Um 14.17 wurde die Negerkusspackung geöffnet.

Der benutzte Testrechner, auf dem später der Emulater installiert werden soll, beseitzt einen Oszillator mit einer Taktfrequenz von 80 MHz, die für den Prozessor auf 40 MHz heruntergeteilt wird. Diese Frequenz ist für den Emulator zu schnell, so daß durch Austausch des Oszillators mit einer Frequenz von 60, 40, 20, 10 oder 6 MHz der Rechner so weit verlangsamt wird, bis der Emulator mithalten kann.

Lars Werner wird mit Hilfe eines Oszilloskopes die Schnittstelle zwischen IU unf FPU analysieren, um den genauen Signalverlauf bei FPU-Befehlen zu bestimmen. Hier war ein großes Problem, da dies in den Handbüchern nur unvollständig und teilweise gegensätzlich beschreiben ist.
Burkhard Reike schreibt hierfür ein kleines Programm, das in einer Endlosschleife FPU-Befehle ausführt, so daß man die Signale am Prozessor messen kann.

Burkhard hielt einen kurzen Vortag über die Library und der C-Schnittstelle zur Programmierung des Coprozessors.

Algrorithmus-Gruppe : "Wir haben gut die hälfte fertig, benötigen noch 2 bis 3 Wochen"

Gesamtmodell : Der Float-Addierer ist jetzt integriert, es müssenaber noch der Subtrahierer, Multiplizierer und der Dividierer ersetzt werden.

Interface-Gruppe : Die Simulation des Interfaces ist ok. die Synthese bereitet noch Probleme.

CVS-Gruppe : Die Verwendung des CVS-Tools bringt nach Ansicht der Gruppe keine Vorteile, da die Verwaltung zu Aufwendig ist.

Das Gesamtmodell sollte im mehrere kleine Teilstücke aufgeteilt werden, um es mit dem CVS-Tool besser verwalten zu können.

spline.h:

```
class Spline{

public:
  Spline();
  Spline(Spline& s);
  ~Spline();
  Spline& operator=(const Spline& s);

  // Members for manipulating the point-set
  int addPt(float x, float y);      // add point (x,y) to set
  int delPt(float x, float y);      // delete point (x,y) from set
  int clrPts();                     // clear point-set

  // SFU-related members (optional, not necessary for calculation)
  int calc();                       // send points to SFU

  // Members for accessing information

    // get xy-coordinates of point No. <nr>
  int getPt(int nr, float *x, float *y) const;

    // get point No. <nr> (as array of size 2)
  const Point& getPt(int nr) const;

    // get range of polynome No. <nr>
  int getRange(int nr, float *x1, float *y1, float *x2, float *y2) const;

    // get coefficients of polynome No. <nr>, which is a*x^3+b*x^2+c*x+d
  int getCoeff(int nr, float *a, float *b, float *c, float *d);

    // return coefficient-array No. <nr>
  const Coeffs& getCoeff(int nr);

  // Misc

    // formated printing of Spline
  void print(ostream& out = cout) const;

    // print out plot-command for Maple(tm) which plots the spline
  int printMaplePlot(ostream& out = cout);

};

extern ostream& operator<<(ostream& s, Spline& x); // just like print
```

```
extern const char* SplineErrText(int nr);      // returns Errortext
```

cspline.h:

```
/* add point (x,y) to list of points */
extern int SplAddPt(float x, float y);

/* remove point (x,y) from list of points */
extern int SplDelPt(float x, float y);

/* clear list of points */
extern int SplClrPts();

/* send points to SFU and start calculation */
extern int SplCalc();

/* get coordinates of point No. <nr> */
extern int SplGetPt(int nr, float* x, float* y);

/* get coefficients of spline-polynome No. <nr> */
extern int SplGetCoeff(int nr, float *a, float *b, float *c, float *d);

/* get range of spline-polynome No. <nr> */
extern int SplGetRange(int nr, float *x1, float *y1, float *x2, float *y2);

/* return errortext No. <nr> */
extern char* SplErrText(int nr);

/* formated printout of information about the spline on standard-output */
extern void SplPrint();

/* printout of Maple(tm) plot-command on standard-output */
extern int SplMaplePlot();
```

```
#include <spline.h>

main(){
  Spline s;
  float a,b,c,d,x1,y1,x2,y2,x,y;

  s.addPt(3,3.5); s.addPt(1,1.1); s.addPt(0,0.5); s.addPt(2,2.3);

  s.getPt(0, &x, &y);
  cout << "(" << x << "," << y << ")\n";
  cout << "(" << s.getPt(0)[0] << "," << s.getPt(0)[1] << ")\n";
  s.getRange(2, &x1, &y1, &x2, &y2);
  cout << x1 << " " << x2 << " " << y1 << " " << y2 << "\n";

  s.calc();

  Spline t(s);
  t.addPt(-2.3,0.9);
  t.getCoeff(0, &a, &b, &c, &d);
  cout << s.getCoeff(0)[3] << "\n";

  Spline u,v;

  u = v = t;

  t.print();
  u.clrPts();
  cout << u << v;
  t.printMaplePlot();

  cout << SplineErrText(t.delPt(100,100)) << "\n";

  v.calc();
}

#include <stdio.h>
#include <cspline.h>

main(){
  float x,y,x1,y1,x2,y2,a,b,c,d;

  SplAddPt(1.2,3); SplAddPt(3,4.5); SplAddPt(-3.89,-0.9); SplAddPt(1,1);
  SplPrint();
  SplDelPt(1,1);
  SplPrint();
  SplGetPt(1,&x,&y);
  printf("point %f %f\n",x,y);
```

```
  SplCalc();
  SplGetRange(0,&x1,&y1,&x2,&y2);
  printf("range %f %f %f %f\n",x1,y1,x2,y2);
  SplGetCoeff(0,&a,&b,&c,&d);
  printf("coeff %f %f %f %f\n",a,b,c,d);
  SplAddPt(-8.91, 10.2);
  SplGetCoeff(0,&a,&b,&c,&d);
  printf("coeff %f %f %f %f\n",a,b,c,d);
  SplMaplePlot();
  SplClrPts();
  printf("%s\n",SplErrText(SplCalc()));
}
```