1}$	$b_{I-2} x_{N-I+2}$
…	…	…	…	…	…	…
$b_2 x_{I-2}$	$b_2 x_{I-1}$	$b_2 x_I$	…	$b_2 x_{N-4}$	$b_2 x_{N-3}$	$b_2 x_{N-2}$
$b_1 x_{I-1}$	$b_1 x_I$	$b_1 x_{I+1}$	…	$b_1 x_{N-3}$	$b_1 x_{N-2}$	$b_1 x_{N-1}$
$b_0 x_I$	$b_0 x_{I+1}$	$b_0 x_I$	…	$b_0 x_{N-2}$	$b_0 x_{N-1}$	$b_0 x_N$
$\Sigma: y_I$	**$\Sigma: y_{I+1}$**	**$\Sigma: y_{I+2}$**	**…**	**$\Sigma: y_{N-2}$**	**$\Sigma: y_{N-1}$**	**$\Sigma: y_N$**

Zeilen

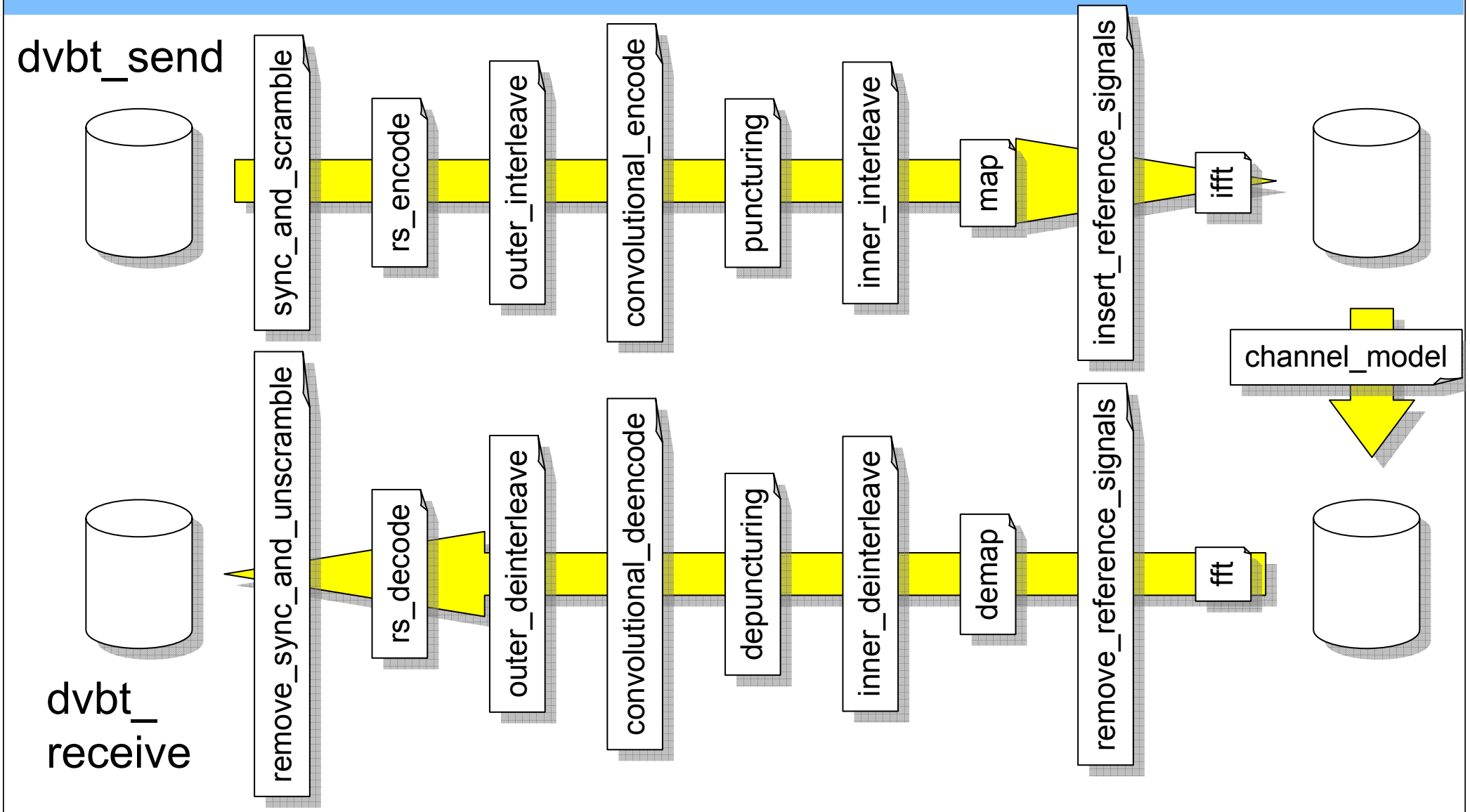
Spalten

```
for r = 0:I
  for c = I:N
    y(c)=y(c) + …
      b(I-r)*x(r+c-I)
```

Matlab Programm



Beispiel: DVB-T Strecke



DVB-T Teststrecke (Teil 2)

- Projektierte Fertigstellung: Dez. 2002
- Testbeispiel: erste 16K eines MPEG-Filmes
- Algorithmen implementiert als Matlab-Funktionen, die jeweils ein MPEG-MUX-Packet bzw. Ein OFDM-Symbol als Parameter bekommen
- In der Mitte: Umpaketierung
- Code läuft auf Matlab und Octave
- Einbettung in die Compiler-Testbench
- Weitergabe des Codes nach Utrecht

Derzeitige Unvollkommenheiten:

- Testrecke läuft im 2K-Mode
- RS fügt im Sender 0 als Paritybytes an, Empfänger ignoriert Paritybytes
- Outer Interleaver noch nicht Spec-Konform
- Noch kein Viterbi, nur “Convolutional Decoder”
- Demapper ist inverser Mapper: sucht komplexe Vektoren in Tabelle
- Referenz-Signale in Arbeit
- FFT/iFFT noch nicht angefangen
- Globale Variable noch nicht in Structs zusammengefasst