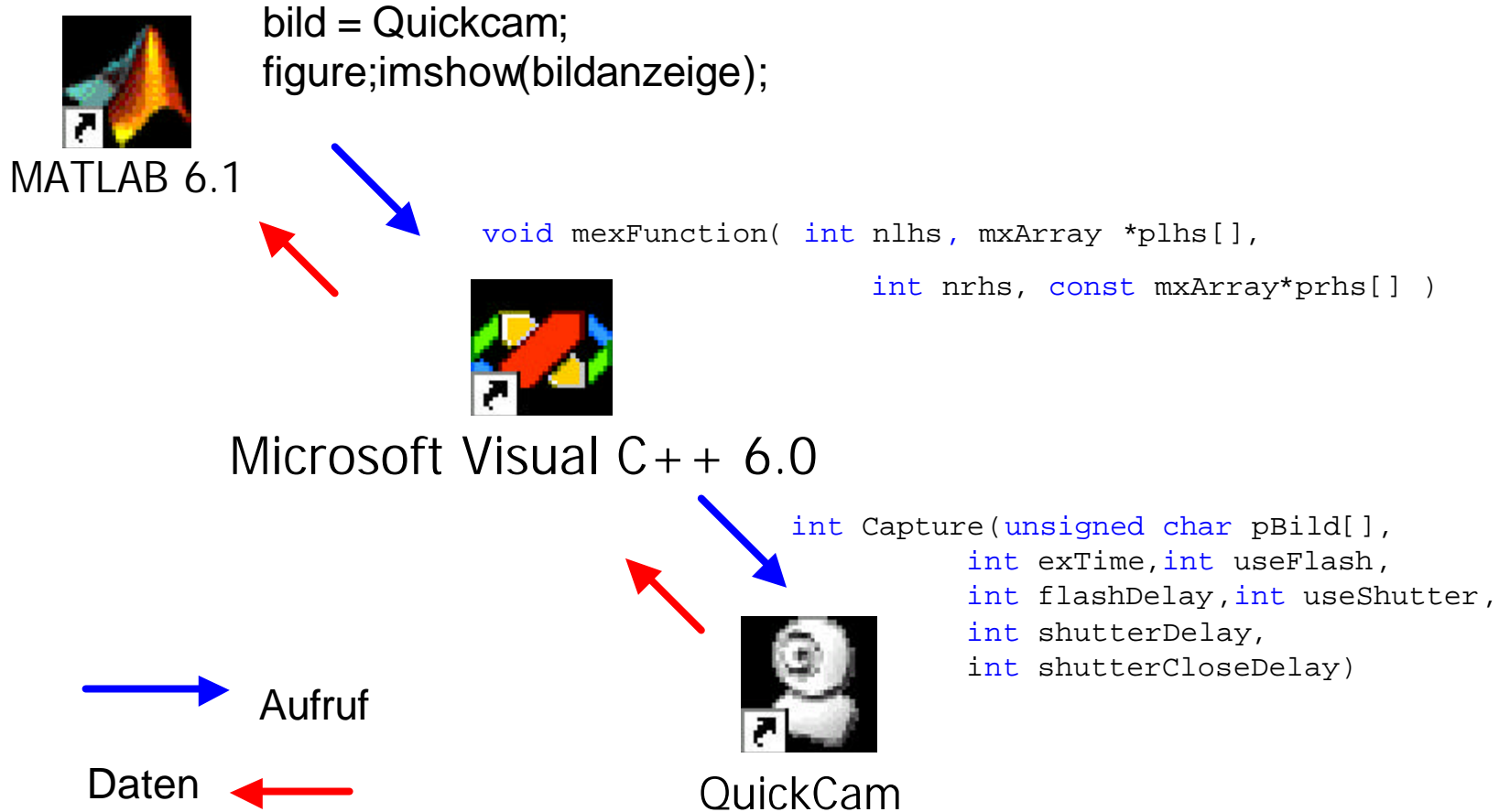


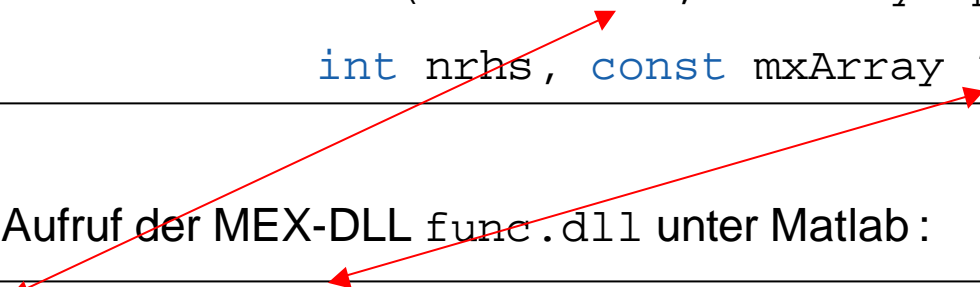
Hardwareeinbindung in Matlab



MEX-Files

Interface-Funktion mexFunction im C-Code

```
void mexFunction ( int nlhs, mxArray *plhs[],  
                  int nrhs, const mxArray *prhs[] )
```



Der Aufruf der MEX-DLL `func.dll` unter Matlab:

```
[u, v] = func (a);
```

Anlegen einer Matrix und Rückgabe der Werte:

```
plhs[0] = mxCreateDoubleMatrix(YSIZE, XSIZE, mxREAL);  
double* z = mxGetPr(plhs[0]);
```

Für einfache Berechnungen usw ist das ok, Hardware benötigt mehr...

MEX-Files

```
// Code-Fragment MEX-File Einbindung
#include <stdio.h>... // C-Dinge
#include "mex.h" // MatLab
#include "PimMegaApiUser.h" // PixeLINK - API (Fire-
Wire-Camera)

HANDLE hImager; // Global Imager Object
int dims[] = {0,0,640,480}; // Dimensionsfeld für Aufnahme
unsigned char *pBild; // Zeiger auf das Bild

mxArray* mBILD;
```

MEX-Files

Left-hand-side

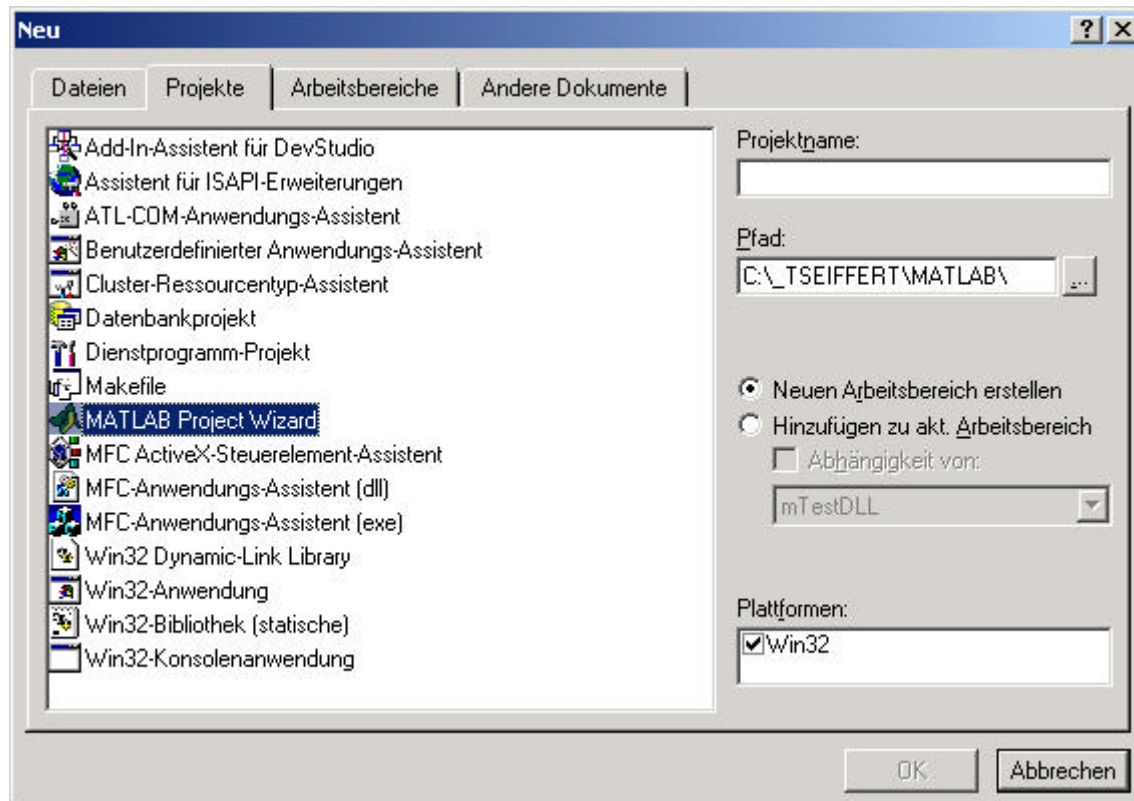
right-hand-side

```
void mexFunction( int nlhs, mxArray *plhs[],
int nrhs, const mxArray*prhs[] )
{
    if (nrhs == 0) { /* Check number of arguments */
        mexErrMsgTxt("Mindestens eine Eingabe
erforderlich!\n");
    }
    /* Create the picture matrix for the return argument
*/
    mBILD = mxCreateNumericArray(2,
bilddims,mxUINT8_CLASS, mxREAL);
    /* Assign pointers to the frame */
    pBild = (unsigned char*)mxGetPr(mBILD);
    //In den Speicherbereich Grabben
    nRetVal = Capture(pBild, params[0 .. 5]);

    if (!nRetVal) { // Rückgabe ist Bildmatrix
        plhs[0] = mBILD;
        pReturn = mxGetPr(plhs[0]);
    }
}
```

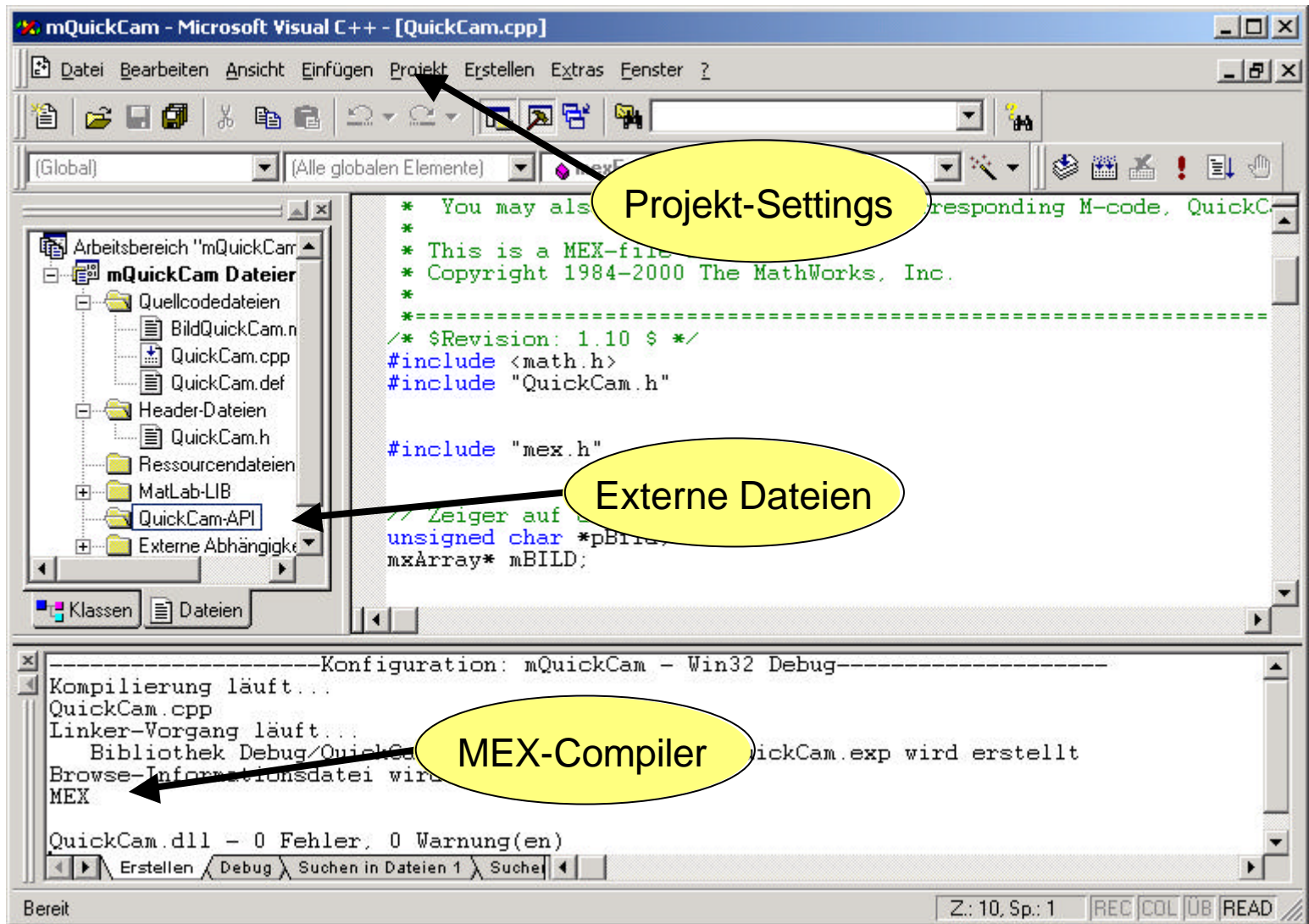
MEX-DLL – der einfachste Weg:

In matlab\bin\win32 das Kommando „mex –setup“ ausführen



Dann kann unter VC Projekte neue (u.a.) MEX-DLL's erstellt werden

Visual-Studio 6.0 Einstellungen für MEX-DLL's



MEX-DLL – „von Hand“

The screenshot shows the Microsoft Visual C++ IDE with the following elements:

- Project Settings:** A yellow oval points to the menu path: `Projekt -> C/C++ -> Präprozessor` with the value `„C:\Programme\matlab6p1\extern\include“`. Another yellow oval points to `Projektsettings -> Linker -> Eingabe` with the value `„C:\Programme\matlab6p1\extern\lib\win32\microsoft\msvc60“`.
- Externe Dateien:** A yellow oval points to the 'Externe Abhängigkeiten' (External Dependencies) folder in the Solution Explorer, which contains 'MatLab-LIB' and 'QuickCam-API'.
- MEX-Compiler:** A yellow oval points to the 'MEX' entry in the 'Bibliothek' (Library) list of the linker options.
- Code:** The main editor shows C++ code with preprocessor directives: `#include <math.h>`, `#include "QuickCam.h"`, and a comment `// Zeiger auf das Bild` followed by `unsigned char *pBild;` and `mxArray* mBILD;`.
- Output Window:** Shows the compilation process: `Kompilierung läuft...`, `QuickCam.cpp`, `Linker-Vorgang 1`, `Bibliothek De`, `Browse-Informati`, and `MEX`.

Reihe offener Fragen zur Verwendung von mex-Files

- Mehrfachaufruf der DLL, globale Variablen
- Speicherfreigabe
- C++, COM – Mechanismen in mex-Files
- DLL's und LIB's für den MEX-Compiler?

Globale Variable ect.

M-File:

```
ans = QuickCam(,init');  
ans = QuickCam(,gain',123);  
bild = QuickCam(,capture',640,480,0.5);
```

C-File:

```
void mexFunction( int nlhs, mxArray *plhs[],  
                 int nrhs, const mxArray*prhs[] )  
{ ...  
  // Initialisierung der Kamera  
  if (!strcmp(strupr(input_buf), 'init')) {  
      InitCam();  
  }  
  
  // Initialisierung der Kamera  
  if (!strcmp(strupr(input_buf), 'capture')) {  
      CaptureFrame();  
  }  
  
  ... }
```

Speicherfreigabe bei Mehrfachaufruf

M-File: bild = QuickCam;

C-File:

```
void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray*prhs[] )
{ ...
// Speicher fürs Bild anlegen
mBILD = mxCreateNumericArray(3, dims, mxUINT8_CLASS, mxREAL);
... }
```

Bei Mehrfachaufruf des M-Files wird der angelegte Speicher nicht mehr freigegeben.

COM-Mechanismen

Client:

```
void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray*prhs[] )
{ ...
    // hier sind keine COM Erweiterungen möglich...

    // gemeinsamen Speicherbereich verbinden
    qcptr = (void **)MapViewOfFile(s_hDibFile,
                                   FILE_MAP_READ|FILE_MAP_WRITE, 0, 0,
    lSize);
    ... }
```

Hardwareeinbindung in Matlab

COM-Mechanismen

Server:

```
// the following defines the connection point interface pointer

CComObject<CDriver>* gpDriver;

int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
    //Initializes the COM library
    CoInitialize(NULL);          .... // Nachricht pollen
    // Main message loop:
    while (GetMessage(&msg, NULL, 0, 0)) {
        ::PeekMessage (&msg, NULL, WM_USER, WM_USER+99, PM_REMOVE);
        if (msg.message == WM_USER)
            switch (msg.wParam )          {
                case QC_MSG_INIT
s_hDibFile = OpenFileMapping(FILE_MAP_WRITE, FALSE, "mQuickCam ");
...

```

Code-Samples für die Einbindung

- **mQuickCam:**
MEX_File das über Shared Memory die Bilder der Logitech Kamera nach Matlab einliest.
- **QuickCamServer**
Dazu muss der Server gestartet sein, der den gemeinsamen Speicherbereich anlegt und nachdem die Windows-Nachricht für das Grabben gekommen ist das aktuelle Bild in den Speicherbereich legt.
- **QuickCamCOM**
MEX-File mit der COM-Einbindung der Logitech-Kamera. COM-Framework funktioniert, die Kamera basiert jedoch auf ActiveX, das ein Handle zum Window brucht. In diesem Fall wäre eine direkte Einbindung der Kamera in das Fenster sinnvoll. (Siehe dazu Vortrag Mhissmann)
- **ocvGrab**
Bilder der Kamera über "Open Source Computer Vision Library" von Intel. Hier werden keine Logitechspezifischen Funktionen verwendet. Das Grabben des Bildes basiert auf Video for Windows.