BioSpec® / MedSpec® / PharmaScan®

MRI/MRS

ParaVision® Programming Course

3-7 April 2006

Bruker BioSpin MRI GmbH

Rudolf-Plank-Str. 23

D-76275 Ettlingen

www.bruker-biospin.de

Table of Contents

Programming in *ParaVision*®

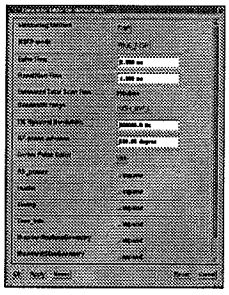
ParaVision Programming Course

Ettlingen, April 3-7, 2007

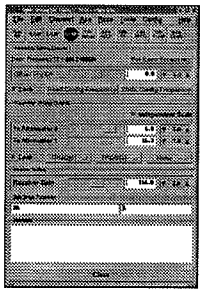
Franek Hennel

ParaVision Overview

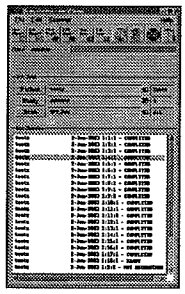
Parameter editor



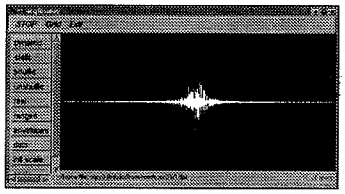
Spectrometer control

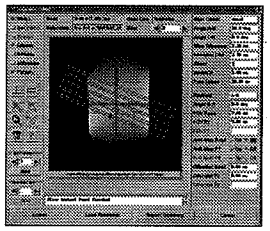


Scan control



Acquisition

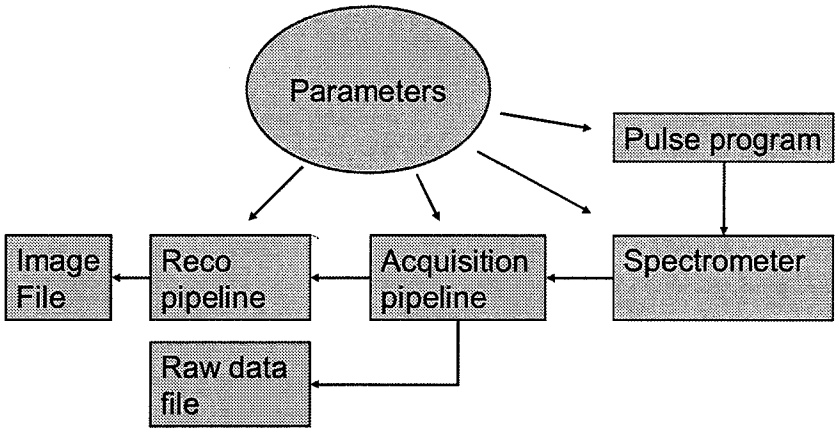




Geometry editor

ParaVision Overview, cont.

- Workflow



Openness of *ParaVision*

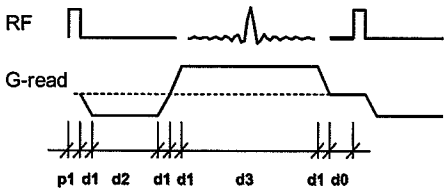
The user can access:

- Timing control
- Parameters in memory
- Parameter files
- Raw data files
- Image files
- Data flow

Timing control

- Pulse program

```
d0 fq1:f1
d8 gatepulse 1
p1:sp1 ph0
d1 grad{(-50)|(0)|(0)}
d2
d1 groff
d1 grad{(50)|(0)|(0)}
50u REC_ENABLE
ADC_INIT(ph0, ph1)
d3 ADC_START
d1 groff
```



Pulse Program Commands

Fixed delays
Variable delays
Incrementable delays
Fixed RF pulses
Variable length pulses
Pulse shapes
Power lists (var. flip angles)
Gradient amplitudes
Gradient ramps
Gradient shapes
Gradient functions (steps)
Frequency lists
Phase lists
...

Access to parameters

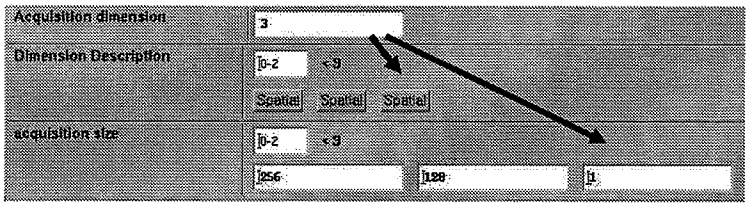
Parameter memory can be accessed from:

- Shell command line
- Shell script (marco)
- C-code
  - Automations
  - Methods

ParaVision Parameter Properties

Parameters have

- Types (int, double, user-defined, ...)
- Attributes (visiblity, etc.)
- Dynamic array sizes
- Relations (reaction procedures)



Access to parameters from shell

- Command line

```
/home/xy> pvcmd -set pvScan NR 100
```

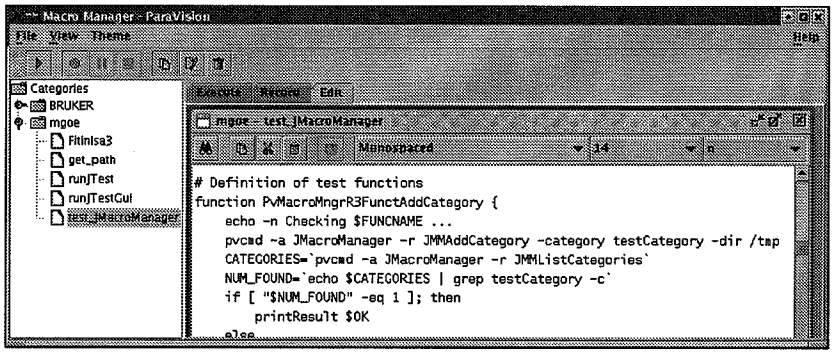
- Shell script

```
#! /bin/bash
path=`pvcmd -a pvScan -r pvDsetPath -path PROCNO`
chmod  -R u+w $path
$XWINNMRHOME/prog/bin/freco -r -n -p -d $path
pvcmd -a pvScan pvDsetSetScanStatus
chmod  ugo-w $path/reco
```

Macros

- Shell scripts placed in .../curdir/<user>/ParaVision/macros are called MACROS.
- Macro Manager's role:
  - Executing macros
  - Recording macros
  - Copying, Deleting
  - Editing
  - Output viewing

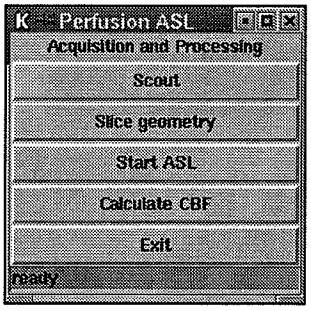
Macro Manager



- Other tools:
  - Command Observer
  - Command Sender

Macro examples

- Any sequence of PV commands can be programmed
- Graphic interfaces may be added with Tcl-Tk

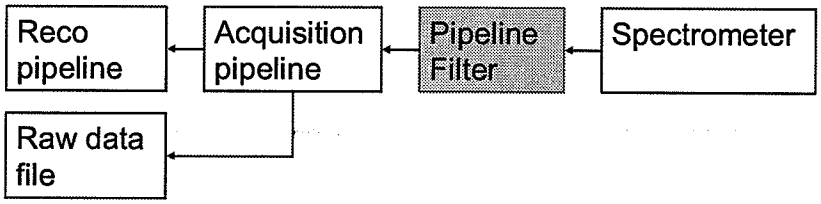


Automations

- Automations are simple c-programs which:
  - Use simplified c-syntax based on c-macros
  - Get parameter values from memory
  - Set parameter values in memory
  - Send PV commands
- Simmilar funtion to macros
- Advantages:
  - Features of c-language, e.g. maths,
  - Faster access to parameters, esp. arrays

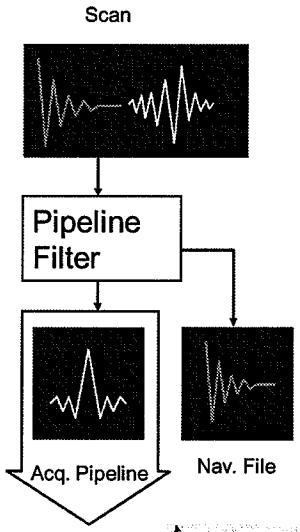
Pipeline Filters

- Pipeline Filters are special Automations which:
  - Access the data on-line
  - Modify data contents and size



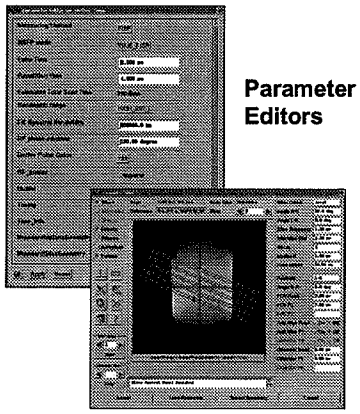
Pipeline Filters: Applications

- Data modification:
  - Offset demodulation,
  - Resampling
  - Non-standard sorting
- Discarding data segments
  - Navigators
- Non-standard accumulation
  - K-space weighting
- Synthesising data
  - "Key-hole" strategies



ParaVision Methods (PVM)

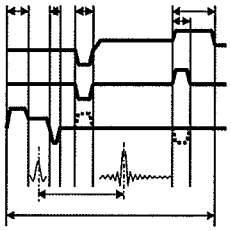
- High-level Interface to Acquisition Parameters



Parameter Editors

METHOD

EXPERIMENT



PVM Features

- Methods are programmed in C
- Independent sources
- Independent binary (one file/method)
- Method-specific parameters can be defined
- Global parameters can be used
- Mechanism for parameter groups
- Pre-defined modules (e.g. DTI)

What a method does (example)

- Introduce "Echo Time"
- Include Echo Time in protocol
- React to protocol loading
  - Echo Time is legal ?
- React when Echo Time is changed
  - Restrict it,
  - Adapt other parameters (TR, etc)
  - Derive delays d1, d2
- React when other parameters change
  - New TR agrees with TE?

"Relations"

Macros, , PVM?

- Pipeline filters are special. But macros, automations, methods - they all read and set parameters. What is actually the difference?
- When to write what?

BRUKER BIOSPIN

BRUKER

- Batch execution of PV commands
  - "I want to run fifty scans overnight"
- Graphic interfaces (Tcl-Tk)
  - "I need a button for each experiment step"
- Starting external software from PV
  - "I want to run my own reco on a selected scan"
- Automatic post-processing
  - "When the scan is ready, I want to start ISA"
- Small modifications at base-level
  - "I am fine with FLASH, but need trim 7 to be 50%"

↳ test an idea before writing a method

BRUKER BIOSPIN

When to write automations

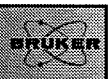
- Small modifications at base-level involving maths or arrays
  - "I want to use FLASH with this powerlist:"

    $\text{Power}[n] = 20 \log (\cos^{-1}(\exp(-(d1/d2))/n),\ n=1...1024$
- Processing with access to several parameters
  - "Reading parameters with *pvcmd* is to slow for me"

When to write methods

- Any experiment with routine interface
  - "I have my macro for modifying FLASH, but each time I change geometry, my changes are lost"
  - "I want to implement my sequence so that EVERYBODY can use it"
- Projects based on PVM modules
  - "Bruker EPI readout is not bad, but I want to use it with my own excitation sequence"
- Adjustments
  - "I want the frequency to be adjusted for the central voxel for each scan"

What we will learn



- Monday, Tuesday:
  - Base-level parameters and pulse programming
  - First meeting with *gre*
- Wednesday
  - PVM programming, part 1 (*gre* mutations)
    - Parameter definitions
    - Relations
    - Timing
    - RF pulses
- Thursday
  - PVM programming, part 2
    - Using modules
    - Creating modules
    - Accelerated MRI
- Friday
  - Pipeline filters

Pulse Program Overview

ParaVision Programming Course
April 3-7, 2006
Paul Freitag

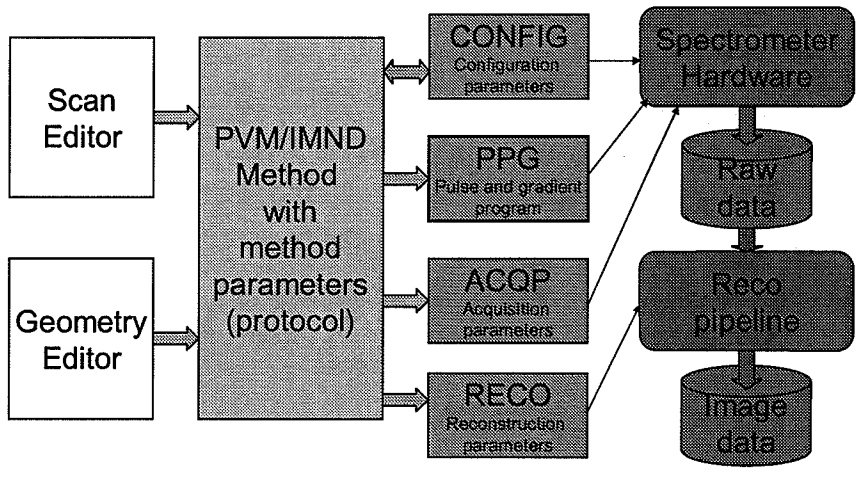
Overview

- **Pulse Programs and Parameters**
  - **General concepts**
  - **First Example**
  - **Control Parameters**
- Introduction To Pulse Programming (1)
  - Development Tools
  - Project 1: Hello World
  - Timing Details / Transmitter Control
  - Dynamic Changes
  - Project 2: Spin Echo
- Introduction To Pulse Programming (2)
  - Gradient Control – Transformations
  - Project 3: Gradients and Parameters
  - Sampling Control
  - Project 4: Phase Cycling and Digitizer Control
  - Gradient Pulses
  - Project 5: Gradients and Pulse Frequency
  - Gradient Control – Ramp
  - Project 6: Gradient Cycling

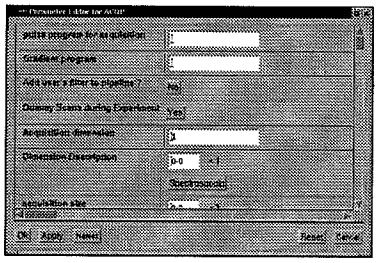
Objectives

- understand the meaning of pulse and gradient programs
- know tools and conventions for pulse programming
- know to write simple pulse programs from scratch
- be able to understand pulse programs delivered with Bruker methods
- be able to solve problems occurring during pulse program development

PPG Concept

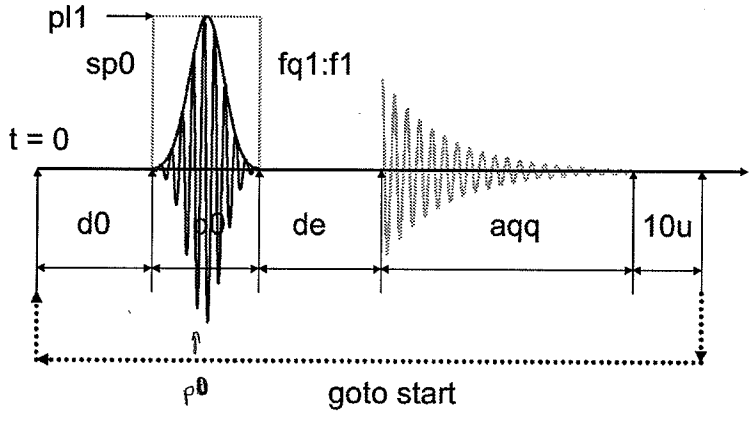


ACQP: Baselevel parameters

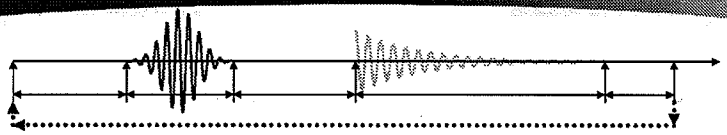


- Sizes/repetitions
- Timing
- Pulses
- Bandwidth
- Digitizer Control
- Gradient transformation
- Gradient trim values
- Gradient ramps
- ....

Pulse Program Overview



Pulse Program Example



```
#include "Avance.incl"
#include "DBX.include"
start, d0 fq1:f1
       (p0:sp0 ph0):f1
       depa REC_ENABLE
       ADC_INIT(ph0,ph1)
       deosc ADC_START
       10u ADC_END
       SETUP_GOTO(start)
       exit
ph1=0
ph0=0
```

Standard Header Files

Pulse Generation

Data Collection

Flow of Control

Phase Definition

different in TB3.0.

BRUKER BIOSPIN

General Info

- ASCII Text – use arbitrary editor
- Default definition in include files: need #include "Avance.incl" and #include "DBX.include"
- Commands are linked to durations (delays) – implicitely or explicitely
- Durations can be specified as parametrized delays or constants
- Constant delays are floating point numbers with time unit s(econd), m(illi), u(icro)
- Semicolons introduce comments

BRUKER BIOSPIN

RF-Pulses

- RF Pulses are generated by a pulse delay
- Parametrized (p0-p63) or constant pulse delays (e.g. 10up) can be used
- A pulse shape can be specified :sp0 - :sp31
- Frequency (SFO1), amplitude (PL[1]) and phase will be set independently for the synthesizer channel.
- Synthesizer channels (:f1) are linked to nuclei parameters

Gradients

- Gradients described in terms of subject oriented coordinate system
- Transformation matrix to change the image geometry
- Individual control of gradient ramps
- User specific gradient shapes
- Phase encoding by variable gradient functions
- Independent

Data Collection

- Standard sequence for data collection

```
depa REC_ENABLE
ADC_INIT(NOPH,ph1)
deosc ADC_START
10u ADC_END
```

- Acquisition delay must be slightly longer than dwell * number of points
- Use synthesizer phase setting ph1
- Phase list must be defined at end of pulse sequence

Parameter Environment

- Run in endless loop for setup mode (gsp):
  start, d0
  SETUP_GOTO(start) (goto start in Pv 3!)
  exit
- In Scan Mode (gop), experiment end is linked to acquired data (ParaVision <= 3) or loop structure (ParaVision 4).
- Parameters control number of acquired scans
- NA (number of averaged scans)
- NI (number of data objects)

Introduction to Pulse Programming (1)

ParaVision Programming Course
April 3-7, 2006
Paul Freitag

Overview

- Pulse Programs and Parameters
  - General concepts
  - First Example
  - Control Parameters
- **Introduction To Pulse Programming (1)**
  - **Development Tools**
  - **Project 1: Hello World**
  - **Timing Details / Transmitter Control**
  - **Dynamic Changes**
  - **Project 2: Spin Echo**
- Introduction To Pulse Programming (2)
  - Gradient Control – Transformations
  - Project 3: Gradients and Parameters
  - Sampling Control
  - Project 4: Phase Cycling and Digitizer Control
  - Gradient Pulses
  - Project 5: Gradients and Pulse Frequency
  - Gradient Control – Ramp
  - Project 6: Gradient Cycling

Tools for Pulse Programming

- Editor – adapt UXNMR_editor environment (Pv3: setres tool in XWINNMR)
- Edit > Pulsprogram
- Tools > Pulsprogram Tool only in PV4.
  (Pv3 Display)

  Interactive editor with syntax check and graphical pulse sequence display

Starting from Scratch

try this at home. PV4 specific?

- New Scan / Cancel during new protocol
- data set status new
- no method defined
- parameters initialised for onepulse
- gradient parameters have no value
- multiple slice packages cannot be defined
- The ACQP parameter PULPROG must be set to the name of the pulse program
- PULPROG parameter defines pulse program in exp/stan/nmr/lists/pp

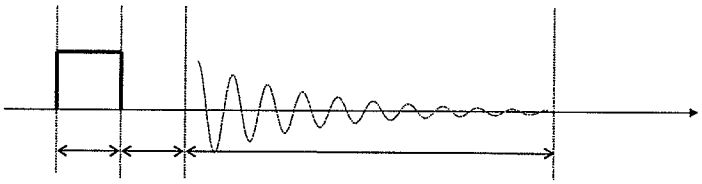
/opt/PPC/PPG/basic PPG

/opt/PV4.0/exp/stan/nmr/lists/pp

Project 1: Hello World

By developing this project, you will learn:

- to write a minimum pulse program from scratch
- to use ParaVision tools for pulse program development
- which parameters need to be set on a new data set
- which parameters implicitly have influence on data acquisition



Processing the Text

- line orientation: \n separate lines
- ; comment lines with semicolon
- significant length of identifiers limited: 7 valid characters
- Standard Headers Avance.incl and DBX.include must be included for imaging applications
- include files : use C preprocessor
- special value PULPROG=*pulseprogram* uses expanded pulse program stored with dataset

↳ guarant

PULPROG example

uses pulse program stored w/ dataset.

PULPROG = simple.ppg

exp/stan/nmr/lists/pp/simple.ppg:

```
#include <deosc.mod>
start, d0
      p0
      aq
goto start
exit
```

exp/stan/nmr/lists/pp/deosc.mod:

```
define delay deosc = { $DEOSC }
"deosc = abs(deosc) * 1e-6"
```

PULPROG = pulseprogram

data/<usr>/nmr/<id>/1/pulseprogram:

```
# 1 "/u/exp/stan/nmr/lists/pp/simple.ppg"
# 1 "/u/exp/stan/nmr/lists/pp/simple.ppg"
# 1 "/u/exp/stan/nmr/lists/pp/deosc.mod" 1
define delay deosc = { $DEOSC }"
deosc = abs(deosc) * 1e-6"
# 1 "/u/exp/stan/nmr/lists/pp/simple.ppg" 2
start, d0
      p0
      aq
goto start
exit
```

Module Concept

- *#include <modulehead>*
- *#include <modulebody>*
- *if (*ParaVisionParameterCondition)
  {}
  else
  {}
- *#define ADC_START* : C-Preprocessor macro definition possible

def'n is in dbx/include directory

Loop structure: Scanner control

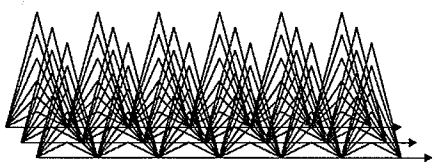
- Number of acquired scans determined by loop parameters:
- RCU based spectrometers:
  acquisition is aborted, as soon as the expected number of scans has been acquired
- Note display options for accumulation:
  - each scan
  - each accumulation
  - each phase encoding step

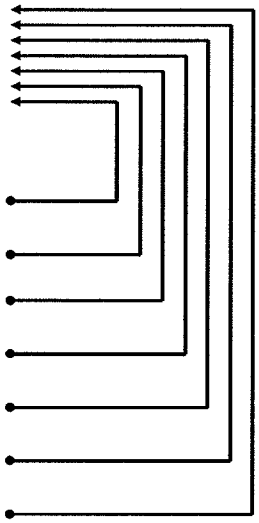
Loop parameters

ie. ways to accumulate the data

- NS:number of simply accumulated scans (same the phase encoding)
- ACQ_ns_list_size / ACQ_ns_list: description of echo accumulation scheme
- ACQ_phase_factor: connected scans
- NI: number of image objects number of slices or numbe. of echoes in MSME
- ACQ_dim / ACQ_size : description of image object
- NA: phase encoding accumulation
- NAE: image accumulation
- NR: number of repetitions

Loop structure Overview





3 NS accumulation

2 ACQ_phase_factor

3 NI Image objects / NSLICES

2 NA accumulation

3 ACQ_size[1] / ACQ_phase_factor

ACQ_size[2]

NR repetition

Timing concepts

- timing units
  s(seconds), m(milli), u(micro)
- resolution 80MHz, 0.125u
- explicit timing <-> implicit timing
- actions start at beginning of delay
- parallel timing possible (use parentheses)
- multiple control commands possible within same delay

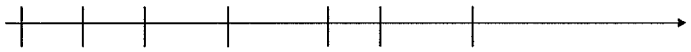
Timing parameters

- D[0-31] : general purpose delays/unit [s]
- P[0-31] : general purpose pulses/unit [u]
- VDLIST: variable delay list/unit [s]
- VPLIST: variable pulse list/unit[u]
- DE: minimum delay/unit [u] ⇒ ringing delay
- DEOSC: minimum acquisition delay[u]
- DEPA: minimum preamp delay[u]

Timing commands

• <float>*u,m,s*	fixed delay
• <float>*up,mp,sp*	fixed pulse delay
• *d0-d31*	parametrized delay
• *p0-p31*	param. pulse delay
• *vd*	variable delay
• *vp*	variable pulse delay
• *de*	min. ringing delay
• *dw*	dwell time (real)
• *depa*	min. delay
• <delay>*<scaling factor>	scaled delay

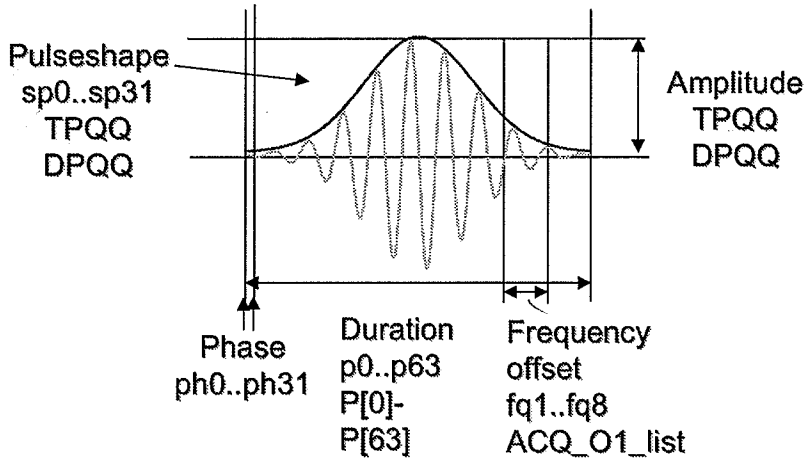
Timing example



10u 10u d0 p1*2 vd deosc

(10u) (20u) ; parallel pulse trains in parentheses
d0
p1*2
vd
deosc

Synthesizer Control Overview



Channel concept

- Eight transmit channels can be addressed
- In pulse program, channel id :f1 - :f8
- Basic frequency defined by nucleus
- Amplifier mapping part of routing
- default channel: f1

Transmitter Control Parameters

- TPQQ[0-15]/DPQQ[0-15]
  - .name: shape file in exp/stan/nmr/lists/wave
  - .power: attenuation in dB from -6dB to 120dB
  - .offset: phase shift generated offset in Hz
- SFO1-SFO8: Basic frequency in Hz
- ACQ_O1_list_size/ACQ_O1_list-ACQ_O3_list – ACQ_O1B_list
  - offset frequency lists – B for second oscillator
- PL[0-31]: default power levels for channels PL[1] for :f1 ...

Transmitter Control Commands

- *fq1:f1-fq8:f1*
  set frequency from ACQ_O1_list ...,
  implicit increment
- *fq1b receive* : Use second DDS for receive
- *fq8b:f1* set frequency from ACQ_O1B_list
- *pl0:f1-pl31:f1*:
  set power level from PL[0]-PL[31]
- *:sp0-:sp31* [*(currentpower)*]
  use shape from TP(DP)QQ[0]-TP(DP)QQ[15],
  use current power as option

Phase List Definition

- *ph0:f1-ph31:f1*
  set phase from ph0-ph31
- ph0-ph31=(<unit>) list definition
- Shortcuts for repeated / shifted groups
  - ph0= { < list function> } * <factor>
  - ph1= { < list function> } ^ {increment}
- phase values as multiples of the fractional unit, default 4
- receiver (mixing) phase only fractional unit 4 available

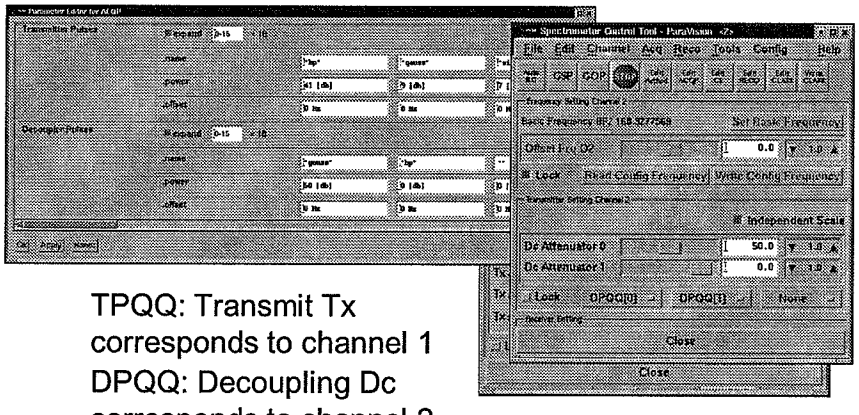
Transmitter Control Details

refers to the freq. channel

- gatepulse 1: amplifier presetting
- preset off: optimize performance (if not included, may have timing probs.)
- shape timing: 0.1u grid
- shape files exp/stan/nmr/lists/wv
- dynamic changes limited
- setup control: linked to sliders
- TPQQ[0-7] : sp0-sp7
  DPQQ[0-7] : sp8-sp15
  TPQQ[8-15] : sp16-sp23
  DPQQ[8-15] : sp24-sp31

Pulse Power Setup

TPQQ and DPQQ connected with sliders in Spectrometer Control Tool:



TPQQ: Transmit Tx corresponds to channel 1
DPQQ: Decoupling Dc corresponds to channel 2

Transmitter Control Example

```
...
10u pl1:f1
10u fq1:f1
d8 gatepulse 1
(p0:sp0(currentpower) ph1):f1
...
ph1 = {{0 1}*2}^1 ; 0 1 0 1 1 2 1 2
ph2 = (360) 0 90 180 270
```

Changing the Flow of Control

- <label>, : mark position in pulse program
- *goto* <label>: jump to label
- *lo to* <label> *times* <counter> : repeat block times counter
- counter may be ParaVision parameter or l0-l31 corresponding to L[0]-L[31]
- *if* "<condition>" *goto* <label>
- *exit* : end execution

Increment, Decrement and Reset

- *ipp0-ipp31, rpp0-rpp31:* increment/reset phase list pointer
- *ip0-ip31, dp0-dp31, rp0-rp31:* increment/decrement/reset phase list
- *ipu0-ipu31, dpu0-dpu31, rpu0-rpu31:* increment,decrement,reset pulse delay by value defined in INP[0]-INP[31]
- id0-id31, dd0-dd31,rd0-rd31: increment, decrement, reset delay by value defined in IND[0]-IND[31]
- *ivd/ivp:* increment pulse/delay list pointer

Calculating in Pulse Programs

- "expression": calculation in double quotes
- C-style syntax for assignment and simple algebra and functions
- time units are preserved
- ParaVision parameters cannot be evaluated directly
- examples:
  "d0 = d1 * 2 - p0"
  "p0 < 100u"

User defined Commands

- New commands can be defined in the beginning of the pulse program (before the first delay) ✓ 7 characters only!
  - *define delay* delayname
  - *define pulse* pulsename
  - *define loopcounter* loopcountername
  - *define list<delay>* delaylistname
  - *define list<pulse>* pulselistname
  - *define list<power>* powerlistname
  - *define list<frequency>* fqlistname

Initialising User Commands

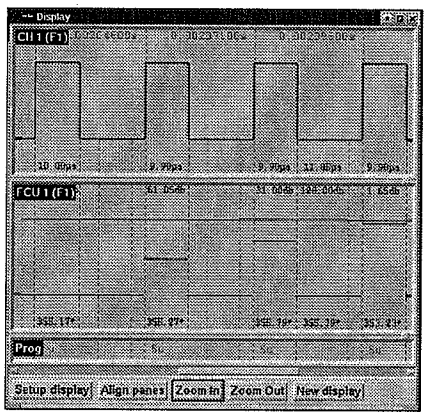
- User command lists must be initialized
  - either directly:
    define list ... = { values }
  - from file:
    define list ... = <file>
  - from ParaVision parameter
    define .... = {$ParaVisionParameterName}

Using and Navigating in Lists

- User defined list commands can be used like the built in commands:
  - *define list*<listtype> *somelist*: definition
  - *somelist*: use list at current position
  - *somelist.inc*: increment list pointer
  - *somelist.dec*: decrement list pointer
  - *somelist.res:* reset list pointer
  - *somelist[ndx]*: access list value at ndx
  - *somelist.idx*: access list pointer in elation
  - *somelist^*: use list with autoincrement

Pulse Power List Example

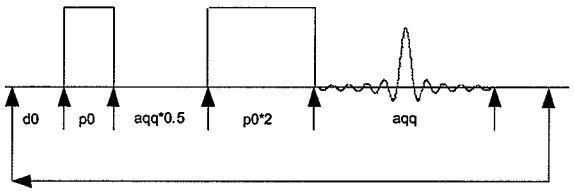
```
define list<power> plist = { 120 60 30 0 }

start, 10u plist:f1 plist.inc
       10up
       5u
       lo to start times 4
exit
```



Project 2: Spin Echo

By developing this project you will learn

- to use parameters for timing
- to implement easy relations between the parameters in the pulse program
- to vary properties of rf-pulses during runtime



Introduction to Pulse Programming (2)

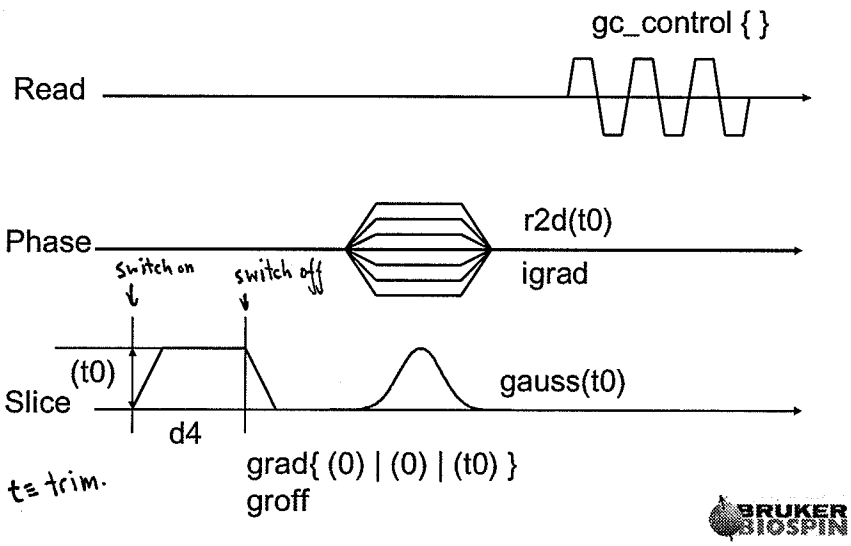
ParaVision Programming Course
April 3-7, 2006
Paul Freitag

Overview

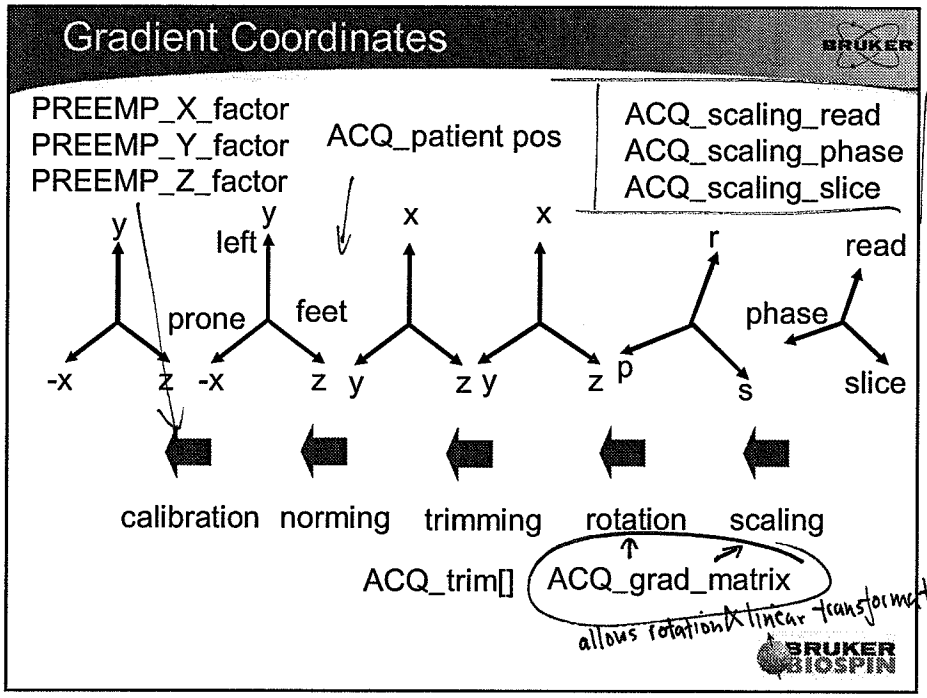
- Pulse Programs and Parameters
  - General concepts
  - First Example
  - Control Parameters
- Introduction To Pulse Programming (1)
  - Development Tools
  - Project 1: Hello World
  - Timing Details / Transmitter Control
  - Dynamic Changes
  - Project 2: Spin Echo
- **Introduction To Pulse Programming (2)**
  - **Gradient Control – Transformations**
  - **Project 3: Gradients and Parameters**
  - **Sampling Control**
  - **Project 4: Phase Cycling and Digitizer Control**
  - **Gradient Pulses**
  - **Project 5: Gradients and Pulse Frequency**
  - **Gradient Control – Ramp**
  - **Project 6: Gradient Cycling**

12000 / 41496 * Hz/cm

Gradient Control Overview



Gradient Coordinates



Gradient Control Parameters

- ACQ_patient_pos:
  normalized subject coordinate system
- NSLICES: number of slices
- ACQ_grad_matrix[][][]
  transformation matrix per slice
- ACQ_trim: trim values
- ACQ_read_scale, ..._phase_..., slice :
  Scaling values in logical coordinates
- ACQ_phase_encoding_mode: ramp mode
- ACQ_phase_enc_start: ramp shift

Gradient Coordinate Transformation

grad{(80,80,100) | (50,50,50) | (0,0,100) }

ACQ_scaling_read/..
{ 1,1,0.5}

ACQ_grad_matrix

$$\begin{vmatrix} 0.7 & 0.7 & 0 \\ -0.7 & 0.7 & 0 \\ 0 & 0 & -1 \end{vmatrix}$$

ACQ_patient_pos
head_left

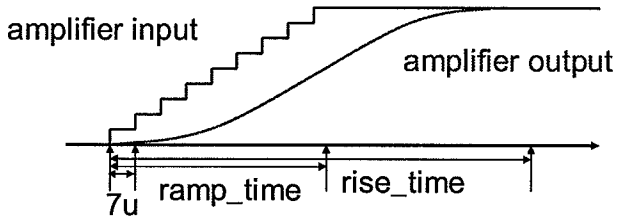
PREEMP_X/Y/Z_factor
{ 0.98, 0.98, 0.94 }

	read x y z	phase x y z	slice x y z
	(0.7, 0.7, 0)	(-0.7, 0.7, 0)	(0, 0, -0.5)
*		*	*
	(80,80,100)	(50,50,50)	(0, 0, 100)
=		=	=
	(56,56, 0) +	(-35, 35, 0)+	(0,0,-50)

=
(21,91,-50)
-> z*-1, xy->yx
(91,21,50)
*
(0.98, 0.98, 0.94)
=
(89.18, 20.58, 47)

Gradient Calibration

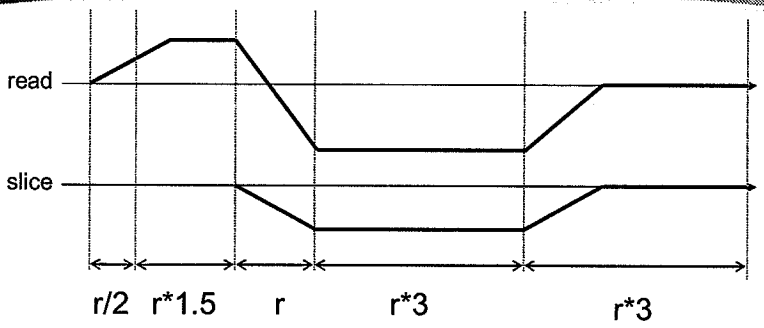
- PREEMP_ramp_time : technical ramp time (amplifier input ramp)
- PREEMP_ramp_mode : ramp description
- PREEMP_rise_time : physical ramp time
- PREEMP_X_factor/Y_factor/Z_factor: calibration constants



Gradient Commands

- *grad*{ <read> | <phase> | <slice> }
  - <read>/<phase>/<slice> gradient strength as percentage:
  - (0) : single trim value, range -100 to +100
  - (100,100,100) : x/y/z trim value
  - (t0)-(t99) : parameter trim value
  - <>+-*trim : expression of trim values
- *groff*: shortcut for *grad{(0)|(0)|(0)}*
- delay marks **start** of gradient ramp
- effective ramp delay PREEMP_ramp_time

Gradient command examples

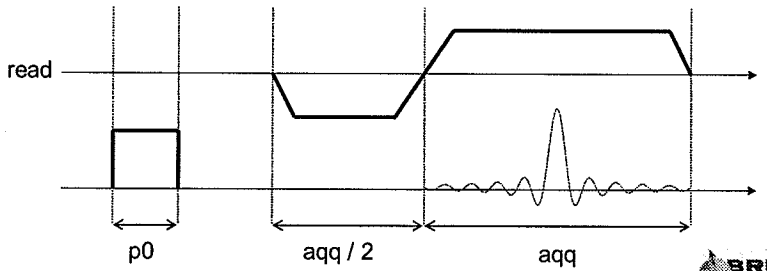



```
ramp*0.5  grad { (20) | (0) | (0) }
ramp*1.5          ; ACQ_trim[0] = {-30, -30, -30 }
ramp        grad { (t0) | (0) | (-20,-20,-20) }
ramp*3
ramp*3   groff
```

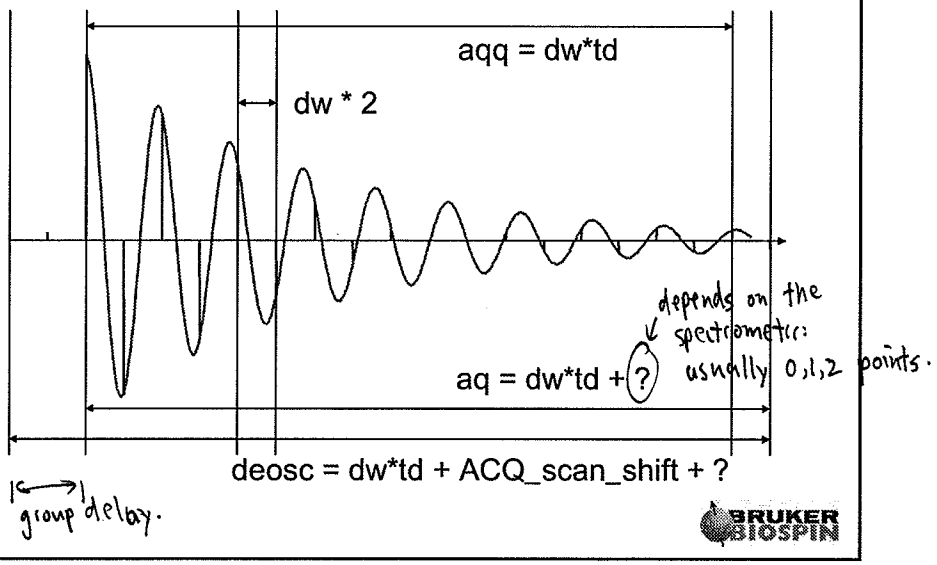
Project 3: First Gradients

By developing this project you will learn

- which parameters must be set to use gradients within a pulse program
- how geometric transformations influence gradients
- how static gradients can be switched in a pulse program



Data Sampling Overview

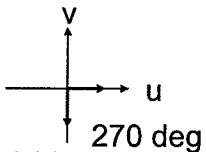


Acquisition Commands

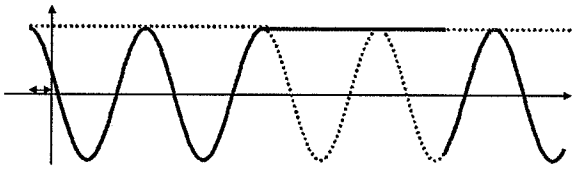
- *REC_ENABLE*:
  - receiver enable for RX22 - at least depa
- *ADC_INIT(_B)(*<mixing>,<reference>*)*
  - prepare acquisition (use fq8b frequency list)
  - set mixing phase and reference phase
  - implicit delay DE - DEPA
- *ADC_START*:
  - mark acquisition of first point
  - must wait DEOSC until ADC_END
- *ADC_END*:
  - mark end of acquisition/minimum delay 10u

Receiver Reference vs Mixing Phase

- ADC_INIT_B(mixing phase , reference phase)
- reference phase -> synthesizer phase
- mixing phase = channel mixing
  (u,v) -> (±u,±v)



- attention: mixing phase prevents fast loops
- Dual DDS: TX -> RX phase continuous
  RX -> TX phase coherent



Scan Parameters

- ACQ_size[0] : size of scan (real valued) alias TD
- ACQ_scan_size: combination flag
- ACQ_phase_factor: scan division
- ACQ_scan_shift: group delay compensation
- SW_h: bandwidth dw = SW_h / TD
- DIGTYP: digitizer selection
- DIGMOD: filter mode analog/digital
- AQ_mod: data sampling mode qdig/qsim

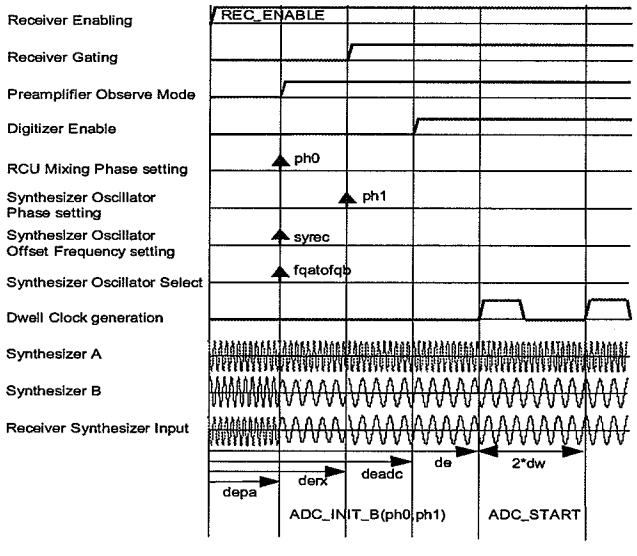
no group delay related to digital filter

Acquisition Timing

- Parameter Setting:
  - ACQ_size[0] = 1024
  - SW_h = 50000 => dw = 20 aqq = 10240u
  - DSPFIRM = DSP_medium
- Derive minimum delay for optimum shift
  - ACQ_scan_shift = -1 -> -26
  - DEOSC = -1 -> -10560
- Derive optimum shift for available delay
  - DEOSC = +10400

  → ACQ_scan_shift = -10

Acquisition Start Details



Acquisition Start Example

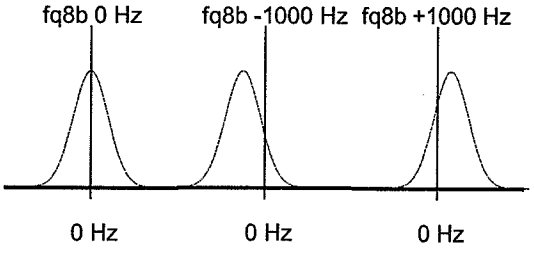
synthesizer b.

```
fq1b:receive ; use ACQ_O1B_list for channel 1
        ...
        10u fq8b:f1 ; set and increment ACQ_O1B_list
        ...
depa    REC_ENABLE
        ADC_INIT_B(NOPH,ph0)
deosc   ADC_START
10u     ADC_END
;       wait at least 800u until next ADC_INIT
```

Project 4: Phase Cycling and Digitizer Control

By developing this project you will learn

- to implement minimum timing for data acquisition
- to select a frequency offset for acquisition
- to run phase cycling



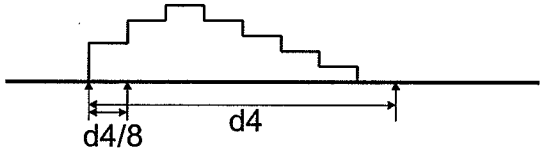
Slice Increment

- One ACQ_grad_matrix for each slice
- Dimension cannot be changed within acqp
- Number of orientations in NSLICES
- change orientation
  - *islice* : next orientation matrix
  - *rslice* : previous orientation matrix
  - *zslice* : first orientation matrix
  - *sslice* : save slice index
  - *rslice* : restore slice index

Gradient Shapes

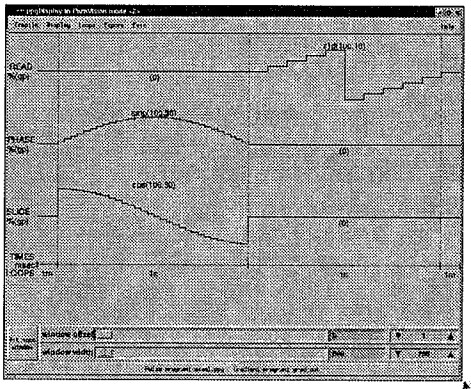
- Ramp functions with length definition
- *grad*{ ... <function>*(*<trim>,<len>*)* | ... }
- length must match function definition
- user defined functions possible
- delay defines step resolution

```
ufun = { 0.5, 0.8, 1.0, 0.8, 0.6, 0.4, 0.2, 0.0 } ...
d4 grad{(ufun(100,8)) | (0) | (0) }
```



Gradient Shape Example

```
1m
1s grad{ (0) | sinp(100,30) | cos(100,30) }
1s grad{ r1d(100,10) | (0) | (0) }
1m groff
exit
```



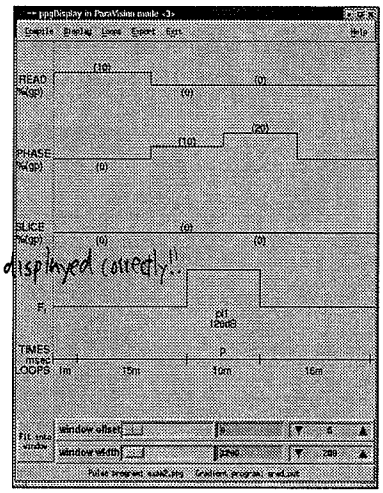
Gradient Sequences

- Gradient sequence in parallel to pulse timing can be defined
- *gc_control* {
  <delay> *grad*{...|...|...}
  <delay> *grad*{...|...|...}
  ...
  }

gc_control Example

```
1m  gc_control {
      10m grad { (10) | (0)  | (0) }
      10m grad { (0)  | (10) | (0) }
      10m grad { (0)  | (20) | (0) }
      10m groff
}
15m  ; pulse timing independent
10mp ; from gradient timing
15m
exit
```

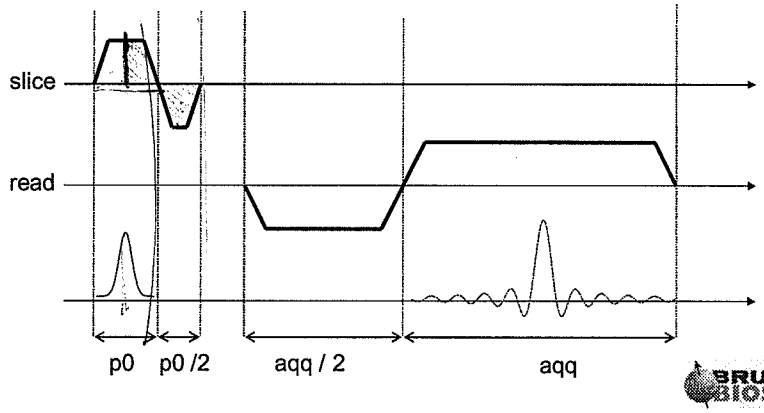
this parallel gradient event timing may not be displayed correctly...



Project 5: Gradients and Pulse Frequency

By developing this project you will learn

- To use a slice selection gradient
- To vary the transmitter frequency for pulses



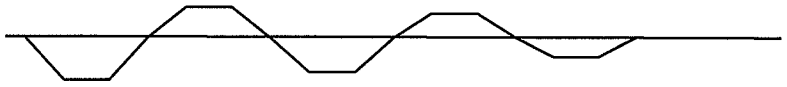
Phase encoding

- ramp list functions *r1d, r2d, r3d*
- linked to parameters ACQ_phase_encoding_mode/enc_start
- further built in functions
  - *sin*
  - *cos*
  - *sinp*
  - *gauss*<truncvalue>
  - *step*
- *lgrad* <function>[<*2d*>|<*3d*>] = length length definition/setup mode

Gradient Ramp Increment

- step through gradient ramp lists
  - *igrad* <function> : increment pointer
  - *dgrad* <function> : decrement pointer
  - *zgrad* <function> : reset pointer
  - *sgrad* <function> : save status
  - *rgrad* <function> : restore status

Gradient Encoding Example: Centered

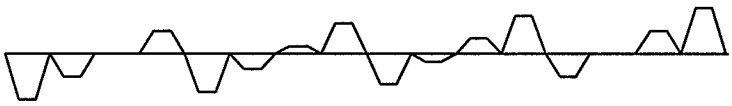


-30 % +20 % -20 % +10 % -10 % 0 %

- ACQ_phase_encoding_mode[1] = Centered
- ACQ_trim[0] = { 30, 30, 30 }
- ACQ_size[1] = 8
- ACQ_phase_enc_start[1] = -1

```
lgrad r2d = ACQ_size[1]
ramp grad{ (0) | r2d(t0) | (0) }
igrad r2d
```

Gradient Encoding Example: RARE

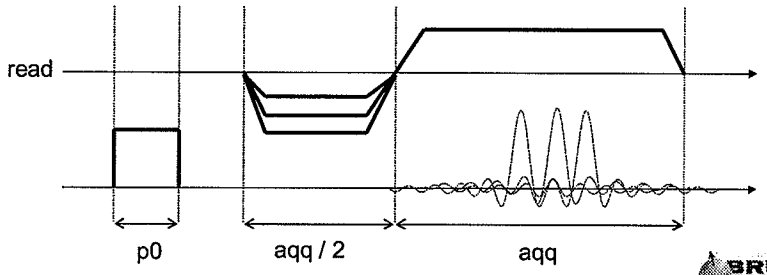


```
slice,   rgrad r2d
cloop, d1 grad((0) | r2d(t0) | (0)
         groff
         igrad r2d
lo to cloop times ACQ_phase_factor
         sgrad r2d
lo to slice time NSLICES
lo to start times ACQ_size[1]/ACQ_phase_factor
```

Project 6: Gradient Cycling

By developing this project you will learn

- to vary gradients during runtime
- to use parameters to control gradient variation
- to switch off gradients during setup mode



Pulse Programming Exercises

ParaVision Programming Course
April 3-7, 2006

Author:
Paul Freitag

Bruker BioSpin MRI GmbH

Table of Contents

1 Starting from scratch

1.1 Objective

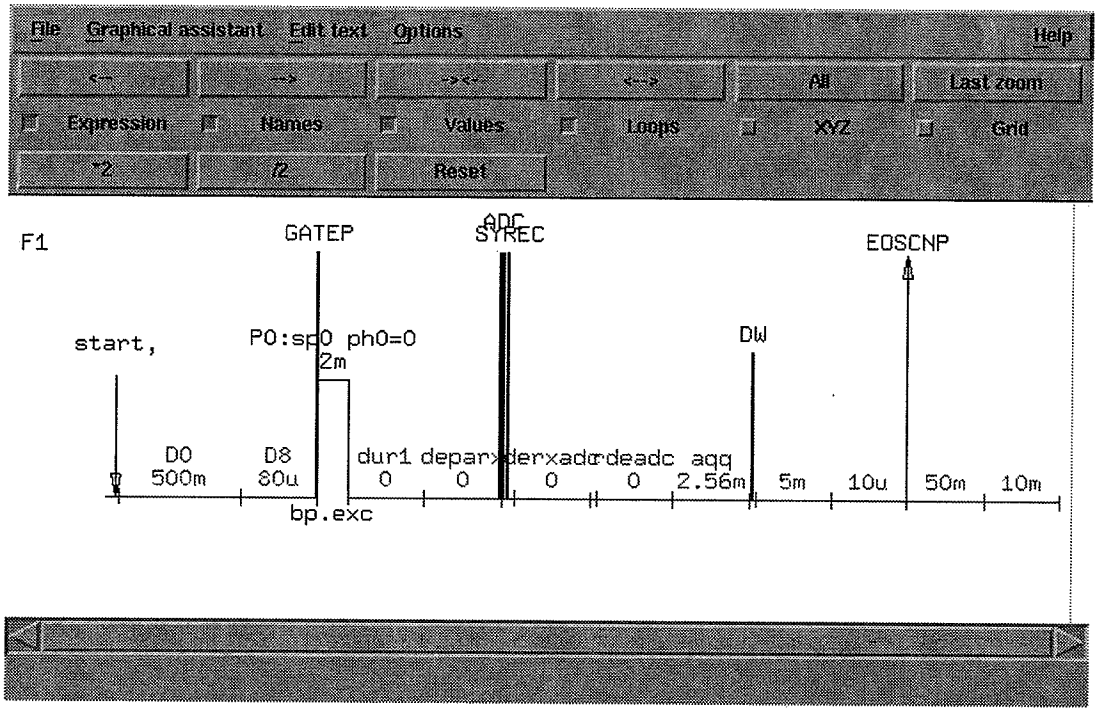
By developing this project, you will learn:

- to rename an easy pulse program for your data set
- to use ParaVision tools for pulse program development
- which parameters need to be set on a new data set

1.2 Project description

In this project, you will create your first pulse program of your own.

For this purpose, you create a new data set without loading a protocol, you create a copy of the pulse program project1.ppg provided in your home directory to the pulse program library directory. You use the parameter editor to select the pulse program for your newly created date set. You use the pulse program display to inspect the pulse program and to correct a syntax error (missing comment sign) in the interactive editor. You set suitable acquisition parameters and run a first pulse-acquire experiment on your spectrometer. Make a list of parameters for which you think that they influence the acquisition. You have finished as soon as you have acquired a reasonable fid file.



1.3 Extensions

If you finish early, you may try to minimize your pulse program, i.e. try to remove as many lines in the pulse program as possible to allow you still to acquire an fid.

1.4 Hints

- Create a new data set (new scan in pvScan), using the cancel button, when asked to load a protocol.
- Pulse programs must be installed in the `<PvInstDir>/exp/stan/nmr/lists/pp` directory.
- Parameters to be set: Pulse Program Name (`PULPROG`), Relaxation delay `D[0]` Pulse delay `P[0]`, Pulse Shape 0 `TPQQ[0].name`, `TPQQ[0].power`. A suitable pulse will be the blockpulse `bp.exc`.
- In order to simplify parameter input, you can open a parameter editor restricted to the parameters relevant to you:
  `pvcmd -a pvScan pvEditPars PULPROG D[0] P[0] TPQQ[0].name TPQQ[0].power`
  TR
- Further parameters have influence on the acquisition: The bandwidth (`SW_h`), the group delay compensation (`ACQ_scan_shift`), the amplifier gating delay `D[8]` (derived from `CONFIG_amplifier_enable`).

making a simple macro:

Macro Manager → record → stop → input name of macro → edit file.

1.5 Solution

```
;
; PPC 2006 Pulse Programming Course
;
; Project 1: first pulse program - simplified SINGLEPULSE
;
; Explicit parameters
;
; P0 - excitation pulse delay
; d0 - relaxation delay
; d8 - CONFIG_amplifier_enable per convention
;

;-------necessary include files---------------------------------
#include<Avance.incl>
#include <DBX.include>

; automatic pulse gating switched off for fast sequences
preset off

;label--delays--commands----gradients-------remarks--------------
start,  d0                                   ; relaxation delay

;-------excitation---------------------------------------------
d8      gatepulse 1                          ; pulse gating
(p0:sp0 ph0):f1                              ; pulse TPQQ[0]

;-------data acquisition----------------------------------------
depa    REC_ENABLE                           ; open receiver
ADC_INIT(ph0, ph1)                           ; prepare acquisition
Aqq     ADC_START                            ; start data sampling
5m                                           ; group delay comp.
10u     ADC_END                              ; end data sampling

;-------data loops----------------------------------------------
SETUP_GOTO(start)

exit

;-------phase definitions---------------------------------------
ph0 = 0                                     ; excitation phase
ph1 = 0                                     ; reference phase
```

2 Pulses, Parameters and Relations

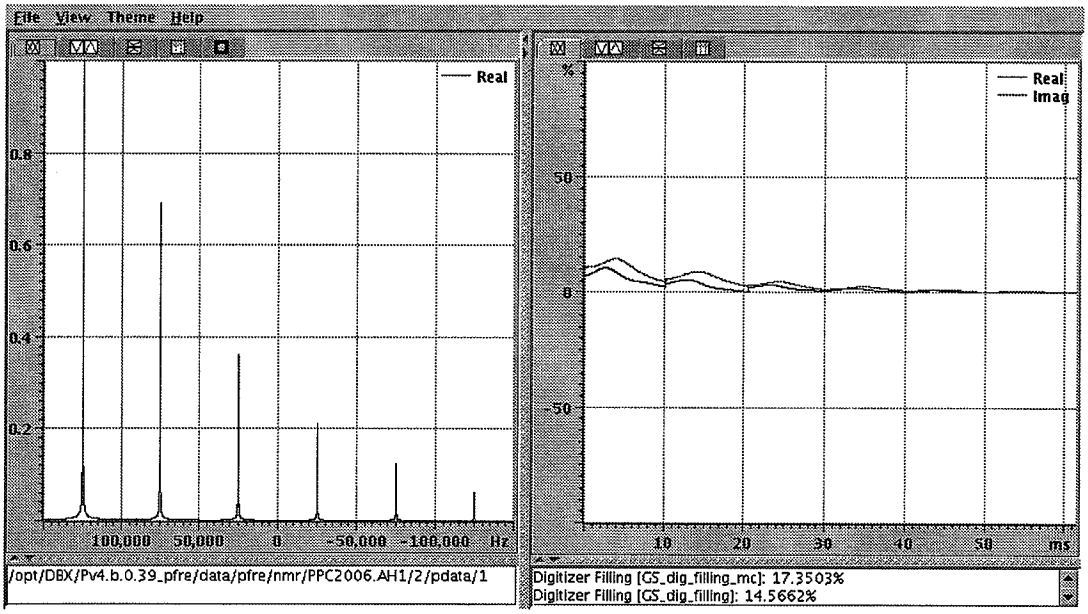
2.1 Objective

By developing this project you will learn

- to use parameters for timing
- to implement easy relations between the parameters in the pulse program
- to vary properties of delays during runtime

2.2 Project

In this project, we want to implement an easy spin-echo 1D method, which can be controlled by base level parameters. By acquiring a series of scans with varying echo time, the method can be used to observe T2 decay.



A spin echo method consists of an excitation pulse, a preparation delay an inversion pulse and the detection period. The spin echo will appear after the same time as the preparation delay. In order to observe the echo, the flip angle of the inversion pulse must be 180deg and the position of the acquisition delay must be adjusted such that the echo will occur in the middle of the acquisition delay. By varying the delay between the excitation and the refocusing pulse, the echo position can be varied.

To achieve this, create a new scan and create a copy of the pulse program from the first project to `project2.ppg`. Add a new pulse into the pulse program controlled by the parameters `P[1]` and `TPQQ[1]` and a phase list `ph2`. Use the delay `D[1]` to describe the echo time. Calculate filling delays in relations in the pulse program in order to position the echo correctly in the acquisition window. Use this first implementation of the pulse sequence to set up suitable parameters for the refocusing pulse and the echo time. Once you observe a reasonable echo, add a loop over the parameter `NI` to the pulse program. Add a variable

delay between the two pulses and the pulse and the acquisition window and vary the delay according to a list in say 6 steps of 50ms each.

2.3 Extensions

Use a different method, to vary the echo time. Possible methods are:

- vd delay and `ACQ_vd_list` parameter with `ivd`
- `define list<delay> vardel = { 0.05 0.1 0.15 0.2 0.25 0.3 }` with `vardel.inc`
- `id1` command in combination with the `IN[1]` parameter
- Incrementing `D[1]` in a "parameter relation"

2.4 Hints

- Clone the first scan
- Copy the pulse program `project1.ppg` to the new pulses program `project2.ppg`
- Remember that the pulse should consist of the two lines
  ```
  d8 gatepulse 1
  (p1:sp1 ph2):f1
  ```
- Add the phase list `ph2=0` to the end of the pulse program – think which phase relative to the excitation pulse is reasonable.
- To calculate the correct relations you may define new delays
  ```
  define delay drest
  ```
- To calculate the correct echo delay, you must take into account, that the `ADC_INIT` command has an implicit delay `de-depa`.
- When the echo delay `D[1]` is introduced between the refocusing pulse and the acquisition delay, a suitable filling delay between excitation and refocusing pulse might calculate
  ```
  "drest =  d1 - de + depa + aqq/2 - p0/2 - d8"
  ```
- You can start an adapted parameter editor
  ```
  pvcmd -a pvScan pvEditPars PULPROG NI D[0] D[1] P[0]
  TPQQ[0].name TOQQ[0].power P[1] TPQQ[1].name TOQQ[1].power
  ```
- Add the loop
  ```
  lo to start times NI
  ```
  just before the `SETUP_GOTO(start)` macro
- When adding a delay list, don't forget to increment the list pointer

2.5 Solution

```
;
; PPC 2006 Pulse Programming Course
;
; Project 2: simple spin echo
;
; Explicit parameters
;
; p0 - excitation pulse delay
; p1 - refocusing pulse delay
; d0 - relaxation delay
; d1 - echo time
; d8 - CONFIG_amplifier_enable per convention
; NI - image object loop
;

;-------necessary include files---------------------------------
#include<Avance.incl>
#include <DBX.include>

;-------internal parameter definitions and relations-------------
define delay dr
"dr = aqq / 2 + dur1 - p0/2 - d8"
define list<delay> vdl = { 0.05 0.10 0.15 0.20 0.25 0.30 }


; automatic pulse gating switched off for fast sequences
preset off

;label--delays--commands----gradients-------remarks--------------
start,      d0                                ; relaxation delay

;-------excitation----------------------------------------------
d8      gatepulse 1                           ; pulse gating
(p0:sp0 ph0):f1                               ; pulse TPQQ[0]

;-------echo time padding---------------------------------------
dr                                            ; echo time padding
vdl                                           ; var. echo spacing

;-------inversion-----------------------------------------------
d8      gatepulse 1                           ; pulse gating
(p1:sp1 ph2):f1                               ; pulse TPQQ[1]

vdl                                           ; var. echo spacing
;-------data acquisition----------------------------------------
depa    REC_ENABLE                            ; open receiver
ADC_INIT(ph0, ph1)                            ; prepare acquisition
aqq   ADC_START                               ; start data sampling
5m      vdl.inc                               ; group delay comp.
10u     ADC_END                               ; end data sampling

;-------data loops----------------------------------------------
lo to start times NI                          ; NI "image" objects

SETUP_GOTO(start)
```

```
exit

;-------phase definitions-----------------------------------------
ph0 = 0                                          ; excitation phase
ph1 = 0                                          ; reference phase
ph2 = 0
```

3 First Gradients

PREEMP_grad.cal - const strength in Hz/cm.

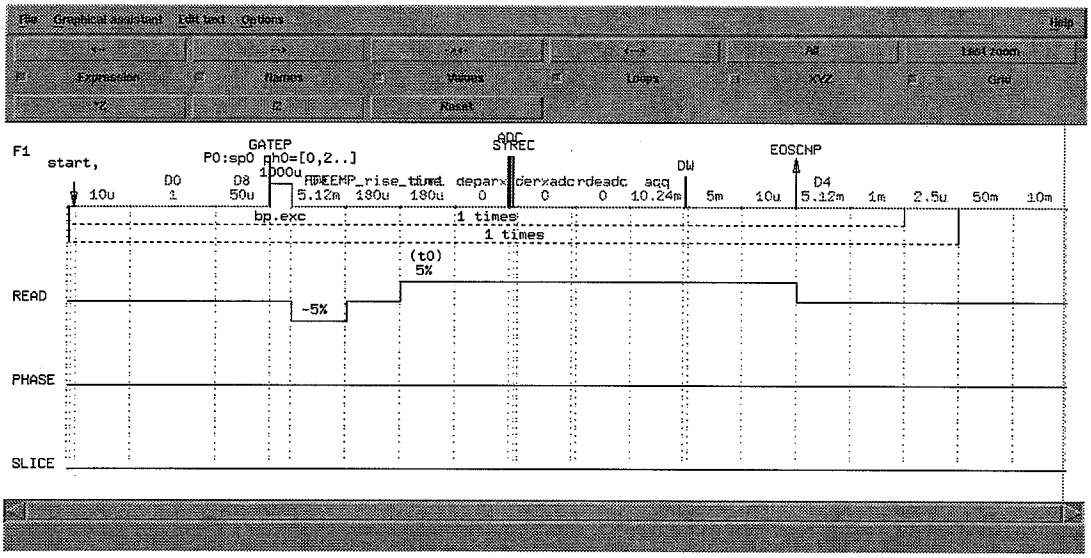
3.1 Objective

By developing this project you will learn
- which parameters must be set to use gradients within a pulse program
- how geometric transformations influence gradients
- how static gradients can be switched in a pulse program

3.2 Project

In this project, we want to implement a basic gradient echo and learn which base level parameters must be used to control the gradient system.

In order to observe a gradient echo, the spin system must be dephased after excitation and rephrased during the acquisition period by applying a gradient of the opposite polarity. The echo will appear when the integral of the rephrasing gradient reaches the same integral as the dephasing gradient.



In order to implement this, create a new scan and create a copy of the pulse program from project 1 to `project3.ppg`, add a dephasing gradient after the end of the excitation pulse which has half of the length of the acquisition interval. Then switch immediately to the opposite polarity and start data sampling. Switch the gradient off, when data sampling is finished. Use the trim value `t0` to parameterize the gradient strength. Have in mind that the minimum delay for switching gradient states depends on the gradient system and preemphasis setting (`PREEMP_ramp_time/PREEMP_rise_time`). Before you can start data acquisition, you must make sure that parameters describing the gradient transformation (`ACQ_grad_matrix`) have been set to a suitable value.

3.3 Extensions

Use different logigal gradient directions to create your echo

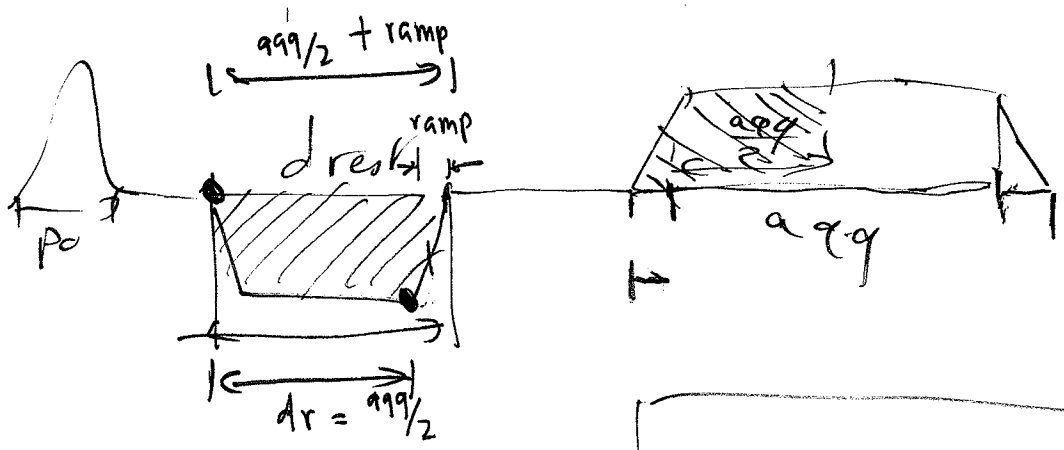
3.4 Hints

- Clone the first scan
- Copy the pulse program `project1.ppg` to the new pulse program `project3.ppg`
- A gradient with trim value e.g. in read direction can be applied with the command
  `grad { (t0) | (0) | (0) }`
- Negative trim values are not allowed but expressions where a trim value is subtracted from the value `(0)`
  `grad { (0)-(t0) | (0) | (0) }`
- You can start an adapted parameter editor
  `pvcmd -a pvScan pvEditPars PULPROG D[0] D[1] P[0] TPQQ[0].name TOQQ[0].power ACQ_trim[0] ACQ_grad_matrix`
- Import the ramp length from a ParaVision parameter: many methods set `D[4]` to the ramp time – remember that ramp parameters are specified in microseconds.
- `Define delay ramp = { PREEMP_rise_time }`
- `"ramp = ramp * 1e-6"`

3.5 Solution

```
;
; PPC 2006 Pulse Programming Course
;
; Project 3: simple gradient echo
;
; Explicit parameters
;
; p0 - excitation pulse delay
; d0 - relaxation delay
; d8 - CONFIG_amplifier_enable per convention
; t0 - trim value ACQ_trim[0]
; NI - image object loop
;

;-------necessary include files---------------------------------
#include<Avance.incl>
#include <DBX.include>

;-------internal parameter definitions and relations------------
define delay ramp = { $PREEMP_rise_time }
"ramp = ramp * 1e-6"
define delay dr
"dr = aqq / 2 "


; automatic pulse gating switched off for fast sequences
preset off

;label--delays--commands----gradients-------remarks--------------
start,  d0                                    ; relaxation delay

;-------excitation----------------------------------------------
        d8  gatepulse 1                      ; pulse gating
        (p0:sp0 ph0):f1                      ; pulse TPQQ[0]

;-------dephasing gradient--------------------------------------
        dr          grad { (0)-(t0) | (0) | (0) }
        ramp        groff

;-------refocussing gradient------------------------------------
        ramp        grad { (t0) | (0) | (0) }; Refocussing
;-------data acquisition----------------------------------------
        depa    REC_ENABLE                   ; open receiver
        ADC_INIT(ph0, ph1)                   ; prepare acquisition
        aqq ADC_START                        ; start data sampling
        5m      groff                        ; gradients off
        10u     ADC_END                      ; end data sampling

;-------data loops----------------------------------------------
lo to start times NI                         ; NI "image" objects

SETUP_GOTO(start)

exit
```

```
;-------phase definitions-----------------------------------------
ph0 = 0                                          ; excitation phase
ph1 = 0                                          ; reference phase
```



ggg/2 - ramp

d res + ramp = ramp + aqq/τ

4 Fast Data Acquisition and Phase Cycling

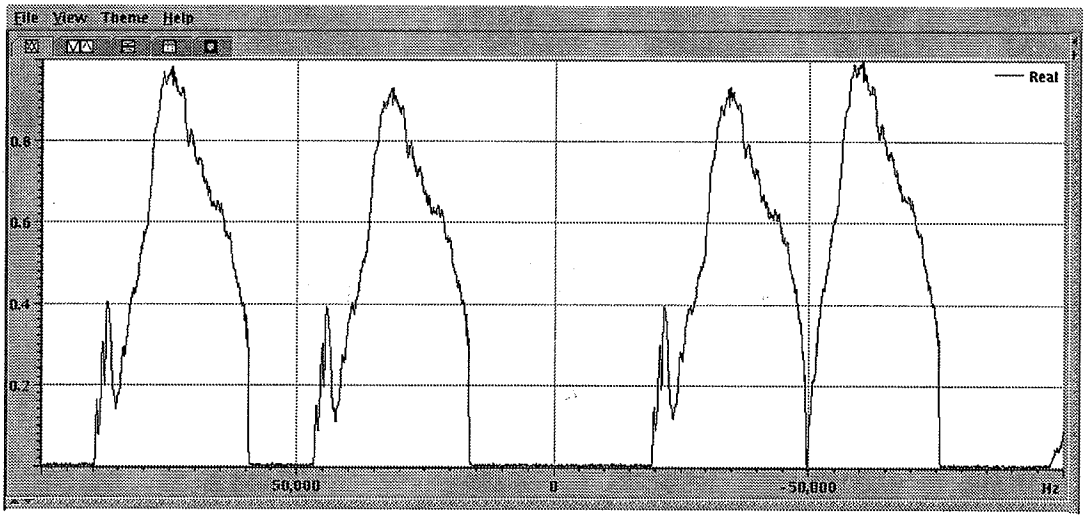
4.1 Objectives

By developing this project you will learn
- to implement minimum timing for data acquisition
- to select a frequency offset for acquisition
- to run phase cycling

4.2 Project

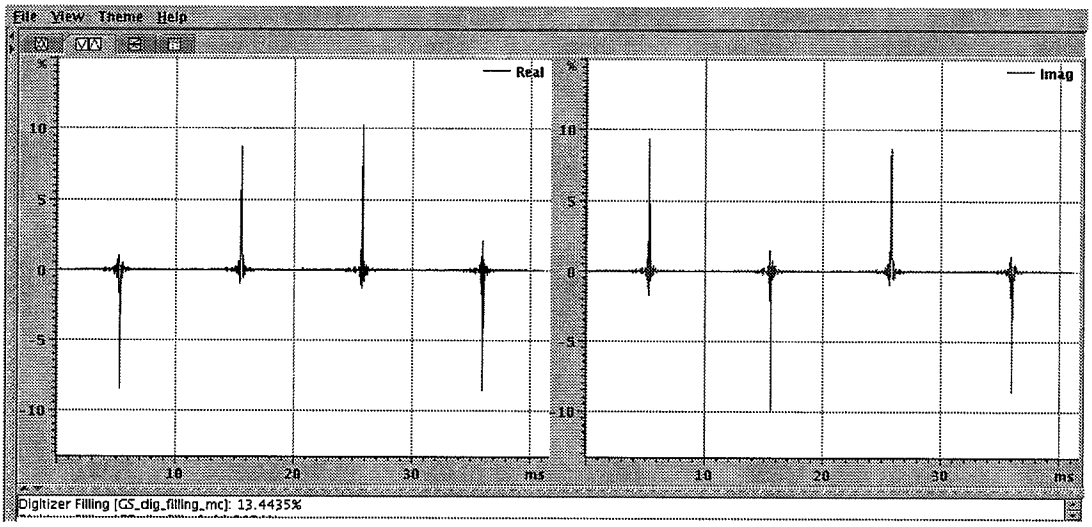
In this project, you will get acquainted with the details of the data collection and phase cycling process.

To apply a read offset in imaging, the LO frequency of the receiver can be tuned during data acquisition. This will lead to a shift of the acquired data in the frequency domain and can be interpreted as a spatial shift in the image domain.



To implement this project clone the scan from project 3, create a copy of the pulse program to `project4.ppg`. To achieve minimum timing in a pulse sequence, replace the fixed delay following the acquisition delay by a delay of length `DEOSC - aqq`. Relations ensure that `DEOSC` will be set to a minimum value when set to a value less than 0.
To implement the read offset, a second synthesizer channel must be used. This is implied by chosing the macro `ADC_INITB` instead of `ADC_INIT`. Usage of the synthesizer must be Prepared by the `fq1b receive` command at the beginning of the pulse program, actually frequency setting is performed by the `fq8b` command by default linked to the `ACQ_O1B_list` parameter. By setting suitable frequencies you can see the signal move in the reconstruction display.

Phase cycling is often used to remove offsets during averaging. By varying the transmitter phase in 90deg steps and combining the complex data of the scans acquired this way accordingly, offsets related to the receiver hardware will cancel out.



To implement phase cycling, add an averaging loop `lo to start times NA`. Use different phase lists for transmit, reference and receiver phase. Make sure that phase pointers are incremented within the averaging loop. Use a phase cycle 0 90 180 270 deg for transmitter and receiver phase and observe that the data adds up, when `NA = 4` is selected. When the receiver phase is kept constant, the signal will cancel out however. Note that the same effect seemss to appear for the reference phase but in fact changing the reference phase will not cancel out offset effects, because data combination is not affected.

4.3 Extensions

- Observe the effect of the `ACQ_scan_shift` parameter on the `DEOSC` parameter

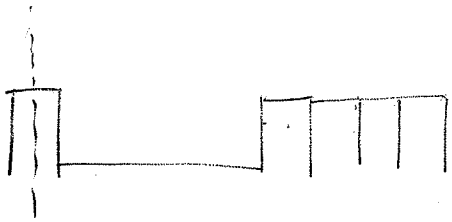
4.4 Hints

- Define the delay `deosc` in the pulse program and import it from the ParaVision parameter `DEOSC`. Rescale the value in a relation from microseconds to seconds:
  ```
  define delay deosc = {$DEOSC}
  "deosc = abs(deosc)*1e-6 - aqq"
  ```
- Set the parameter `DEOSC` to a negative value to have the relations calculate the minimum value for `DEOSC`.
- Initialize the parameter `ACQ_O1B_list_size` with 3 and set the values of `ACQ_O1B_list={-1000,2000,0}`.
- For phase cycling, define three phase lists `ph0, ph1, ph2` after the `exit` command.
- Use the phase command `ph0` to set the transmitter phase `p0 ph0:f1`.
- Use the phase command `ph2` for the receiver reference and `ph1` for the receiver mixing phase.
- Don't forget to increment the phase lists in the main acquisition loop using the `ipp0, ipp1, ipp2` commands.
- Set the parameter `NA = 4` to average data
- Set the parameter `NI = 4` to observe the different phase settings simultaneously.

4.5 Solution

```
;
; PPC 2006 Pulse Programming Course
;
; Project 4: data acquisition details
;
; Explicit parameters
;
; p0 - excitation pulse delay
; d0 - relaxation delay
; d8 - CONFIG_amplifier_enable per convention
; t0 - trim value ACQ_trim[0]
; NI - image object loop
; NA - averaging loop

;-------necessary include files---------------------------------
#include<Avance.incl>
#include <DBX.include>

;-------internal parameter definitions and relations-------------
define delay deosc = { $DEOSC }
"deosc = abs(deosc) * 1e-6 - aqq"
define delay ramp = { $PREEMP_rise_time }
"ramp = ramp * 1e-6"
define delay dr
"dr = aqq / 2 "


; automatic pulse gating switched off for fast sequences
preset off
; use second synthesizer for LO frequency
fq1b receive

;label--delays--commands----gradients-------remarks--------------
start,  d0 fq8b:f1                          ; select LO freq.

;-------excitation----------------------------------------------
        d8  gatepulse 1                     ; pulse gating
        (p0:sp0 ph0):f1                     ; pulse TPQQ[0]

;-------dephasing gradient--------------------------------------
        dr grad { (0)-(t0) | (0) | (0) }
        ramp groff

;-------refocussing gradient------------------------------------
        ramp grad { (t0) | (0) | (0) }      ; Refocussing
;-------data acquisition----------------------------------------
        depa    REC_ENABLE                  ; open receiver
        ADC_INIT_B(ph2, ph1)                ; prepare acquisition
        aqq ADC_START                       ; start data sampling
        deosc   groff                       ; minimum eosc delay
        10u     ADC_END                     ; end data sampling
        10u     ipp0 ipp1 ipp2

;-------data loops----------------------------------------------
lo to start times NI                        ; NI "image" objects
```

```
lo to start times NA                                    ; averaging loop
        10u     rpp0 rpp1 rpp2                          ; reset phases

SETUP_GOTO(start)

exit

;-------phase definitions-----------------------------------------
ph0 = 0 2 1 3                                           ; excitation phase
ph1 = 0                                                 ; reference phase
ph2 = 0 2 1 3                                           ; receiver phase
```



5 Gradients and Pulse Frequency

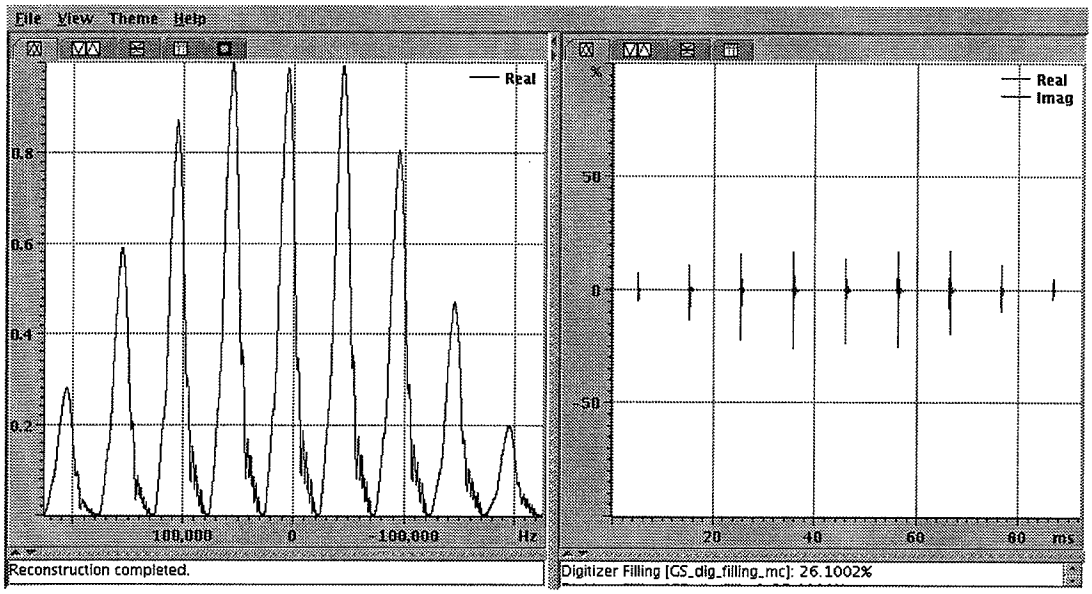
5.1 Objectives

By developing this project you will learn
- to use a slice selection gradient
- to vary the transmitter frequency for pulses

5.2 Project

In this project you extend the gradient echo developed in project 3 by slice selection and by varying the excitation frequency to acquire profiles for different slices.

To implement slice selection, the slice gradient must be applied during the excitation pulse and refocused afterwards. For symmetric excitation pulses the integral of the refocusing gradient must be the inverse of the integral from the middle of the pulse to the end. The profile of the excited slice depends on the gradient strength and the bandwidth of the applied excitation pulse.



To implement the project, clone the previous scan, copy the pulse program to `project5.ppg`. Switch the slice gradient on immediately before the excitation pulse and switch it to the opposite polarity immediately after the pulse. Switch the gradient off after half of the the pulse duration. Use the trim value `t1` to control the gradient strength
In order to change the transmitter frequency, apply a frequency list at the beginning of the acquisition loop to frequency channel 1. Remember to change the frequency after each excitation. Use the `ACQ_O1_list` / `ACQ_O1_list_size` parameters to define different offset frequencies and set `NI` to `ACQ_O1_list_size` in order to observe the different slice profiles simultaneously.

5.3 Extensions

Use a different list syntax for frequency switching – remember that the `fq1-fq8` commands include an automatic increment of the acquisition pointer.

5.4 Hints

- You can start an adapted parameter editor
  ```
  pvcmd -a pvScan pvEditPars PULPROG  NI D[0] P[0] TPQQ[0].name
  TOQQ[0].power ACQ_n_trim ACQ_trim[0] ACQ_trim[1]
  ACQ_O1_list_size ACQ_O1_list
  ```
- If you don't use the built-in frequency setting commands `fq1-fq8`, do not forget to switch increment the frequency list at suitable points.

5.5 Solution

```
;
; PPC 2006 Pulse Programming Course
;
; Project 4: data acquisition details
;
; Explicit parameters
;
; p0 - excitation pulse delay
; d0 - relaxation delay
; d8 - CONFIG_amplifier_enable per convention
; t0 - trim value ACQ_trim[0]
; t1 - trim value ACQ_trim[1] : slice selection
; NI - image object loop
; NA - averaging loop

;-------necessary include files---------------------------------
#include<Avance.incl>
#include <DBX.include>

;-------internal parameter definitions and relations-------------
define delay deosc = { $DEOSC }
"deosc = abs(deosc) * 1e-6 - aqq"
define delay ramp = { $PREEMP_rise_time }
"ramp = ramp * 1e-6"
define delay dr
"dr = aqq / 2 "
define delay dq
"dq = ramp - d8 - 10u"
define delay df
"df = (p0 + ramp) / 2"

; automatic pulse gating switched off for fast sequences
preset off
; use second synthesizer for LO frequency
fq1b receive

;label--delays--commands----gradients-------remarks--------------
start,  d0 fq8b:f1                          ; select LO freq.

;-------excitation with slice selection -------------------------
        dq      grad { (0) | (0) | (t1) }
        3u      fq1:f1                      ; set transmit freq.
        d8  gatepulse 1                     ; pulse gating
        (p0:sp0 ph0):f1                     ; pulse TPQQ[0]
        ramp    groff
;-------slice selection refocussing----------------------------
        df      grad { (0) | (0) | (0) - (t1) }
        ramp    groff

;-------dephasing gradient---------------------------------------
        dr grad { (0)-(t0) | (0) | (0) }
        ramp groff

;-------refocussing gradient-------------------------------------
        ramp grad { (t0) | (0) | (0) }      ; Refocussing
```

```
;-------data acquisition-----------------------------------------
        depa    REC_ENABLE                      ; open receiver
        ADC_INIT_B(ph2, ph1)                    ; prepare acquisition
        aqq ADC_START                           ; start data sampling
        deosc   groff                           ; minimum eosc delay
        10u     ADC_END                         ; end data sampling
        10u     ipp0 ipp1 ipp2

;-------data loops----------------------------------------------
lo to start times NI                            ; NI "image" objects
lo to start times NA                            ; averaging loop
        10u     rpp0 rpp1 rpp2                  ; reset phases

SETUP_GOTO(start)

exit

;-------phase definitions---------------------------------------
ph0 = 0 2 1 3                                   ; excitation phase
ph1 = 0                                         ; reference phase
ph2 = 0 2 1 3                                   ; receiver phase
```

6 Gradient Cycling

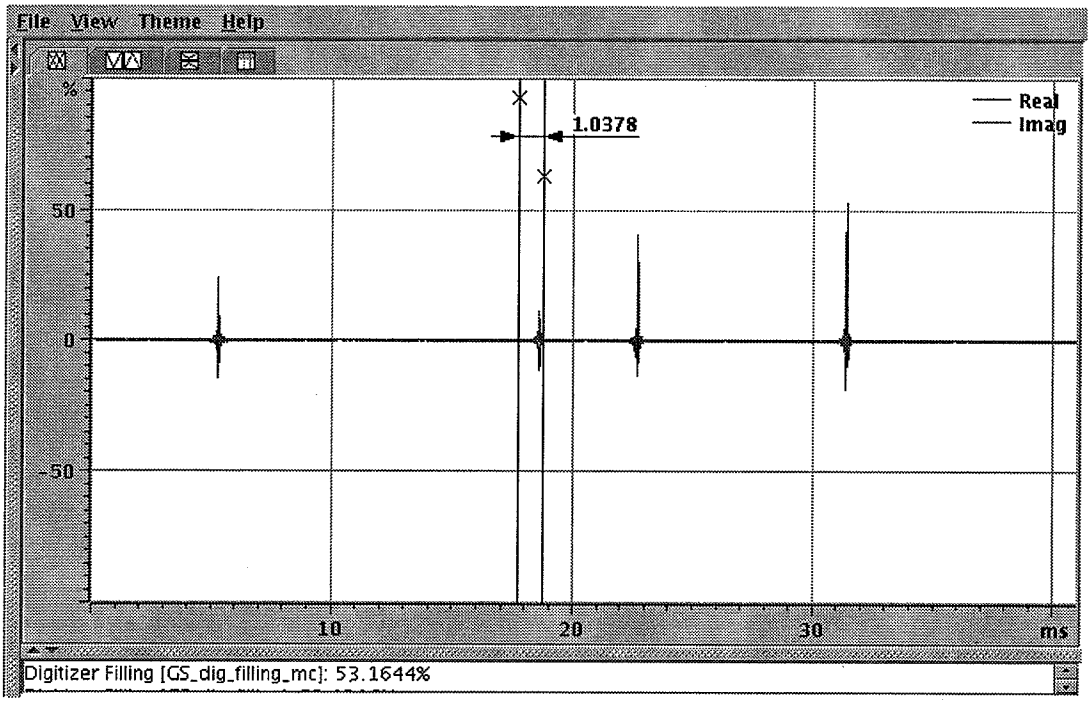
6.1 Objectives

By developing this project you will learn
- to vary gradients during runtime
- to use parameters to control gradient variation
- to switch off gradients during setup mode

6.2 Project

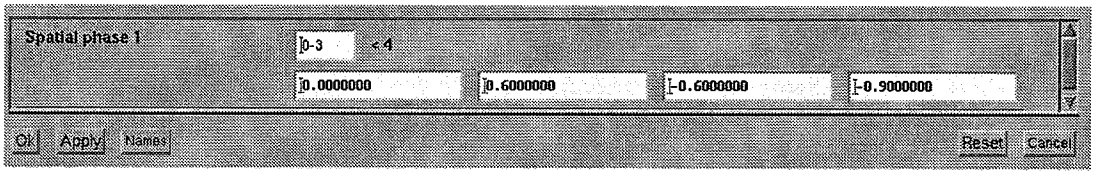
In this project you will learn how to vary the strength of a gradient linearly within a pulse sequence. This is the basic for phase encoding in many imaging sequences. You will learn to implement such a gradient "ramp" within a pulse program and understand default links to acquisition parameters.

Under certain conditions it is desirable to acquire a gradient echo not at the middle but at a different position. One way to vary the echo position is to vary the strength of the dephasing gradient. That is what we are going to do in this experiment.



We start by creating a new dataset and copying the pulse program from project 3 to `project6.ppg`. Then the trim value of the dephasing gradient will be modified by adding `r2d(t0)` ramp to the gradient value. In the pulse sequence, the length of the gradient ramp must be declared by a statement `lgrad r2d`. The ramp function must be incremented by an `igrad r2d` statement for each loop cycle. To have full access to default parameters for the `r2d` ramp, we have to increase the dimension of the experiment `ACQ_dim = 2`. E.g. `ACQ_size[1] = 4`. Then `lgrad r2d = ACQ_size[1]` and the loop over NI should be

replaced by a loop over `ACQ_size[1]`. The discrete points used for the ramp function can be influenced by the `ACQ_phase_encoding_mode[1]` parameter. The easiest way to observe this is, to select a `User_Defined_Encoding` and select discrete values in the range -1 to +1 in the parameter `ACQ_spatial_phase_1` (the size of the array is controlled by the parameter `ACQ_spatial_size_1`).



6.3 Extensions

Use the `<2d>` modifier in the ramp definition to restrict the gradient function to a single value in setup mode.

6.4 Hints

- To observe several scans simultaneously in the display, select `ACQ_phase_factor = ACQ_size[1]`
- You can start an adapted parameter editor

```
pvcmd -a pvScan pvEditPars PULPROG  ACQ_dim ACQ_size[1]
ACQ_phase_factor D[0] P[0] TPQQ[0].name TOQQ[0].power
ACQ_n_trim ACQ_trim[0] ACQ_phase_encoding_mode[1]
ACQ_spatial_size_1 ACQ_spatial_phase_1
```

6.5 Solution

```
;
; PPC 2006 Pulse Programming Course
;
; Project 4: data acquisition details
;
; Explicit parameters
;
; p0 - excitation pulse delay
; d0 - relaxation delay
; d8 - CONFIG_amplifier_enable per convention
; t0 - trim value ACQ_trim[0]
; t1 - trim value ACQ_trim[1] : slice selection
; ACQ_size[1] - image object loop
; NA - averaging loop

;-------necessary include files---------------------------------
#include<Avance.incl>
#include <DBX.include>

;-------internal parameter definitions and relations-------------
define delay deosc = { $DEOSC }
"deosc = abs(deosc) * 1e-6 - aqq"
define delay ramp = { $PREEMP_rise_time }
"ramp = ramp * 1e-6"
define delay dr
"dr = aqq / 2 "
define delay dq
"dq = ramp - d8 - 10u"
define delay df
"df = (p0 + ramp) / 2"

; definition of gradient ramp
lgrad r2d<2d> = ACQ_size[1]

; automatic pulse gating switched off for fast sequences
preset off
; use second synthesizer for LO frequency
fq1b receive

;label--delays--commands----gradients-------remarks--------------
start,  d0 fq8b:f1                              ; select LO freq.

;-------excitation with slice selection -------------------------
        dq      grad { (0) | (0) | (t1) }
        3u      fq1:f1                          ; set transmit freq.
        d8  gatepulse 1                       ; pulse gating
        (p0:sp0 ph0):f1                         ; pulse TPQQ[0]
        ramp    groff
;-------slice selection refocusing-----------------------------
        df      grad { (0) | (0) | (0) - (t1) }
        ramp    groff

;-------dephasing gradient---------------------------------------
        dr grad { (0)-(t0)-r2d(t0) | (0) | (0) }
        ramp groff
```

```
;-------refocusing gradient-------------------------------------
        ramp grad { (t0) | (0) | (0) }      ; Refocusing
;-------data acquisition---------------------------------------
        depa    REC_ENABLE                  ; open receiver
        ADC_INIT_B(ph2, ph1)                ; prepare acquisition
        aqq ADC_START                     ; start data sampling
        deosc   groff                       ; minimum eosc delay
        10u     ADC_END                     ; end data sampling
        10u     ipp0 ipp1 ipp2 igrad r2d    ; increment ramp

;-------data loops----------------------------------------------
lo to start times ACQ_size[1]               ; encoding steps
lo to start times NA                        ; averaging loop
        10u     rpp0 rpp1 rpp2              ; reset phases

SETUP_GOTO(start)

exit

;-------phase definitions---------------------------------------
ph0 = 0 2 1 3                               ; excitation phase
ph1 = 0                                     ; reference phase
ph2 = 0 2 1 3                               ; receiver phase
```

Applied Pulse Programming

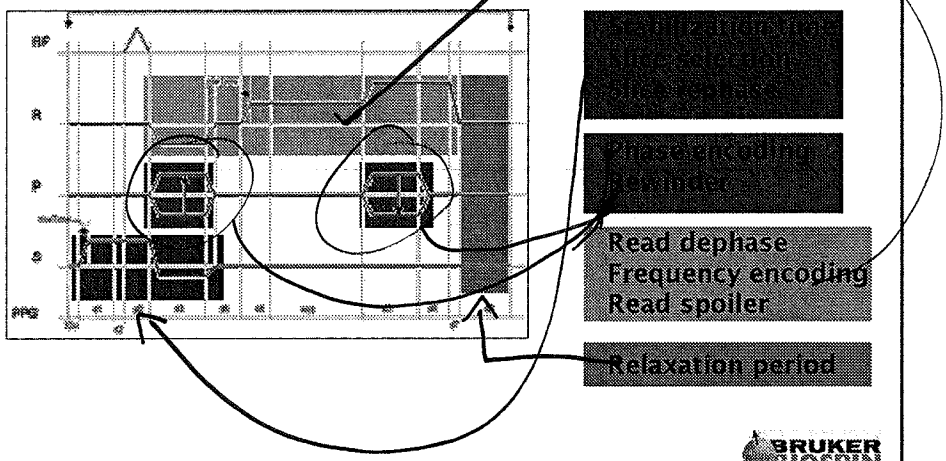
ParaVision Programming course
April 03-07, 2006

J. VOIRON

INTRODUCTION

- Interpretation of the „gre“ pulse program (PPG)
- Project 1 : multi-frames & 3D
- Project 2.1: slice refocussing
- Project 2.2 : slice thickness

PROJECT1: „gre“ ppg interpretation



PROJECT1: „gre“ ppg interpretation

- *initialization of 2d phase & slice loops*
- start,
- *set the emitter/receiver frequency offset*
- stabilization time
- slice selective excitation
- slice rephase
- phase encoding } ← Applied at the same time (timing optimization)
- read dephase
- frequency encoding
- read spoiler
- phase rewinder } ← Applied at the same time (timing optimization)
- relaxation period
- *increment of slice gradient*
- lo to start times NSLICES
- *reset of slice gradient*
- lo to start times NA
- *increment of 2d phase gradient*
- lo to start times ACQ_size[1]
- *reset of 2d phase gradient*
- lo to start times NR
- *phase cycling list*

PROJECT1: „gre“ ppg interpretation

necessary because the last rep. ended with a spoiler.

```
; stabilization time

10u fq8b:f1 (1)
d1 fq1:f1   (2)

(1): only activated with the
ADC_init_B macro

(2): immediatly switched on
```

set Tx offset frequency.

set Tx.

start,
stabilization time
slice selective excitation
slice rephase
phase encoding
read dephase
frequency encoding
read spoiler
phase rewinder
relaxation period
lo to start times NSLICES
NA
ACQ_size[1]
NR

PROJECT1: „gre“ ppg interpretation

```
; slice selection

d1 fq1:f1 grad{(0)|(0)|(t0)}
d2 gatepulse 1
(p0:sp0 ph1):f1
```

start,
stabilization time
slice selective excitation
slice rephase
phase encoding
read dephase
frequency encoding
read spoiler
phase rewinder
relaxation period
lo to start times NSLICES
NA
ACQ_size[1]
NR

PROJECT1: „gre“ ppg interpretation

```
; phase encoding

d3 grad{(t2)|r2d(t3)|(t1)}
d5 groff
```

start,
- stabilization time
- slice selective excitation
- **slice rephase**
- **phase encoding** } simultaneous
- read dephase
- frequency encoding
- read spoiler
- phase rewinder
- relaxation period

lo to start times — NSLICES, NA, ACQ_size[1], NR

PROJECT1: „gre“ ppg interpretation

```
; frequency encoding

denab REC_ENABLE grad{(t5)|(0)|(0)}
      ADC_INIT_B(ph1,ph0)
aqq   ADC_START
```

Rx phase, ref. phase.

start,
- stabilization time
- slice selective excitation
- slice rephase
- phase encoding
- read dephase
- frequency encoding
- read spoiler
- phase rewinder
- relaxation period

lo to start times — NSLICES, NA, ACQ_size[1], NR

PROJECT1: „gre“ ppg interpretation

```
; read spoiler & phase encoding

d3 grad{(t8)|r2d(t6)|(0)}
d6 grad{(t8)|(0)|(0)}
d7 groff
```

start,
- stabilization time
- slice selective excitation
- slice rephase
- phase encoding
- read dephase
- frequency encoding
- read spoiler
- **phase rewinder**] simultaneous
- relaxation period

lo to start times
- NSLICE
- NA
- ACQ_size[1]
- NR

PROJECT1: „gre“ ppg interpretation

```
; relaxation delay

d0 ADC_END
```

start,
- stabilization time
- slice selective excitation
- slice rephase
- phase encoding
- read dephase
- frequency encoding
- read spoiler
- phase rewinder
- **relaxation period**

lo to start times
- NSLICE
- NA
- ACQ_size[1]
- NR

PROJECT1: 3D generalization of „gre“

- Use a new *r3d* ramp function associated to the *l1* loop (to initialize and to add in the loop structure)
- Apply it in the slice selection direction:

```
d3 grad{(t2)|r2d(t3)|(t1)+ r3d(t4)}
```

- Don't forget to add it for the phase rewinder as well:

```
d3 grad{(t8)|r2d(t6)|r3d(t7)}
```

PROJECT1: 3D generalization of „gre“

- Test procedure:
  - Switch the spatial acquisition parameter to „3D“
  - sinc10H/1ms/30°
  - Matrix = 128x64x64
  - Slice thickness = 20mm
  - TR / TE = 30/3.5ms

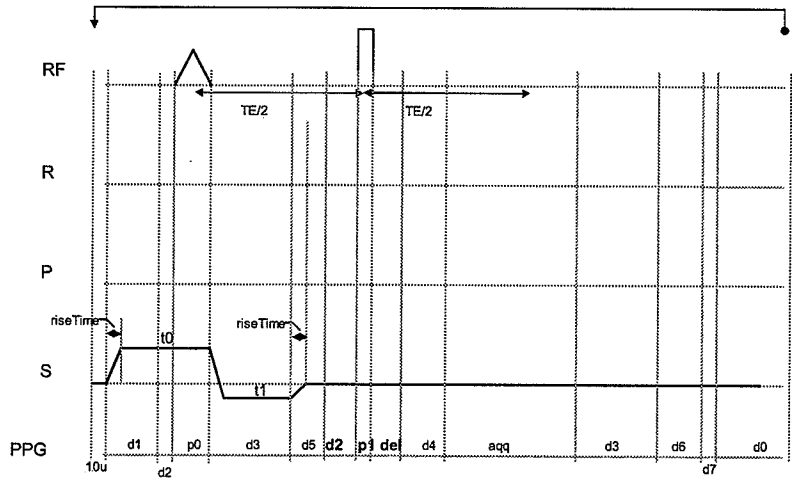
PROJECT1: multi-frames "gre"

- Implement a new movie loop (l10) (*)
- Initialize L[10] and NI accordingly (*)
- Modify the object ordering parameter (ACQ_obj_order in ACQ_INFO)

(*) refer to the section 2.2 of your handout...

- Test procedure:
  - Acquire 3 slices at 3 very different positions and 4 frames per slice.
  - Check your image sorting...

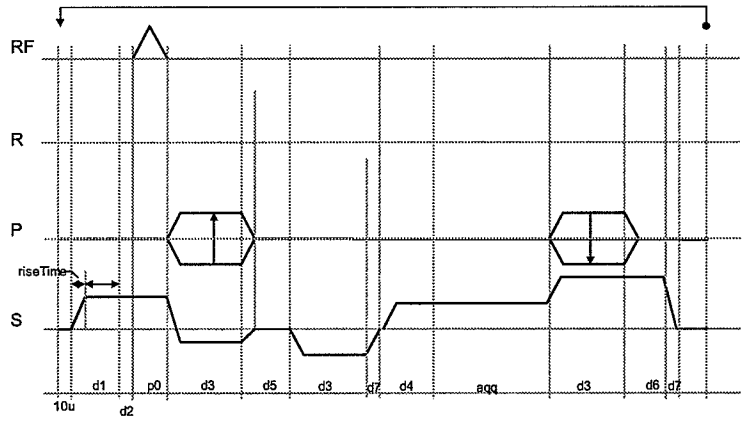
PROJECT 2.1: slice refocussing



PROJECT 2.1: slice refocussing

- Follow the realization hints in section 3.4.1 of your handout (page16)…

PROJECT 2.2: slice thickness



PROJECT 2.2: slice thickness

- Implement the modifications according to the sequence diagram of the sequence (also, look at the section 3.4.2 of your handout...)
- Change the *ACQ_grad_matrix [i][j][k]* :

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Image ***Image of Slice***

PROJECT 2.2: slice thickness

- Export the scan into TOPSPIN
- Extract the middle FID (i.e.#64) from the 2D dataset: type „*rser*“
- Fourier transform: type „*FT*“
- rephase the spectrum (0 and 1 phase correction order) until you get the right slice profile
- Derive the slice thickness using:

$$\Delta Z = (FOV * \Delta f) / SW\_h$$

Applied Pulse Programming

ParaVision Programming course
April 03-07, 2006

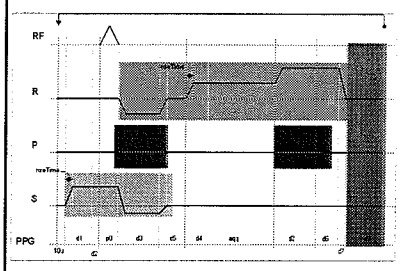
J. VOIRON

INTRODUCTION

- **Interpretation** of the *gre*(*) pulse program (*gre.ppg*)
- **Project 1** : multi-frames & 3D
- **Project 2.1**: slice refocussing
- **Project 2.2** : slice thickness

(*) simplified version of a gradient echo imaging (FLASH)

PROJECT1: „gre" ppg interpretation



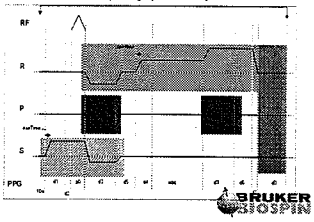
Stabilization time
Slice selection
Slice rephase

Read dephase
Frequency encoding
Read spoiler

Relaxation period

PROJECT1: „gre" ppg interpretation

- *Definition of delays*
- *initialization of 2d phase & slice loops*
- start,
- *set the emitter/receiver frequency offset*
- stabilization time
- slice selective excitation
- slice rephase
- **phase encoding** } ← Applied at the same time (timing optimization)
- read dephase
- frequency encoding
- read spoiler } ← Applied at the same time (timing optimization)
- **phase rewinder**
- relaxation period
- *increment of slice gradient*
- lo to start times NSLICES
- *reset of slice gradient*
- lo to start times NA
- *increment of 2d phase gradient*
- lo to start times ACQ_size[1]
- *reset of 2d phase gradient*
- lo to start times NR
- *phase cycling list*

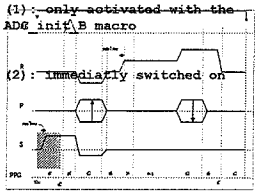


PROJECT1: „gre" ppg interpretation

```
10u fq8b:f1(1)
d1 fq1:f1(2) grad{(0)|(0)|(t0)}
```

(1) : only activated with the ADC_init_B macro

(2) : immediatly switched on

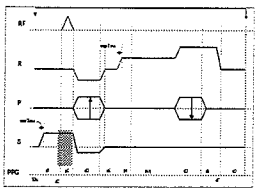


start,
stabilization time
slice selective excitation

lo to start times NSLICES
NA
ACQ_size[1]
NR

PROJECT1: „gre" ppg interpretation

```
d2 gatepulse 1
(p0:sp0 ph1):f1
```

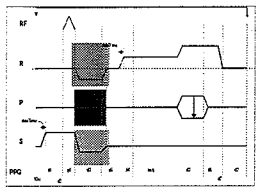


start,
stabilization time
slice selective excitation
slice rephase

lo to start times NSLICES
NA
ACQ_size[1]
NR

PROJECT1: „gre“ ppg interpretation

```
d3 grad{(t2)|z2d(t3)|(t1)}
d5 groff
```

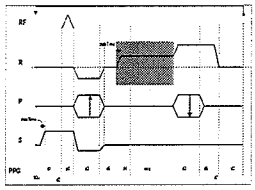


start,
- stabilization time
- slice selective excitation
- slice rephase
- **phase encoding**
- read dephase
- frequency encoding

lo to start times — NSLICES
NA
ACQ_size[1]
NR

PROJECT1: „gre“ ppg interpretation

```
denab REC_ENABLE grad{(t5)|(0)|(0)}
      ADC_INIT_B(ph1,ph0)
aqq   ADC_START
```

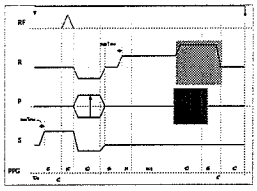


start,
- stabilization time
- slice selective excitation
- slice rephase
- phase encoding
- read dephase
- frequency encoding
- read spoiler

lo to start times — NSLICES
NA
ACQ_size[1]
NR

PROJECT1: „gre“ ppg interpretation

```
d3 grad{(t8)|z2d(t6)|(0)}
d6 grad{(t8)|(0)|(0)}
d7 groff
```

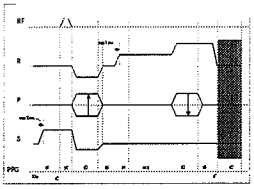


start,
- stabilization time
- slice selective excitation
- slice rephase
- phase encoding
- read dephase
- frequency encoding
- read spoiler
- **phase rewinder**
- relaxation period

lo to start times — NSLICE
NA
ACQ_size[1]
NR

PROJECT1: „gre“ ppg interpretation

```
d0 ADC_END
```



start,
- stabilization time
- slice selective excitation
- slice rephase
- phase encoding
- read dephase
- frequency encoding
- read spoiler
- phase rewinder
- **relaxation period**

lo to start times — NSLICE
NA
ACQ_size[1]
NR

PROJECT1: „gre“ ppg interpretation

```
#include<Avance.incl>
#include <DBX.include>
preset off

define delay denab
"denab = d4 - de + depa"

lgrad z2d<2d> = L[0]
zgrad z2d
lgrad nslice = NSLICES
Zslice

#include <MEDSPEC.include>

start,    10u          fq8b:f1
          d1           fq1:f1                    grad{( 0)| (0)| (0)}
          d2           gatepulse 1
          (p0:sp0 ph1):f1
          d3                                     grad{(t2)|z2d(t3)|(t1)}
          d5                                     groff
          denab        REC_ENABLE                grad{(t5)| (0)| (0)}
                       ADC_INIT_B(ph1, ph0)
          aqq          ADC_START
          d3                                     grad{(t8)|z2d(t6)|(0)}
          d6                                     grad{(t8)| (0)| (0)}
          d7                                     groff
          d0           ADC_END
          1u           lslice
lo to start times NSLICES
          1u           rpp1         zslice
lo to start times NA
          1u           rpp1         lgrad z2d
lo to start times lo
          1u
lo to start times NR
goto start
Exit

;---------------------------------------------------------------
ph0 = 0
ph1 = 0
;---------------------------------------------------------------
```

PROJECT IMPLEMENTATION: Recommendations

- All implementation will be based on the ***gre*** method:
  to load from the Parameter Editor for MethodClass (from a scan status *„New“*)
- All proposed pulse programs are stored under:
  /opt/PPC/PPG/AppliedPPG
- In order to be launched, these PPGs must be copied under:
  /opt/PV4.0/exp/stan/nmr/lists/pp

PROJECT1: multi-frames „gre“

- In PPG: Implement a new movie loop (l10) (*)
- In ACQP: Initialize L[10] and NI accordingly (*)
- In ACQP: Modify the object ordering parameter (*)
  (ACQ_obj_order in ACQ_INFO)

Note: **the ACQP parameters associated to the frame loop (L[10], NI, ACQ_obj_order) are NOT implemented in the source code of the method**
➔ to set manually

(*) refer to the section 2.5.1 of your handout...

- Test procedure:
  - Acquire 3 slices at 3 very different positions and 4 frames per slice.
  - Check your image sorting...

Note: also, change the RECO_rotate parameter: **(0, 0)➔(0.5,0)**

PROJECT1: multi-frames „gre“

```
initialization of 2d phase & slice loops
start,
set the emitter/receiver frequency offset
                stabilization time
                    slice selective excitation
                slice rephase
                 phase encoding
                read dephase
                     frequency encoding
                read spoiler
                phase rewinder
                   relaxation period
increment of slice gradient
lo to start times NSLICES
reset of slice gradient
    lo to start times l10
        lo to start times NA
            increment of 2d phase gradient
            lo to start times ACQ_size[1]
            reset of 2d phase gradient
                lo to start times NR
phase cycling list
```

PROJECT1: multi-frames „gre“

- **Image sorting:**
  - **Image display (output):**

slice1: fr1(0) *fr2*(1) *fr3*(2) *fr4*(3)
slice2: fr1(4) *fr2*(5) *fr3*(6) *fr4*(7)
slice3: fr1(8) *fr2*(9) *fr3*(10) *fr4*(11)

  - **Acquisition order (input):**

Sequential:

frame1: sl1(0) *sl2*(4) *sl3*(8)
frame2: sl1(1) *sl2*(5) *sl3*(9)
frame3: sl1(2) *sl2*(6) *sl3*(10)
frame4: sl1 (3) *sl2*(7) *sl3*(11)

Interlaced:

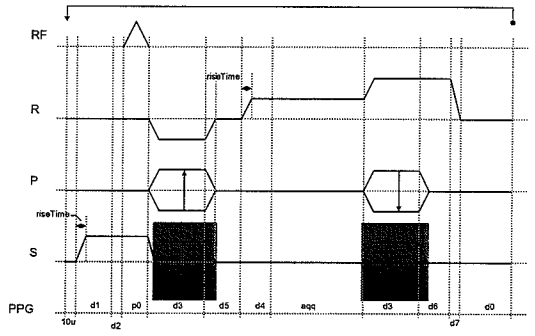
frame1: sl1(0) *sl3*(8) *sl2*(4)
frame2: sl1(1) *sl3*(9) *sl2*(5)
frame3: sl1(2) *sl3*(10) *sl2*(6)
frame4: sl1(3) *sl3*(11) *sl2*(7)

➔ACQ_obj_order =

(0) (4) (8) (1) (5) (9) (2) (6) (10) (3) (7) (11)

(0) (8) (4) (1) (9) (5) (2) (10) (6) (3) (11) (7)

PROJECT1: 3D generalization of „gre“



PROJECT1: 3D generalization of „gre“

- In PPG: Use a new r3d ramp function associated to the l1 loop (to initialize and to add in the loop structure) (*)
- In PPG: Apply it in the slice selection direction: (*)

```
d3 grad{(t2)|r2d(t3)|(t1)+ r3d(t4)}
```

- In PPG: Don't forget to add it for the phase rewinder as well: (*)

```
d3 grad{(t8)|r2d(t6)|r3d(t7)}
```

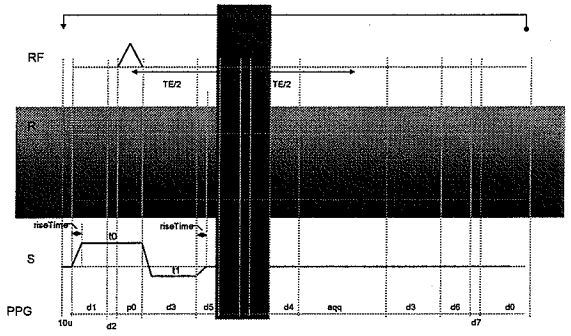
Note: **the ACQP parameters associated to the third dimension (L[1], ACQ_trim[4] & [7]) are already implemented in the source code of the method.**

(*) refer to the section 2.5.2 of your handout...

PROJECT1: 3D generalization of „gre“

- Test procedure:
  - Switch the spatial acquisition parameter to „3D“ In StandardInplaneGeometry class
  - sinc10H/1ms/30°
  - Matrix = 128x64x64
  - Slice thickness = 20mm
  - TR / TE = 30/3.5ms

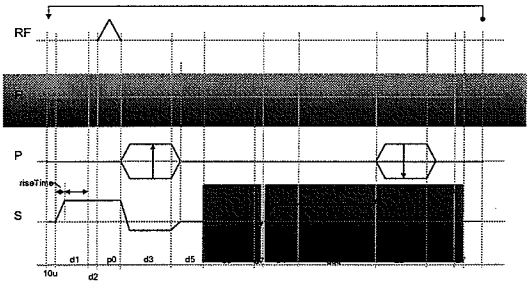
PROJECT 2.1: slice refocussing



PROJECT 2.1: slice refocussing

- Follow the realization hints in section 3.4.1 of your handout (based on the sequence diagram)
- Test procedure:
  - Select 1 centered slice
  - [redacted]
  - [redacted]
  - [redacted]
  - P[1]=200us
  - TPQQ[1] name = „bp.exc"
    power = PVM_RefAtt - 20 log(1000us/200us) - 6[dB]
    offset = 0Hz
  - GSP mode: observe the echo position when changing the slices in Z-dir working in axial orientation

PROJECT 2.2: slice thickness



PROJECT 2.2: slice thickness

- Implement the modifications according to the sequence diagram (also, look at the section 3.4.2 of your handout...)
- Change the ACQ_grad_matrix [i][j][k] :

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Image ***Image of Slice***

PROJECT 2.2: slice thickness

Signal processing procedure:

- Export the scan into TOPSPIN
- Extract the middle FID (i.e.#64) from the 2D dataset: type „***rser***"
- Fourier transform: type „***FT***"
- rephase the spectrum (0 and 1 phase correction order) until you get the right slice profile
- Derive the slice thickness using:

$$\Delta Z = (FOV * \Delta f) / SW\_h$$

THANK YOU FOR YOUR ATTENTION

Applied Pulse Programming

ParaVision Programming Course
April 03-07, 2006

Author:
Jérôme VOIRON

Bruker BioSpin MRI GmbH

1 Some "rehashes" about the pipeline acquisition in ParaVision

1.1 Role of the pulse program

In ParaVision, a pulse program is just one part of a complete measurement method. The role of the pulse program is to produce experimental data points which can be analyzed and displayed afterwards by the process of reconstruction and image display. Within ParaVision, these basic parts of collecting data and the reconstruction process are linked together in the acquisition & reconstruction pipeline. The pipeline expects a fixed, but parameterized flow of the data. As a consequence of this pipeline architecture, the loop structure for the pulse program is fixed: see below.

1.2 Loop description

```
start,
                          ...
                          ...
                          ADC_START (of size ACQ_scan[0] )
                          ...
lo to start times
                                          ACQ_ns_list[i]
                                    ACQ_phase_factor
                               NI
                          NA
                     ACQ_size[1]
                     ...
                ACQ_size[n]
           NAE
NR
```

Distinguish between three types of "loop parameters":

1.2.1 Parameter describing the dimensionality of one object

ACQ_size[0, 8]	Dimension of the experiment. The standard reconstruction will apply a Fourier transformation in these dimensions.

1.2.2 Parameter describing the type of phase encoding

ACQ_phase factor	Segmentation type. i.e. RARE factor, EPI segmentation...

1.2.3 Parameter describing the number of reconstructed objects:

NI =		Number of images. Typically used for :
• NSLICES	<---->	• total number of slices (NSLICES)
• ACQ_ns_list_size	<---->	• echo images (T2 measurements)
• L[i]	<---->	• T1 weighted images (T1 measurement)
• L[i]	<---->	• diffusion encoding
• L[i]	<---->	• velocity encoding
• ...		• ...

Note: "L" is a user- defined loop counter

NA	Number of averages used before the phase-encoding increments.
NAE	Number of averaged experiment used after the phase-encoding increments.
NR	Number of repeated experiments. i.e. time course investigations

Particular case :	**Multi-echo experiment (MSME)**
`ACQ_ns_list[0, NI-1]`	Direct data averaging of size `ACQ_size[0]`. Application: echo collection in MSME
`NI`	Number of echo images x number of slices (ACQ_ns_list_size) (NSLICES)
`NA, NAE, NR`	Kept unchanged: can be still used.

At first glance, this pipeline structure seems to be somewhat complicated. However, we don't need to write down explicitly all of these loops in the pulse program as long as they are not required. Loops which are not required can be skipped by setting the corresponding base-level parameter to one.

1.3 Application to a practical example

A 2D multi-slices gradient echo sequence will be considered in the following of this chapter.

1.3.1 Sequence elements and loop structure for the gradient echo sequence

The main elements of such a gradient echo sequence are the following:

- relaxation delay within one slice
- delays between consecutive slices
- slice-selective excitation
- phase-encoding
- frequency-encoding

According to the previous loop descriptions, the required loop structure for the 2D gradient echo sequence is schematized below. Its main components are written in bold fonts and the linkage components are in italic.

Please, notice that the following loops were skipped in the following way:

- `ACQ_ns_list[i] = 1`
- `ACQ_phase_factor = 1`
- `NAE =1`
- `ACQ_size[i > 1] = 0`

```
Initialization of 2d phase loop
start,    relaxation delay within one slice
          set the emitter/receiver frequency offset
          slice selective excitation
          phase encoding
          frequency encoding
increment of slice gradient
lo to start times NSLICES (*)
reset of slice gradient
lo to start times NA
increment of 2d phase gradient
lo to start times ACQ_size[1]
reset of 2d phase gradient
lo to start times NR

[phase cycling list]
```

(*) implicitly related to NI (see § 1.2.3)

1.3.2 Ppg implementation strategy

From the previous example, one can derive the following strategy of ppg implementation:

> ***Combine together the described in-loop elements as well as the delays and gradient switching points.***
>
> ***Insert the loop counters and the slice frequency offsets as well as the phase gradient increment statements.***

2 PROJECT 1: Gradient Echo sequence ("gre")

2.1 Objectives

By developing this project you will learn how to:

- interpret a complex PPG: 2D "gre" sequence
- modify it in order to obtain a PPG capable of:
  - multi frames acquisition (typical use: cardio movies)
  - 3D acquisition: 3D "gre" sequence

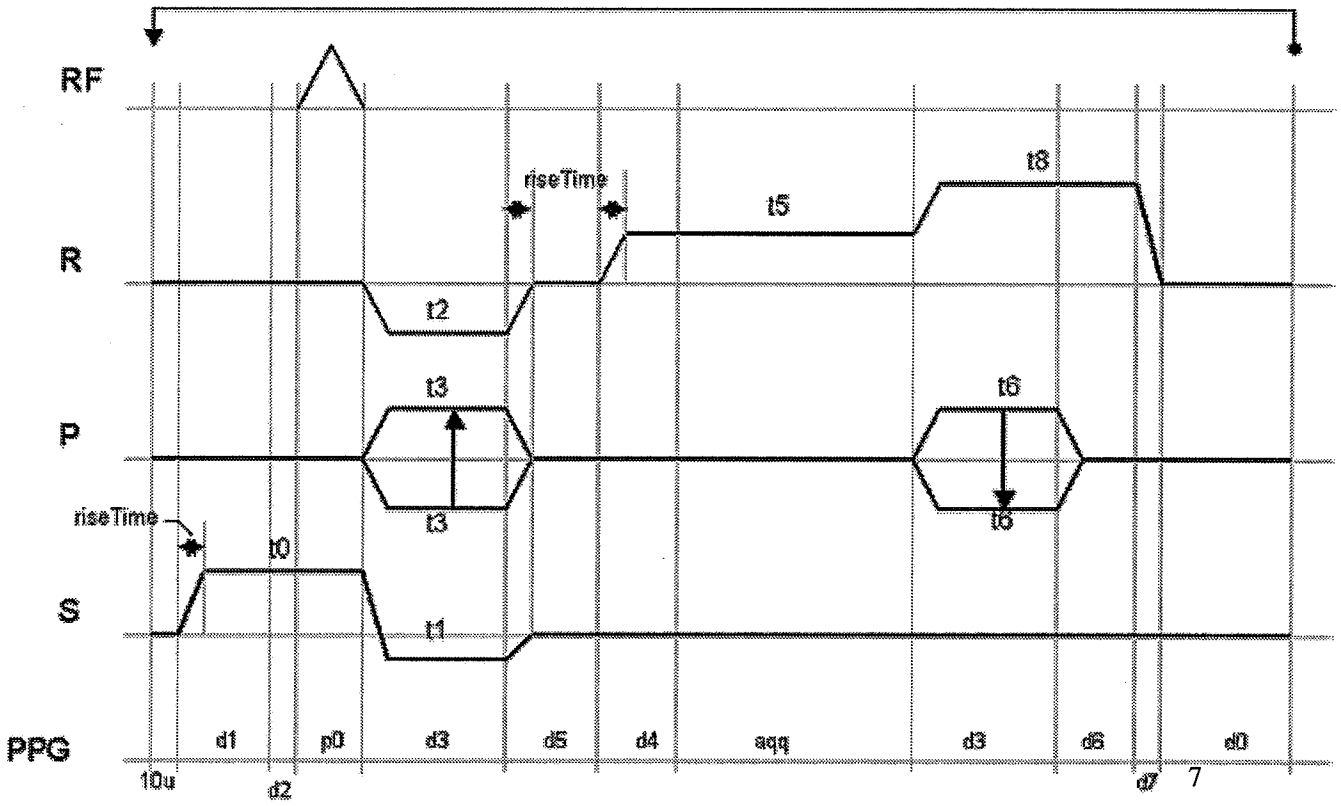
2.2 Project description

The first part of this project consists of analyzing a realistic 2D imaging sequence: *gre*. This is a simplified version of gradient echo imaging (FLASH) which will be used as a basis of all programming projects developed during this course.

In the next step you will modify the gre sequence to make it capable of acquiring movie frames. You should add an additional loop and set the corresponding ACQP parameters.

Finally, you will extend the gre sequence to a 3D acquisition.

2.3 "gre" method features



Note: the slice selection spoiler switching points have been combined in a way allowing optimizing the sequence timing.

- 2D gradient-echo sequence
- Multi-slices
- Multi-packages
- Multi-frames (movie mode)
- Multi-averaging
- Multi-repetitions
- No particular phase-cycling

The acquisition loop structure (from inner to outer loops):

1- slices

2- movie frames

3- accumulation

4- phase-encoding

5- repetitions

2.4 Interpretation of the "gre" pulse program (also look at the sequence diagram)

```
#include<Avance.incl>
#include <DBX.include>
preset off


;===========================================================
;   definition of delays
;===========================================================

define delay denab
"denab =  d4 - de + depa"

;===========================================================
;   declaration of 2d and 3d loop
;===========================================================


lgrad r2d<2d> = L[0]
zgrad r2d

lgrad slice = NSLICES
```

```
zslice
#include <MEDSPEC.include>

;=====================================================================
;Delay/Pulse                spec control          gradients
;=====================================================================
;------------------start of the main loop-----------------------

start,10u                fq8b:f1

;------------------slice selection------------------------------

      d1                 fq1:f1              grad{( 0)| (0) |(t0)}
      d2                 gatepulse 1
     (p0:sp0 ph1):f1


;------------------slice rephase, read dephase, phase encoding-------

      d3                                   grad{(t2)|r2d(t3)|(t1) }
      d5                                     groff

;------------------frequency encoding---------------------------------

      denab              REC_ENABLE        grad{(t5)| (0) | (0)}
                         ADC_INIT_B(ph1, ph0)
      aqq                ADC_START

;------------------read spoiler, phase encoding----------------------

      d3                                     grad{(t8)|r2d(t6)|(0)}
      d6                                     grad{(t8)| (0) | (0) }
      d7                                     groff
      d0                 ADC_END

;------------------slice loop-----------------------------------------

      1u                                     islice
lo to start times NSLICES

;------------------averaging loop-------------------------------------

      1u                 ipp1                zslice
lo to start times NA

;------------------2D loop--------------------------------------------

      1u                 rpp1                igrad r2d   ;2d loop
lo to start times l0

;------------------repetition loop------------------------------------
      1u
lo to start times NR
goto start
exit
;=============================================================
;phase lists
```

```
ph0 = 0
ph1 = 0
;=============================================================
```

2.5 Realization hints

1. **Implementation of the multi-frames loop**. First, copy gre.ppg in MovieGre.ppg without forgetting to update the PULPROG variable accordingly.
   ECG triggering considerations impose that all slices are acquired for each movie frame. Thus, you need:
   - to implement (considering the sections 2.2 and 2.3 of this course) a new movie loop (l10) placed at adequate position in the ppg loop structure,
   - to initialize L[10] and NI accordingly (for NI, refer to its definition at section 2.2 of this course),
   - to modify the object ordering parameter (ACQ_obj_order in ACQ_INFO) in order to get the right object sorting, namely all frames displayed for each slice in the expected order. Of course, the object ordering parameter directly depends on the object ordering mode: per default, this mode is set to Interlaced.

   In order to test this new feature, acquire 3 slices at 3 very different positions (one crossing the air bubble at the top, one at the centre and the last one at the bottom of the phantom) and set the number of frames to 4. Keep the object ordering mode to Interlaced. Check if your number of acquired frames fits with the expected one, and if they are displayed in the right order (4 frames acquired for each slice in the right order).

2. **Implementation of 3D acquisition**. For that purpose, copy first gre.ppg in 3DGre.ppg without forgetting to update the PULPROG variable accordingly.
   You need to implement a new r3d function and to initialize it with the L[1] parameter (associated to the l1 loop parameter). This second phase-encoding gradient (applied in the slice selection direction) will be applied during the slice rephasing lobe of gradient. Use the table below to help you changing the 2D pulse program in a 3D pulse program.
   In order to test this new feature, don't forget to switch the spatial acquisition dimension parameter (StandardInplaneGeometry class) to the 3D value (by default this parameter is set to 2D), and use the following parameters to obtain a good 3D image quality:
   - Sinc10H /1ms/30°
   - Matrix size = 128x64x64
   - Slice thickness = 20mm
   - TR = 30ms / TE = 3.5ms

PPG's COMMENTED PART	DELAY	TRIMS	COMMENTS
slice rephase, *read dephase,* *phase encoding*	d3 (slice dephasing / rephasing rise-time)	t1, t4	During the slice rephasing time, use *r3d* to set the phase-encoding gradient in the second direction.

Read spoiler, *Phase encoding*	d3 (+rise-time "up") d6 (+rise-time "down")	t7	

2.6 Proposed Solution

```
#include<Avance.incl>
#include <DBX.include>
preset off


;============================================================
;   definition of delays
;============================================================

define delay denab
"denab =  d4 - de + depa"

;============================================================
;   declaration of 2d and 3d loop
;============================================================


lgrad r2d<2d> = L[0]
zgrad r2d
lgrad r3d<3d> = L[1]
zgrad r3d

lgrad slice = NSLICES
zslice
#include <MEDSPEC.include>

;=====================================================================
;Delay/Pulse                spec control          gradients
;=====================================================================

;------------------start of the main loop-----------------------

start,10u                fq8b:f1

;------------------slice selection-------------------------------

      d1                 fq1:f1             grad{( 0)| (0) |(t0)}
      d2                 gatepulse 1
     (p0:sp0 ph1):f1

;------------------slice rephase, read dephase, phase encoding --

      d3                                  grad{(t2)|r2d(t3)|(t1)+r3d(t4)}
      d5                                    groff

;------------------frequency encoding----------------------------

      denab              REC_ENABLE         grad{(t5)| (0) | (0)}
```

```
                    ADC_INIT_B(ph1, ph0)
     aqq            ADC_START



;-----------------read spoiler, phase encoding---------------------

     d3                                 grad{(t8)|r2d(t6)|r3d(t7)}
     d6                                 grad{(t8)| (0) | (0) }
     d7                                 groff
     d0             ADC_END

;-----------------slice loop-----------------------------------------

     1u                                 islice
lo to start times NSLICES

;-----------------movie loop-----------------------------------------

     1u                                 zslice
lo to start times l10

;-----------------averaging loop-------------------------------------

1u              ipp1
lo to start times NA

;-----------------2D loop--------------------------------------------

     1u             rpp1               igrad r2d   ;2d loop
lo to start times l0

;-----------------3D loop--------------------------------------------

     1u                                 igrad r3d   ;3d loop
lo to start times l1

;-----------------repetition loop------------------------------------

lo to start times NR
goto start
exit
;============================================================
;phase lists

ph0 = 0
ph1 = 0
;============================================================
```

Note: the lines written in bold fonts correspond to "gre" ppg's modifications according to the Project1 specification.

3 PROJECT 2: Slice Selection Test Procedures

Before running your pulse program in a routine mode, it can be necessary to test for instance the slice selection you have implemented. Two properties of the slice selection pulses can be easily verified simply by modifying your 2D Gradient Echo pulse program. We will thus check:

- the slice rephasing property
- the slice thickness property

3.1 Objectives

By developing this project you will learn how to:

- introduce a new π non-selective pulse in your "gre" pulse program file
- modify some ACQP parameters influencing your pulse program and in particular:
  - ACQ_trim parameters
  - ACQ_grad_matrix parameters

3.2 Projects description

3.2.1 Project 2.1: Slice refocusing

Based on the former 2D gradient echo ("gre") pulse program, you will implement a (π/2 selective) – (π non-selective) sequence. The non-selective π pulse must be a block pulse (bp) placed at TE/2. Because we are only interesting in checking the reliability of the slice refocusing pulse, the readout and phase-encoding gradients will be ignored.
From a theoretical point of view, if the phase dispersion produced by the slice selection gradient has been properly rephased, then the echo position must not be dependent from any local magnetic field (B0). Practically and assuming the slice selection to be along the z-axis, it means that changing the values of the z-shim must not change the echo position (only the echo shape is expected to change). We will check this property.

3.2.2 Project 2.2: Slice thickness

By slightly modifying the timing of the 2D gradient echo ("gre") pulse program and setting the readout gradient in the same direction than the slice gradient (modify the ACQ_grad_matrix array parameter), you will able to generate an excitation profile.
The slice thickness will be checked by exporting the acquired 2D image, extracting a single row of the 2D dataset and processing the corresponding signal into TopSpin. The slice thickness will be derived from the slice profile according to the following formula:

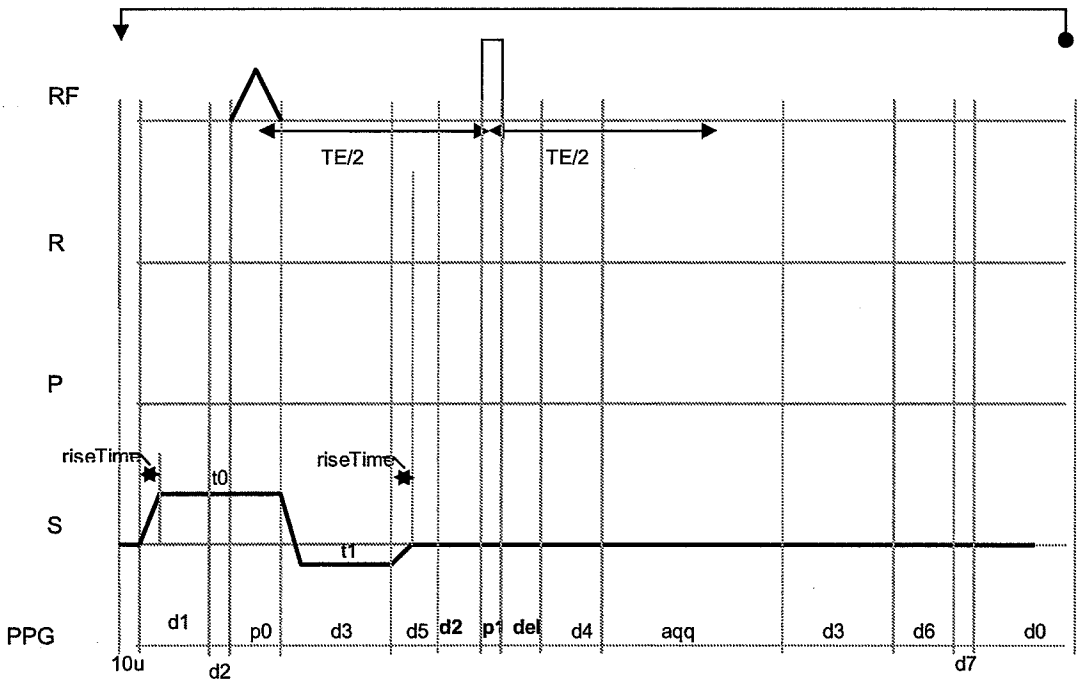
$$\Delta Z = (FOV * \Delta f) / SW\_h \qquad (1)$$

If the slice thickness differs from the selected value, check either your method files or the bandwidth of your excitation pulse.

Note: after implementing these changes in the pulse program, you can no longer expect having right calculated values for the parameters that are derived from the "gre" method relations (TE, TR...).

3.3 Method features

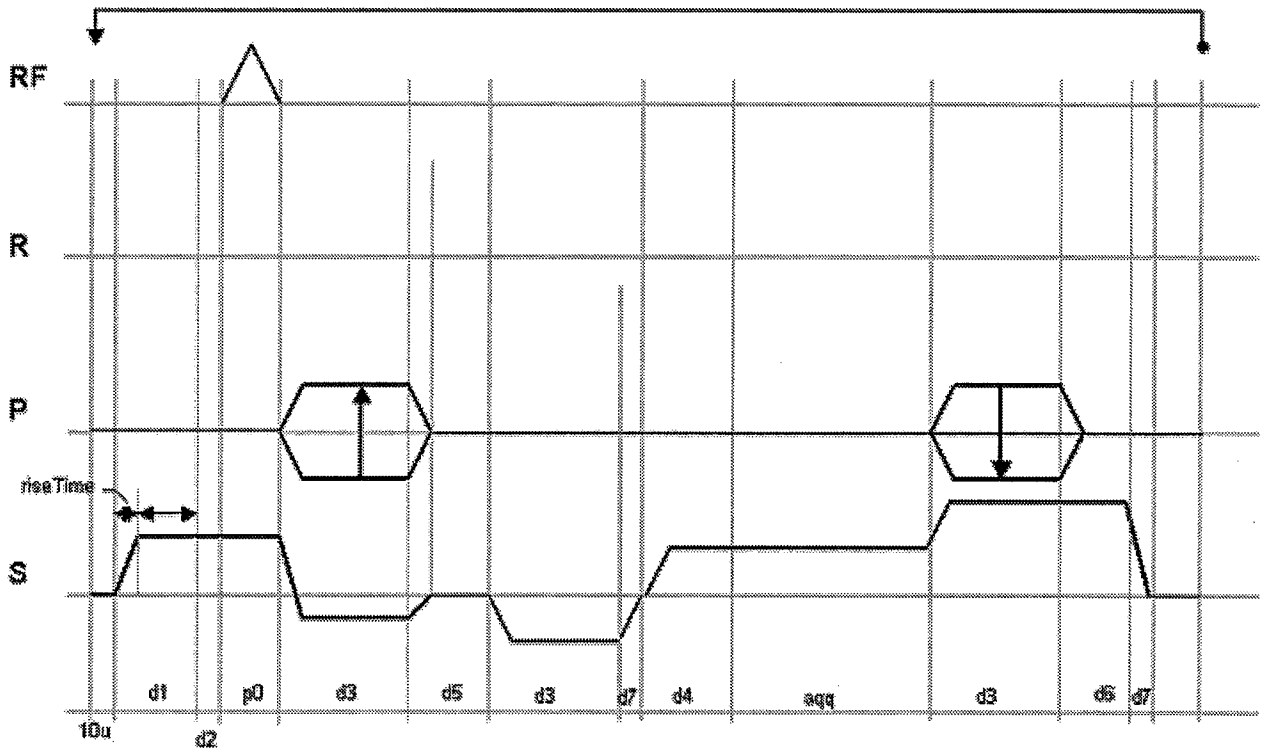
3.3.1 Project 2.1



Special features: modified 2D gradient echo sequence

- (π/2 selective) –(π non-selective) sequence
- no gradients applied in the readout and phase-encoding directions

3.3.2 Project 2.2



Special features: modified 2D gradient echo sequence

- slice selection and readout gradients in the same direction
- phase-encoding gradient

3.4 Realization hints

3.4.1 Project 2.1

1. Copy gre.ppg in ppcTest1.ppg and follow the next implementation steps to modify it. Don't forget to update the PULPROG variable accordingly.

2. Remove all the gradients in the readout and phase-encoding directions by setting the corresponding trim values to 0.

3. Define a new delay `del` in ppcTest1.ppg and insert it after the variable delay d5 :

   - `define delay del "del = p0/2+d3+d5+d2-denab-aqq/2"`

Note: make sure that d5 (≈TE) is big enough to allow a reliable `del` calculation.

4. Insert a block pulse "bp" just before the delay `del` (use the pulse duration p1, the shape pulse sp1, and the phase ph2). Define the ph2 frequency list at the end of your ppg :

   - `ph2 = 1`

Note: the new π pulse parameter value is independent on the "gre" method. Thus you will need to initialize its corresponding ACQP parameters. The Pulse Program Tool can be used to display your modified ppg only if the ACQP parameters have been initialized.

5. It is also necessary to insert a gate pulse command directly between the d5 duration and the block pulse "bp" (use the d2 delay).

6. You are now able to test your new sequence by keeping the predefined shims. Follow the following recommendations :

   - select only one slice and center it
   - TE = 50ms / SWh = 50000Hz, Matrix size = 1024x128
     (=> d5 =38ms / aqq=20.5ms)
   - In the ACQP class:
     - Set the PULPROG parameter according to your new ppg's name
     - ACQ_O1B_list = 3000Hz (setting the receiver frequency offset off-resonance will make the echo monitoring easier)
     - P[1] = 200us
     - TPQQ [1]
       .name = "bp"
       .power = to calculate according the formula below (2)
       .offset = 0Hz

Note: use the following formula to derive the attenuation value (in dB) for the block-pulse 180° / 200us:

TPQQ[1].power = PVM_RefAtt – 20 log(1000us/200us) – 6 [dB] (2)

with PVM_RefAtt : reference attenuation calculated for a 90°/1ms block-pulse shape

7. Run your sequence in GSP, and observe the echo position by changing the shim values in the Z direction (use the shim tool for that purpose). Furthermore, don't forget to reset your changes before closing the shim tool. If your slice selection gradient is well calculated, no shift of the echo position is expected.

8. If you change the slice rephasing gradient trim value by about 5% (use the ACQ_trim[1] parameter), you can now simulate a bad trim calculation. In such a case, expect a shift of the echo position is to expect.

3.4.2 Project 2.2

1. Copy gre.ppg in ppcTest2.ppg and follow the next implementation steps to modify it. Don't forget to update the PULPROG ACQP variable accordingly.

2. Move the dephasing lobe of the readout gradients (t2 trim) after the delay d5.
   You will need to add a new d7 delay for the ramp time down of the dephasing gradient lobe.

3. After updating the PULPROG variable according to ppcTest2.ppg, visualize the pulse program with the Pulse Program Tool. Then, by considering the pulse program diagram related to the project 2.2 change the values of the ACQ_grad_matrix parameter accordingly.
   Assuming the axial orientation as default spatial orientation, the required changes can be described as followed:

$$ACQ\_grad\_matrix_{init} = [[1\ 0\ 0]\ [0\ 1\ 0]\ [0\ 0\ 1]]$$

$$\Downarrow$$

$$ACQ\_grad\_matrix_{final} = [[0\ 0\ 1]\ [0\ 1\ 0]\ [0\ 0\ 1]]$$

   Check your pulse program again by re-opening the Pulse Program Tool. You will need to compile your ppg in X/Y/Z in order to really see what happened along the 3 main axis.

4. You can now measure the slice thickness. Use the following parameters:
   - sinc10H / 2ms
   - TR = 1000ms
   - Position your slice outside the inhomogeneous areas

   Export the scan into TopSpin and determine the half height of the phase profile after Fourier transformation. For that purpose, type into TopSpin the following commands:
   - `rser` for extracting a fid number from the 2D dataset : select the middle fid # 64,
   - `FT` for applying the Fourier transformation,
   - use the phasing icon (TopSpin panel) for phasing the spectrum: adjust the zero and first phase correction order by pressing the 0 and 1 buttons (panel at the top of the current spectrum window) to improve the phasing.

   By applying the formula (1), derive the value of the slice thickness and compare it to the expected value.

3.5 Proposed solutions

3.5.1 Project 2.1 : ppcTest1.ppg

```
;=============================================================
;   definition of delays
;=============================================================

define delay denab
"denab =  d4 - de + depa"
define delay del
"del =  p0/2 +d3 +d5 +d2 - denab -aqq/2"


;=============================================================
;   declaration of 2d and 3d loop
;=============================================================



#include <MEDSPEC.include>

;=============================================================================
;     D/P                spec control          gradients
;=============================================================================
start,10u                fq8b:f1
      d1                 fq1:f1                grad{( 0)| (0) |(t0)}
      d2                 gatepulse 1
     (p0:sp0 ph1):f1
      d3                                       grad{(0)|(0)|(t1)}
      d5                                       groff
      d2                 gatepulse 1
      (p1:sp1 ph2):f1
      del
      denab              REC_ENABLE
                         ADC_INIT_B(ph1, ph0)
      aqq                ADC_START
      d3
      d6
      d7
      d0                 ADC_END
goto start
exit
;=============================================================
;phase lists

ph0 = 0
ph1 = 0
ph2 = 1
;=============================================================
```

3.5.2 Project 2.2 : **ppcTest2.ppg**

```
#include<Avance.incl>
#include <DBX.include>
preset off

;=============================================================
;   definition of delays
;=============================================================

define delay denab
"denab =  d4 - de + depa"



;=============================================================
;   declaration of 2d and 3d loop
;=============================================================


lgrad r2d<2d> = L[0]
zgrad r2d

lgrad slice = NSLICES
zslice

#include <MEDSPEC.include>



;=============================================================================
;     D/P                spec control          gradients
;=============================================================================
start,      10u          fq8b:f1
      d1                 fq1:f1            grad{( 0)| (0) |(t0)}
      d2                 gatepulse 1
     (p0:sp0 ph1):f1
      d3                                   grad{(0)| r2d(t3)|(t1)}
      d5                                   groff
      d3                                   grad{(t2)| (0)|(0)}
      d7                                   groff
      denab              REC_ENABLE        grad{(t5)| (0) | (0)}
                         ADC_INIT_B(ph1, ph0)
      aqq                ADC_START
      d3                                   grad{(t8)|r2d(t6)| (0) }

      d6                                   grad{(t8)| (0) | (0) }
      d7                                   groff
      d0                 ADC_END
      1u                                   islice
lo to start times NSLICES
      1u            ipp1                   zslice
lo to start times NA
      1u                 rpp1              igrad r2d
lo to start times l0
      1u
lo to start times NR
```

```
goto start
exit
;===============================================================
;phase lists
ph0 = 0
ph1 = 0
;===============================================================
```

ParaVision Methods (PVM)

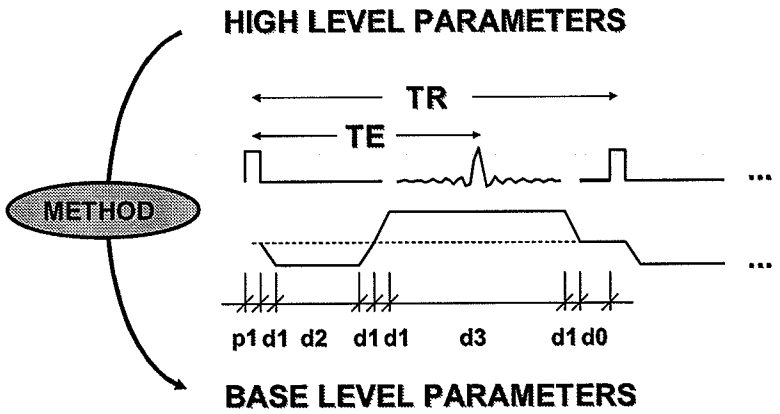
ParaVision Programming Course
April 3-7, 2006

Development in *ParaVision*

- Base-level programming
  - Pulse program
  - Acquisition parameters
- High-level programming (Methods)
  - Intuitive parameterization
  - Control of consistency
  - Graphic interface (geometry)

METHODS

- Experiment description



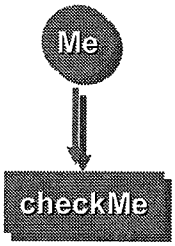
Advantages of PVM vs. IMND

- One binary file per method
- Separated source code
  - one directory per method
- Easy and safe installation
  - one file to be exported
  - distribution with/without sources
  - methods can be private for one user
- New features
  - Parameter redirections
  - Local parameters
  - Toolbox library
  - Methods have full control of base-level parameters

Basics of PVM Programming

- Parameter relations
- Method structure
- Tools for user develepment
  - Toolbox library
  - Modules

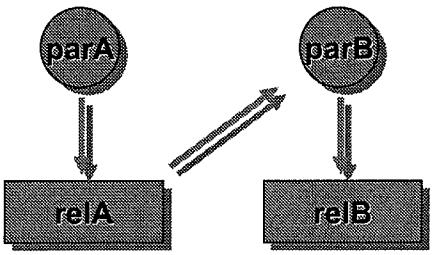
Parameter relations




```
int parameter
{
  display_name "I'm positive";
  relations checkMe;
} Me;
```

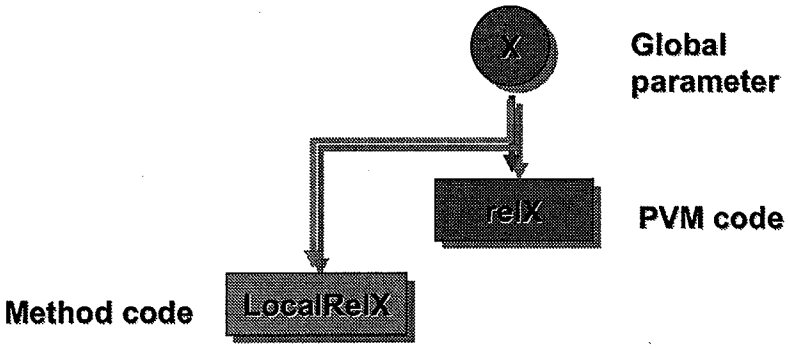
```
void checkMe(void)
{
  if(Me < 1)
    Me = 1;
}
```

Chain of relations



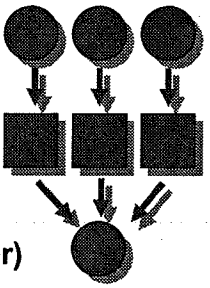

```
void relA(void)
{
  ParxRelsParRelations("parB",Yes);
}
```

Relation redirection




```
relations X LocalRelX;
```

Group redirection



Global parameter group

Default relations

Call-back parameter (handler)

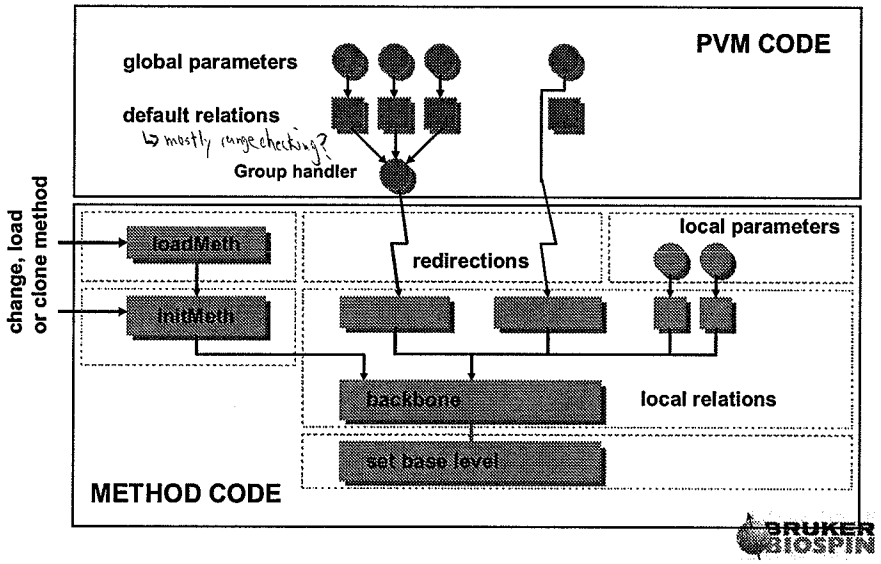
PVM code

Method code

```
relations handler LocalRels;
```

BRUKER BIOSPIN

METHOD STRUCTURE



BRUKER BIOSPIN

Method source code:

Selected files in the directory FLASH/

need not modify

Makefile	**compilation recipe**
FLASH.c	**glue file**
parsDefinitions.h	**parameter declarations**
parsLayout.h	**definition of MethodClass**
callbackDefs.h	**redirections**
parsRelations.c	**backbone & local relations**
initMeth.c	**function initMeth()**
FLASH.ppg	**pulse program**

Location of methods

SOURCES: Bruker:
$PvHome/prog/parx/src/
Local, typically:
$HOME/methods/src/

BINARIES: Bruker:
$PvHome/prog/parx/methods
Local, usable for all:
$PvHome/prog/parx/pub
Local private:
$ParxMethodSearchPath, typically:
$HOME/methods/bin/

Tools for user development

- Example methods with source code
  - FLASH, RARE, GEFC,
  - SINGLEPULSE, CSI,
  - FISP, EPI, FAIR_EPI
  - DtiStandard, DtiEpi (licensed)
  - gre - basis for this course
- Toolbox libraries
- Modules
- copyMethod script

PVM Toolbox

- Documented library
  - .../docu/english/pvman/pvm_docu/index.html
- Validity guaranteed for future versions (interfaces will never change)
- Organisation
  - ATB_ acquisition
  - STB_ sequence
  - PTB_ paravision
  - MRT_ MR utilities - eg. calc gradient strength for a particular resolution
  - UT_ general utilities
  - CFG_ configuration always better to rely on CFG fcns to maintain forward compatibility

Toolbox documentation

double MRT_MinSliceThickness(double sliceThickHz,
double sliceGradRatio,
double limitingSliceGrad,
double limitingRephaseGrad,
double gradCalConst)

Calculate the minimum slice thickness
There are no error conditions for this function.

Toolbox Classification:
FUNCTION

Toolbox Functions Called: MRT_MaxSliceGrad, MRT_SliceThickness.

Parameter Relations Called:
NONE

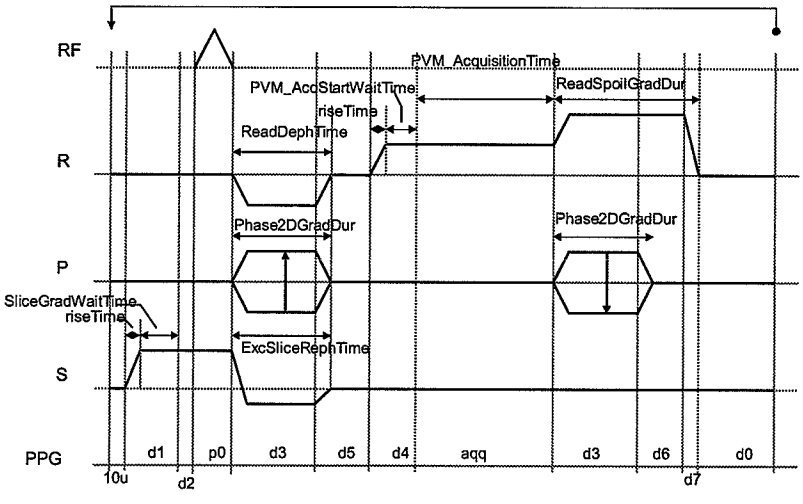
Parameters:
sliceThicknessHz - Thickness of slice (Hz).
sliceGradRatio - The ratio of the strengths of the slice ...

PVM Toolbox – examples

```
STB_InitRFPulse(&ExcPulse, "gauss", 1., 90.);

gconst = CFG_GradCalConst();

Acqtime = MRT_AcqTimeEffSWh(effSWh, size[0]);

ATB_SetNI(nslices*nechoes);
```

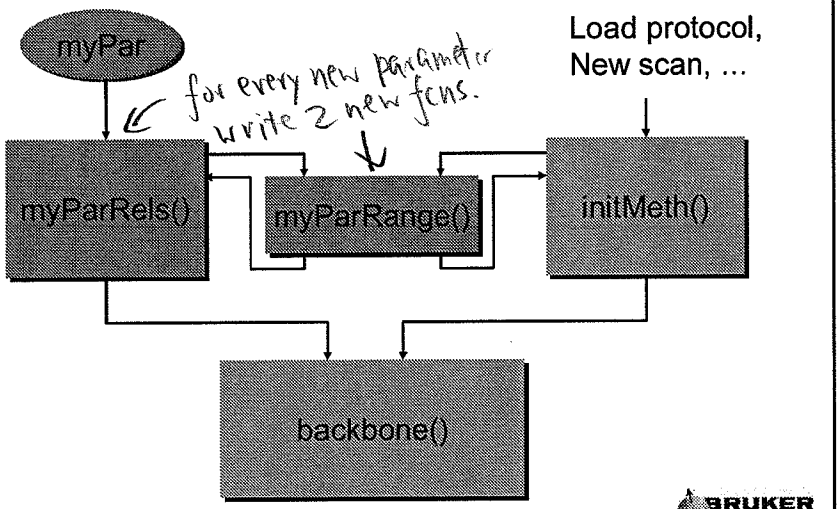
PVM programming: first steps

gre the template method



PARCOMP_DEBUG=3

Relations and range checking



Ralations and range checking, cont.

parsDefinitions.h

```
int parameter {
  relations myParRels;
} myPar;
```

parsRelations.c

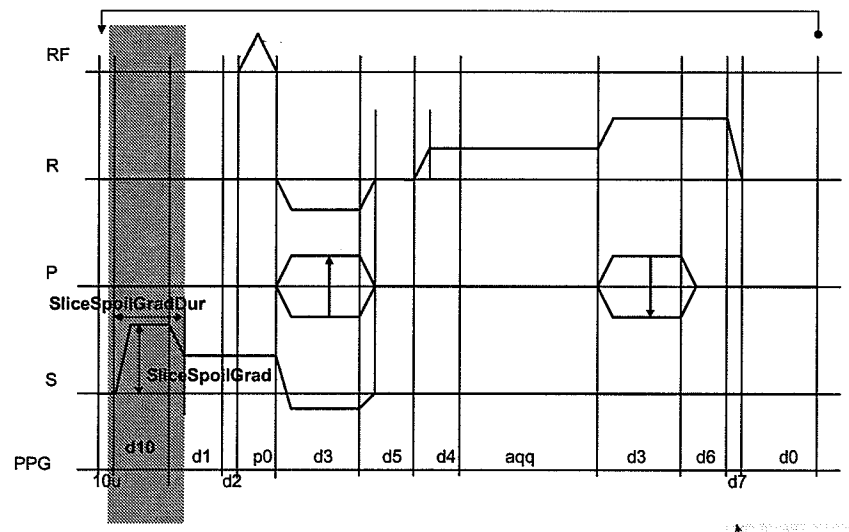
```
void myParRels() {
  myParRange();
  backbone();
}
void myParRange(){
  if(!ParxRelsParHasValue("myPar"))
    myPar = 5;
  else
    myPar = MAX_OF(myPar,0);
}
```

built in fcn.

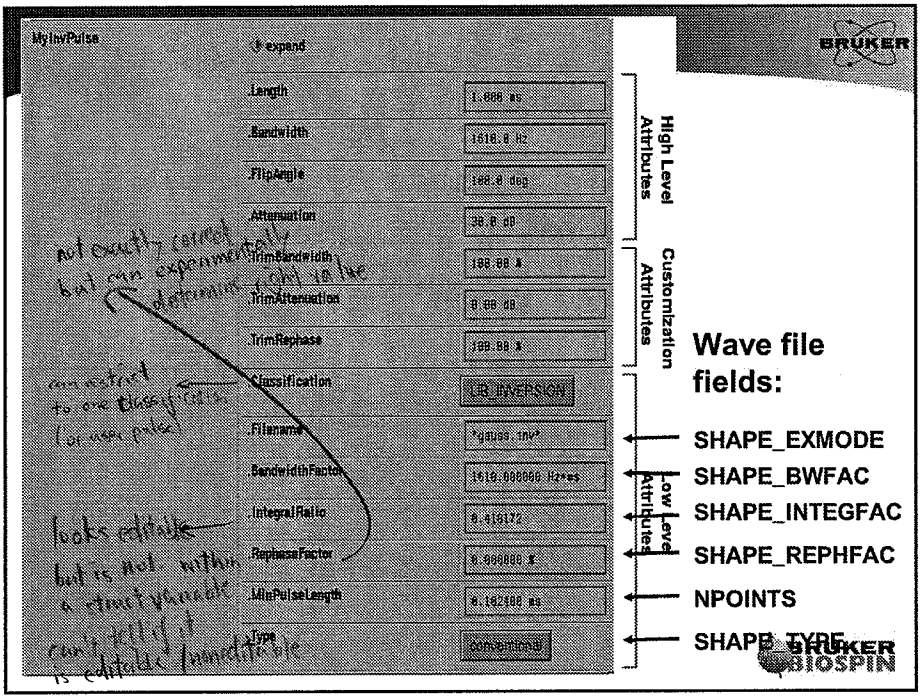
Initial value

Legal range

spGre: Introducing a spoiler



RF Pulses in PVM



Definition of an RF pulse

```
PVM_RF_PULSE_TYPE parameter
{
  display_name "Excitation Pulse";
  relations    ExcPulseRel;
}ExcPulse;
```

- **PVM_RF_PULSE_TYPE** is a structure representing physical properties of an RF pulse (length, bandwidth, etc.)
- You can define several pulses
- You can define pulse arrays

BRUKER BIOSPIN

RF-pulse parameter relations

```
void ExcPulseRel(void)
{
  UT_SetRequest("ExcPulse");
  ExcPulseRange();
  backbone();
}
```

Needed for a correct update (correct event handling on parameter edit)

```
void ExcPulseRange(void)
{
  individual restrictions (legal flip-angle, classification, etc.)
  ...
  STB_CheckRFPulse(&ExcPulse);
}
```

given as a pointer b/c the function needs to change its contents

General check

convenient range check - library fcn

Update of RF Pulses

To be called in backbone()

```
void updateRFPulses(char *nucleus)
{
  YesNo refAvailable;
  double refAtt=30;

  refAviable = CFG_GetGlobRefAtt(nucleus,&refAtt);

  STB_UpdateRFPulse("ExcPulse", &ExcPulse,
                    refAvailable, refAtt);
}
```

returns Y/N has ref attenuation been set yet?

convenient range check, contained in library

RF Pulses: transfer to base level

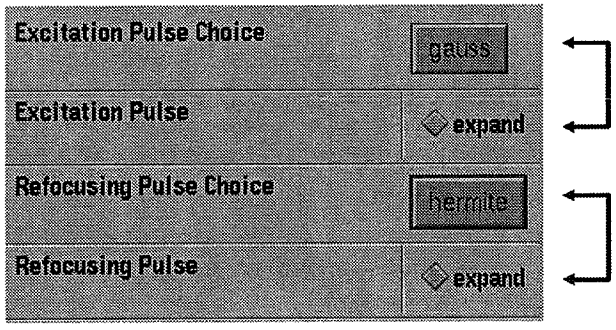
Called in BaseLevelRalations()

```
void setPpgParameters(void)
{
  ...
  TPQQ[0].power  = ExcPulse.Attenuation;
  TPQQ[0].offset = 0.0;
  ParxRelsParRelations("TPQQ",Yes);

  P[0] = ExcPulse.Length * 1000;
  ParxRelsParRelations("P",Yes);
  ...
}
```

Conversion ms → µs

RF Pulse "Enums"

- Easy access to available pulse shapes
- Automatic update from .../lists/wave



Enums are related to pulses

Using RF-Pulse Enums

definition

```
PV_PULSE_LIST parameter
{
  display_name "Excitation Pulse Choice";
  relations    ExcPulseEnumRel;
} ExcPulseEnum;
```

initialisation

```
STB_InitExcPulseEnum("ExcPulseEnum");
```

Exc, Rfc, Inv, depending on classification of the related pulse

Using RF-Pulse Enums, cont.

Connection PULSE → ENUM (after pulse update)

```
STB_UpdateExcPulseEnum(
                    "ExcPulseEnum",
                   &ExcPulseEnum,
                    ExcPulse.Filename,
                    ExcPulse.Classification);
```

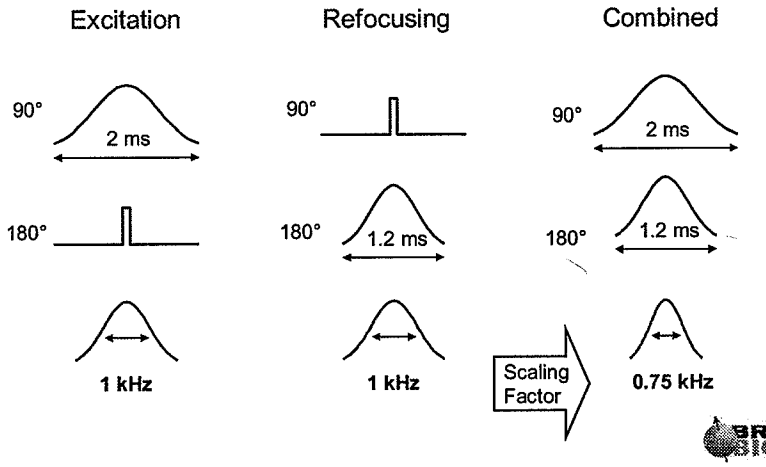
Connection ENUM → PULSE (in enum's rels)

```
STB_UpdateExcPulseName("ExcPulseEnum",
                   &ExcPulseEnum,
                    ExcPulse.Filename,
                   &ExcPulse.Classification);
```

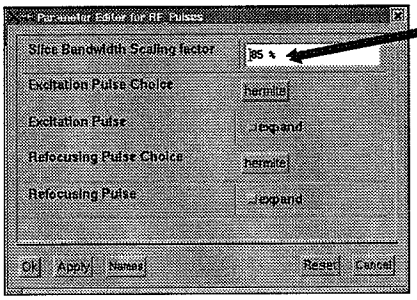
Exc, Rfc, Inv

Bandwidth Scaling in Spin Echo

An echo of the 1kHz-90 — 1kHz-180 sequence is not 1kHz broad.



Bandwidth Scaling in SE, cont.



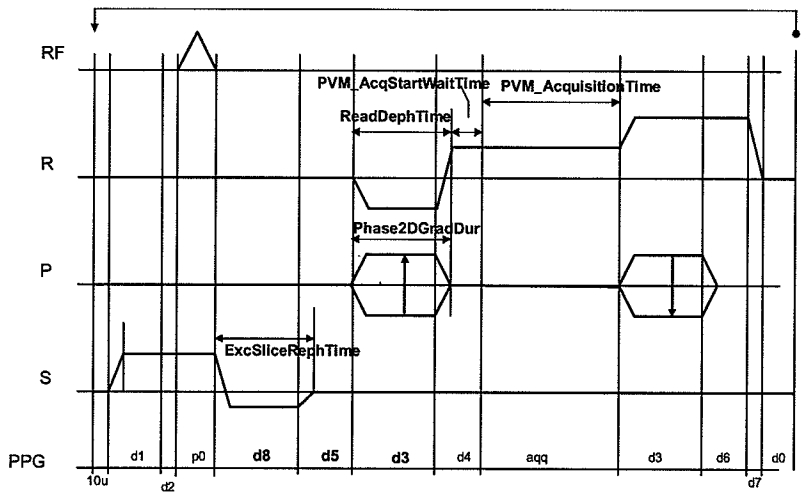
Scaling factor depends on shape combination:

Gauss – Gauss:	75.8%
Hermite – Hermite:	85.8%
Sinc3 – Sinc3:	85.5%

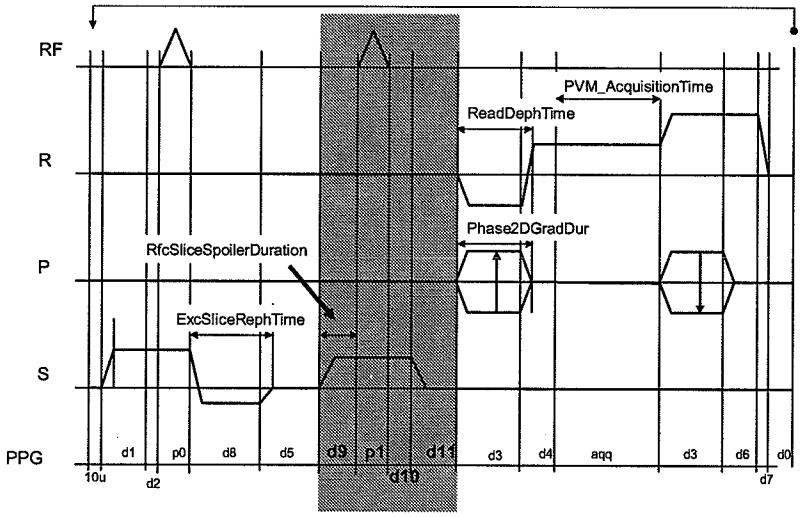
Other combinations not measured.

See Advanced Users Manual, chapter 5.

ItGre: Timing modification



spinEcho: Introducing an RF pulse



Modules in PVM

Modular design

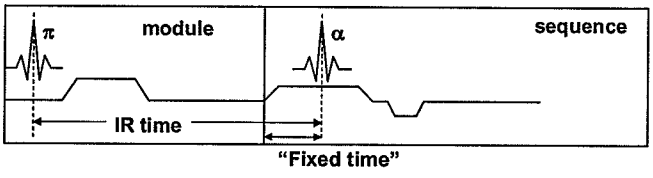
- Find elements which often repeat
- Make them building blocks for the user
- Focus:
  - Interface for the user
  - Interface to the rest of the sequence
- Examples:
  - Contrast preparation (fat sup, IR, MT)
  - Diffusion preparation
  - EPI readout

PV 4.0: slice selective inversion

Module components

- Piece of pulse program
- Group of parameters
- Library functions:
  - Initialisation
  - Update
  - Calculation of base-level parameters
- Interface to the main code
  - Global parameters (e.g. module duration)
  - Update arguments (e.g. nucleus)

Usage example: Inversion-Recovery



"Fixed time"

PULSE PROGRAM

```
start,  d0
        #include<inversion.mod>
        Excitation
        Acquisition
        lo to start times Ny
```

Method code

Update fixedTime

Update Inversion group:

```
STB_UpdateInversion(fixedTime);
```

Update repetition time:

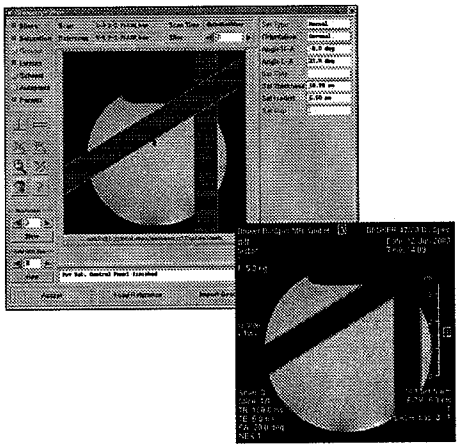
```
minTR = PVM_InvModuleTime + …;
TR = MAX_OF(TR, minTR);
```

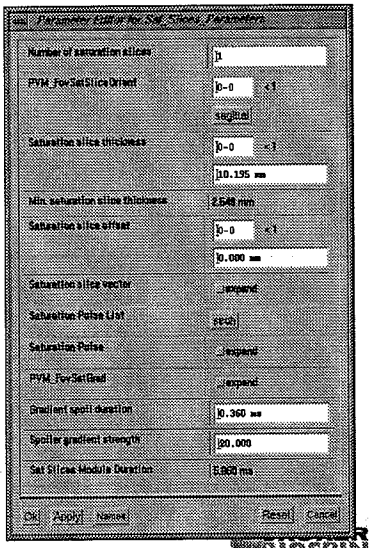
Set base-level parameters:

```
ATB_SetInversionBaseLevel();
```

don't turn on too many modules at once!

Saturation Slices Module

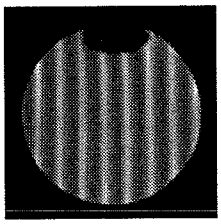




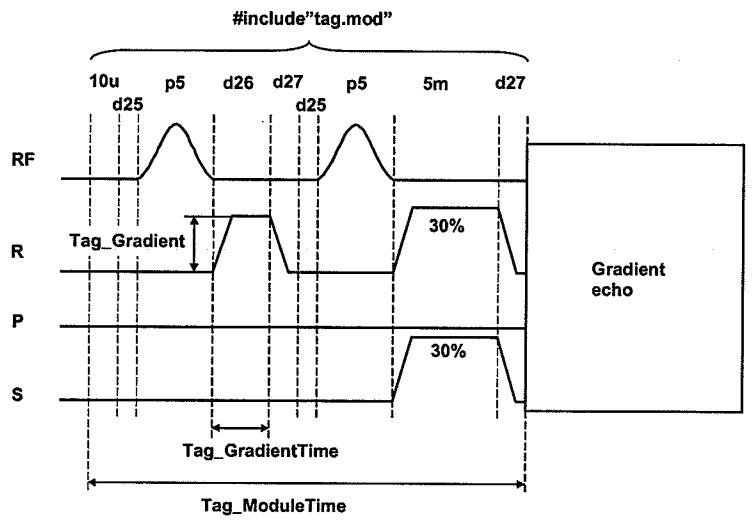
greTag: Creating a tagging module

New parameter group:
Tagging

Contains
Tag_Spacing
Tag_Gradient
Tag_GradientTime
Tag_RFPulse
Tag_ModuleTime

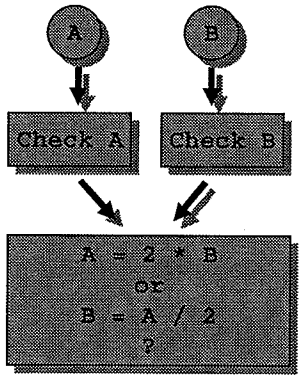


greTag, cont.



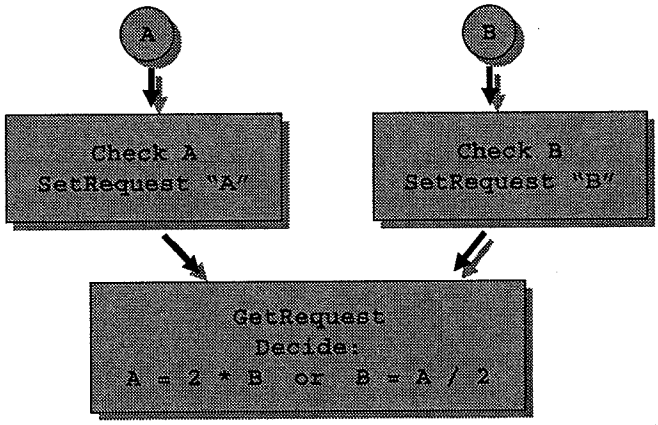
Request Handling

- Problems for group updating:
  - Who was changed?
  - Which structure field?
  - Which array element?



Request Handling, cont.

- Using UT_Get/SetRequest()



Request Handling, cont.

```
void myParRels(void)
{
  UT_SetRequest("myPar");
  myParRange();
  backbone();
}
```

```
void updateMyGroup(void)
{
  const char *const parlist = "myPar,anotherPar";
  const char *const structlist = "amplitude,duration";
  int requestDetected, parNo, structNo;

  requestDetected = UT_GetRequest(
                    parlist,&parNo,
                    structlist,&structNo);
...
}
```

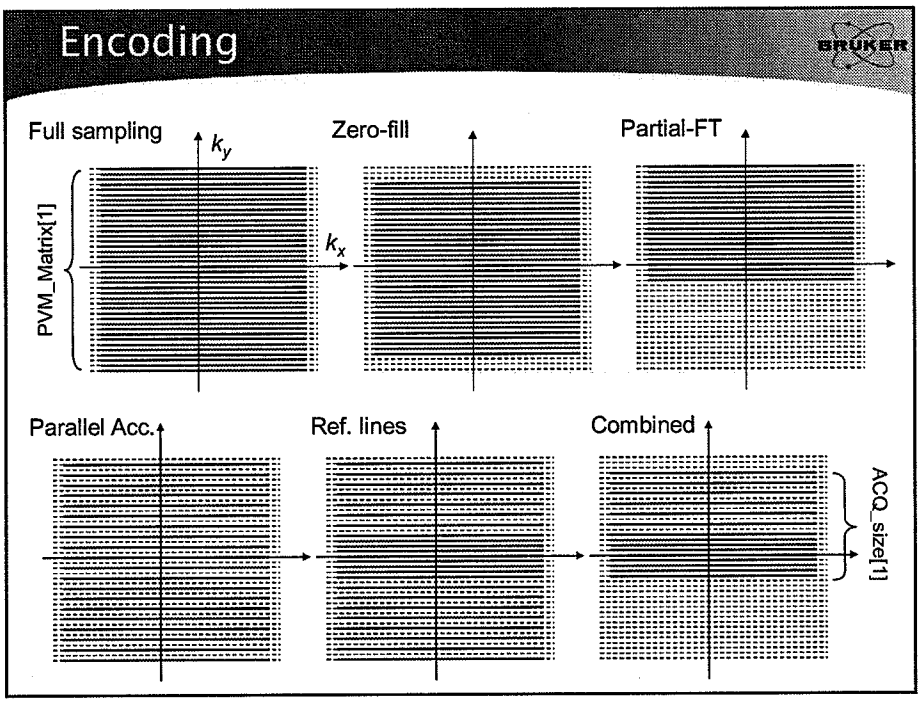
Project: Accelerated Imaging (encGre)

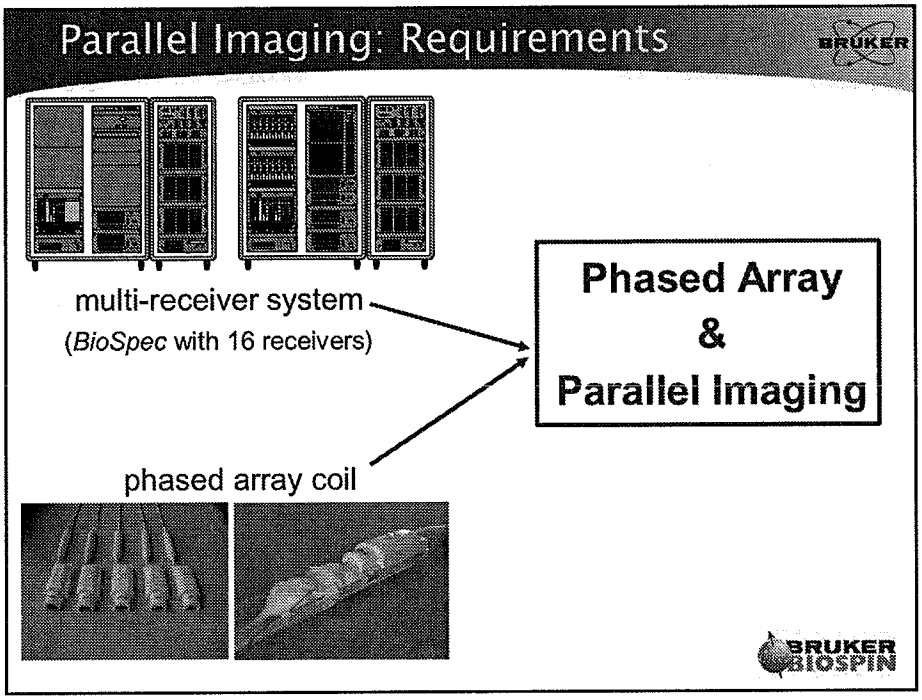
ParaVision Programming Course
April 3-7, 2006
Sascha Köhler

New parameter group

Encoding:

- Parallel Imaging
- Linear and centric k-space sampling
- Segmentation
- Partial Fourier encoding
- Zero filling


Encoding
Full sampling
k_y
Zero-fill
Partial-FT
PVM_Matrix[1]
k_x
Parallel Acc.
Ref. lines
Combined
ACQ_size[1]

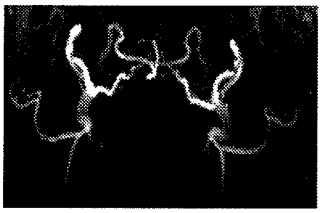

Parallel Imaging: Requirements
multi-receiver system
(*BioSpec* with 16 receivers)
phased array coil
Phased Array
&
Parallel Imaging
BRUKER BIOSPIN

Parallel Imaging: Advantages

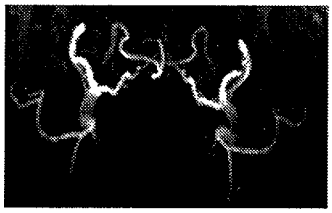
- Reduce scan time, but keep image quality

Time-of-flight angiography of rat brain:
matrix size = 256 x 256 x 128, resolution: 117 x 148 x 250 µm

10 min

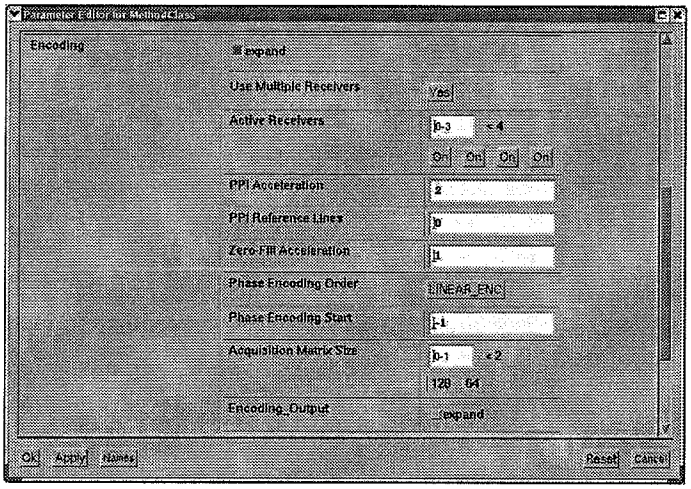


5 min, Acceleration = 2



identical image quality

Encoding group



Your Task

Implementation of new parameter group *Encoding* into PVM method *gre*.

Steps:

1) Use copyMethod to copy *gre* to *encGre*.
2) Include *Encoding* in the file **parsLayout.h**.
3) In **callbackDefs.h** the group handler must be directed to the method's *backbone* function:
   - relations PVM_EncodingHandler backbone;

Your Task

Steps:

5) In the *backbone* function in **parsRelations.c** the *Encoding* group must be updated:

```
STB_UpdateEncoding(PTB_GetSpatDim(),      /* dimension */
                   PVM_Matrix,            /* image size */
                   PVM_AntiAlias,         /* a-alias */
                   &seg_size,             /* segment size */
                   SEG_SEQUENTIAL,    /* segmenting mode */
                   Yes,                   /* PI in 2nd dim
allowed */
                   Yes,             /* PI ref lines in 2nd dim
allowed */
                   No);           /* partial ft in 2nd dim NOT
allowed */
```

Your Task

Steps:

6) Use *PVM_EncMatrix* for experiment time calculation in **parsRelations.c.**

(PVM_EncMatrix describes the acquisition matrix and is defined as product of image size and anti-aliasing factor)

7) Use *PVM_EncMatrix* for setting *ACQ_size* in **BaseLevelRelations.c.**

8) Set ACQP parameters for *User Encoding* in **BaseLevelRelations.c.**

Your Task

Steps:

9) Setting base-level parameters for controlling a multiple receiver acquisition:
   - toolbox function *ATB_SetMultiRec()*
     - Yes
       - special pulse program required (only for older versions of spectrometer hardware)
     - No (for Avance II)

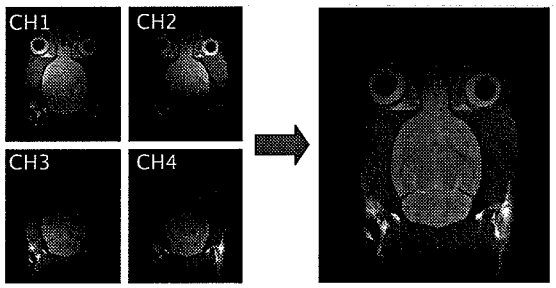
```
If(Yes == ATB_SetMultiRec())

ATB_SetPulprog("encGre.4ch");
```

Your Task

Steps:

10)Multi receiver data require a special reconstruction mode which is defined in the **RecoRelations.c**:

- each array element receiving separately copy *RecoRelations.c* from the master solution



Additional Exercise

Segmented acquisition

- Use *PVM_RareFactor* for updating the Encoding group

The purpose of this exercise is to demonstrate the functionality of the new introduced Encoding group:

- only a few steps are necessary to realize segmented acquisition

Method Programming in PVM
Part 1

ParaVision Programming Course
April 3-7, 2006

Authors:

Franciszek Hennel

Sascha Kőhler

Markus Wick

Bruker BioSpin MRI GmbH

Table of Contents

Introduction

As you know from the previous course, the acquisition and reconstruction are controlled by a pulse program and by numerous "base-level" parameters grouped mainly in the ACQP and RECO classes. In principle, it is possible to set up an experiment by editing the pulse program setting these parameters directly. On the other hand, one usually prefers to specify the measurement in terms of "high level" parameters which may not have direct representation in ACQP or RECO. For example, it is more convenient to specify the echo time than to set delays used in the pulse program. The mechanism linking the high-level requests with the base-level parameters is provided by the **ParaVision Methods (PVM)**. When you type in the scan editor: "Echo time = 50 ms", it is the currently active method which checks if this value is allowed and calculates the corresponding delays. Similarly, when you move a slider controlling the field of view in the geometry editor, the active method takes care of not letting you go below the minimum and calculates the necessary gradient amplitudes. In general, the **role of a method** is to:

- provide high-level parameters for experiment description,
- give the parameters initial values in a new scan,
- decide which parameters should be visible, editable and saved in protocol files,
- react to loading a protocol,
- react to changes of parameters made by the user in the geometry editor, scan editor, method editor, or by a shell command,
- check if the new parameter values are legal, and if all parameters agree with each other,
- derive base-level parameters from high-level parameters.

During this course you will learn programming PVM methods by examples. We will start by an overview of an existing **template method *gre*** and by preparing your working environment for PVM programming. The first programming example will be a simple copy of *gre* just to see how methods are compiled and installed.

In the next project (*spGre*) a user-defined spoiling gradient pulse will be introduced to the *gre* sequence. This example will show you how to add **new parameters** to a method, control their legal range, and program dependencies with other parameters (relations).

Another project (*spinEcho*) will you teach you the handling of **radio frequency pulses** in PVM methods. A special structure type which describes RF pulse properties will be introduced together with toolbox functions for controlling such structures. You will understand the mechanism which derives a necessary attenuation for a given pulse flip angle and shape, pulse bandwidth for a given duration and vice vera, etc. As you will see, all this is achieved by calling a proper updating function for a user defined RF pulse parameter. Additionally, you will also learn using pulse enums, auxiliary parameters allowing an easy selection of a valid pulse shape.

In the same project (*spinEcho*) you will also see how to use **array parameters**, change their sizes and set their elements. This will be shown on the basis of a local frequency list for the refocusing RF pulse.

Each **exercise project** is described step by step in its "**Realization hints**" section. These hints are sufficient (we believe) to complete the projects but still remain open to your invention. Some projects contain suggestions for further extensions which you can follow if the time allows. Should you need more detailed guidance, each project description also contains elements of source-code listing which can simply be copied to your files. Finally, if you prefer analyzing a **ready solution** rather than programming yourself, all projects have been pre-programmed and you can find their complete code in the /opt/PPC/PVM directory.

1 First Steps - A copy of the gre method (project myGre)

1.1 Objectives

This projects explains:

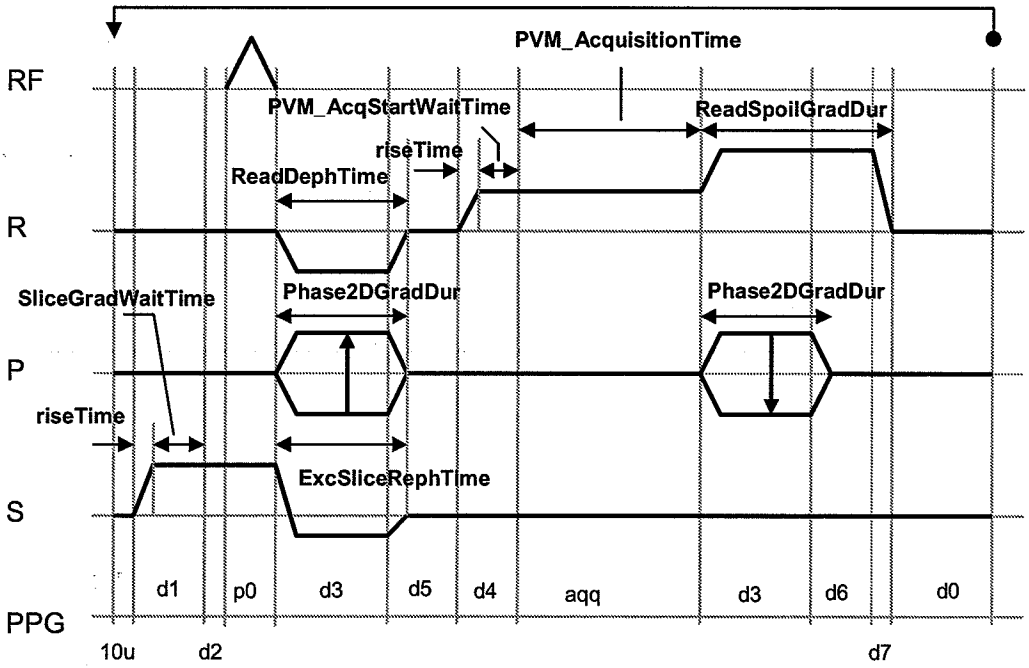
- How to prepare your login settings for PVM programming
- How to use the *copyMethod* script to produce a new method based on an existing example
- How to compile a new method and make it active in ParaVision
- How the source code of the template PVM method *gre* is organized.

1.2 Project description

This project is our first step in PVM programming. We will prepare the method programming environment in your home directory and produce a simple copy of the method *gre*. This is a simplified gradient echo method provided in ParaVision 3.0.2 as a programming template. It will be used throughout this course as a starting point for your projects.

We will have a look at the organization of the code of the new method (*myGre*) and discuss the purpose of all files. Debugging messages will be activated and we will be able to follow the function calling chain resulting from a parameter change, from a new scan selection, or loading a protocol. This exercise will help you understand the role of methods in ParaVision.

Finally, we will purposely produce a few typical compilation errors and runtime errors in the source code so that you can recognize and avoid them easily in future.



1.3 Specific method parameters

The method contains several parameters defining the sequence timing, such as the duration of phase encoding or readout dephasing. They are defined in the diagram above.

1.4 Code organization in gre

The source code is distributed in several files:

Makefile	The control of the compilation
parsDefinitions.h	Definitions of main-level parameters. Other definition files are included in this file by #include
parsLayout.h	Definition of the MethodClass. Other definition files are included in this file by #include
parsRelations.c	Relations of parameters from the main level (e.g. PVM_EchoTime) and the backbone function. The backbone is called from all parameter relations and is responsible for inter-dependence of parameters. At its end the backbone calls SetBaseLevelParm() and SetRecoParams().
BaseLevelRelations.c	Contains SetBaseLevelParm() which sets all ACQP parameters based on PVM parameter values established by the backbone.

FreqEncPars.h, FreqEncLayout.h, FreqEncPars.c

Code responsible for the group of parameters describing the frequency encoding. The h-files contain parameter definitions and the definition of the FrequencyEncoding parclass; they are included in the parsDefinitions.h and parsLayout, respectively. The c-file contains

- range checkers of all parclass memebers (e.g. ReadDephTimeRange() for ReadDephTime)
- relations for all parclass members (e.g. ReadDephTimeRels())
- InitFreqEncoding() – a function giving all group members default values, used in initMeth()
- FreqEncodingLimits() – a function calculating the minimum FOV in the frequency encoding direction. Used in backbone before the update of geometry.
- FreqEncodingGradients() – a function calculating all gradient amplitudes used on the readout channel. Used in the backbone after the update of geometry

SliceSelPars.h, SliceSelLaout.h, SliceSel.c

Files which manage the slice selection channel in a similar way as the ones for the Frequency Encoding.

Phase2DPars.h, Phase2DLayout.h, Phase2D.c,

Phase2DPars.h, Phase2DLayout.h, Phase2D.c

RFPulsePars.h, RFPulseLayout.h, RFPulse.c

Simmilar for the 2D and 3D phase encoding gradients and forRF pulses.

1.5 Your modifications

1.5.1 Preparing the programming environment:

You need to create a directory for the method source code sub-directories and another one for the binary method files. The typical locations are:

```
$HOME/methods/src
$HOME/methods/bin
```

Create these directories.

Now you have to tell ParaVision where to search for method binary files by defining the environment variable *ParxMethodSearchPath*. Edit the file *.bashrc* in your home directory and add the following line:

```
export ParxMethodSearchPath=$HOME/methods/bin
```

Log out and in again to make this definition active.

1.5.2 Using copyMethod

Use copyMethod to produce a copy of *gre*, called *myGre* in your source directory. Type

```
/opt/PV4.0/prog/bin/copyMethod
```

and follow the dialog. The script will automatically start the compilation of the new method. Restart ParaVision and check if the new method is available by selecting it in a new scan. Try to display its pulse program (do not forget to copy it in the pp directory). Acquire some images with *myGre* and get familiar with the meaning of its parameters.

1.5.3 Activating debug messages

You will notice that each function in the *myGre* source code includes a pair of **DB_MSG** macro calls:

```
void backbone(void)
{
  DB_MSG(("---> backbone"))
  ...
  DB_MSG(("<--- backbone"))
}
```

These are the so called debugging messages used to show the function calling chain during the execution of parameter relations. The DB_MSG macro behaves exactly like the printf function – it can be given a format string followed by a list of numeric arguments, or just a simple text. The output goes to the startup shell window of ParaVision. The only difference is the necessary double pair of brackets.

To activate the debug macros you need to

- define the macro DEBUG to 1 in each c file in which the messages should be active
- define the macros DB_MODULE and DB_LINE_NR to one if you want the debug messages to be acompanied by file names and line numbers, respectively.

A typical definition is

```
#define DEBUG       1
#define DB_MODULE   1
#define DB_LINE_NR  0
```

Your task: ~~Remove the DNDEBUG option from the Makefile~~, activate the debug messages in initMeth.c, parsRelations.c and frecEnc.c, and de-activate them in BaseLevelRelations.c. Recompile the method by typing:

```
make clean cproto depend
make install
```

Restart Paravision and produce the following situations:

- produce a new empty scan and select the *myGre* method in Method editor
- modify the value of *Read Spoiler Duration*
- select a different scan, and go back to the one containing *myGre*

Each time, note what debug messages are printed in the startup window. Reconstitute the function calling chain. Check if it corresponds to the graph of a recommended method structure shown in Advanced Manual, Method Programming.

1.5.4 Typical compilation errors

Syntax error

Remove a colon at the end of a code line (anywhere) and recompile by *make install*. Find the error by reading the compiler messages. If you are using the emacs editor, you can browse

through the compilation errors with the F6 key and be directed to the corresponding code positions automatically.

Missing prototype warning

Open parsRelations.c and at the end of the file write the following funtion

```
void dummy(void)` {}
```

Call this function anywhere in the backbone. Compile by typing

```
make install
```

A warning message will appear that the function *dummy* was called without a prototype. In C every function must be decalared (by its prototype) before it is used. However, in method programming you do not need to care about prototypes. The only thing you need each time you define a function (or change a function's interface) is to recompile by typing

```
make clean cproto depend
```

followed by

```
make install
```

Now your modified myGgre will compile OK. Next time, when your code changes do not concern function definitions, it is enough to compile with the second command.

1.5.5 A typical runtime error

Open the initMeth.c file and find the line

```
ParxRelsMakeNonEditable("Phase3DGradDur,ReadDephTime");
```

Change slightly the name of the first parameter and recompile. The compiler will accept the change because the new name is still a valid string. However, when you restart ParaVision, it will be impossible to edit the myGre method (all parameters will disappear except for Method). The reason will be explained in the startup window: the parameter handling mechanism will not be able to change the attributes of a non-existing parameter.

In conclusion, whenever your method or method parameters disappear from the screen, this is due to an error in the method which has not been detected by the compiler. Look at the startup shell messages to figure out the exact reason.

Finally, remember to remove all introduced errors! You will need a working *myGre* for next projects. To be quite sure, remove *myGre* and generate it with *copyMethod* again.

2 Method Parameters: Adding a spoiler to gre (project spGre)

2.1 Objectives:

By developing this project you will learn how to

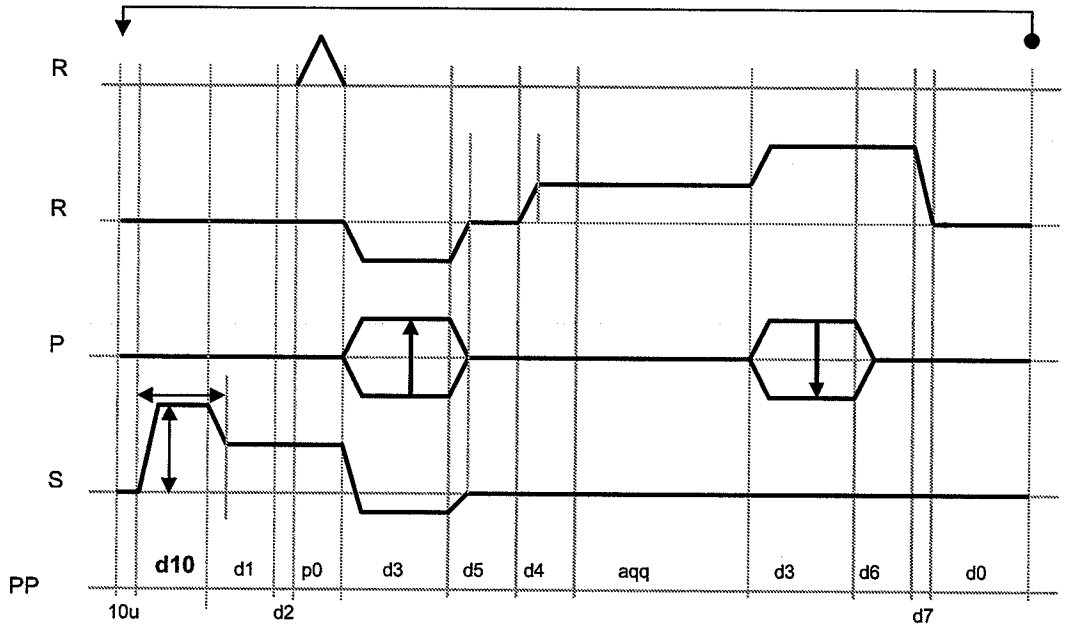
- define new parameters
- program a variable gradient pulse

2.2 Project description

We will start from the single gradient echo method (gre) and modify it to add a variable gradient spoiler in the slice gradient direction before the excitation RF pulse.

The most important modification consists of creating of two new parameters to control the amplitude and the duration of the spoiler. The repetition time of the sequence should be modified and a new gradient trim point will be created to define the spoiler amplitude.

Method "spGre"



2.3 Specific method parameters

SliceSpoilGrad – spoiler amplitude in percent of the maximum gradient.

SliceSpoilGradDur – duration of the gradient spoiler in ms.

2.4 Realisation hints

2.4.1 Starting point

The method should be developed based on the *gre* method (discussed in the first project).

2.4.2 Steps

1. Use copyMethod to copy *gre* to *spGre*.
2. Define the new double parameters *SliceSpoilGrad* and *SliceSpoilGradDur* (in SliceSelPars.h). Declare their relations as *SliceSpoilGradRel* and *SliceSpoilGradDurRel*.
3. Add the the two parameters in the *SliceSelection* parameter class (in SliceSelLayout.h).

4. Write (in SliceSel.c) the *SliceSpoilGradRel* and the *SliceSpoilGradDurRel* relations. Each relation should consist of calling a range checking function for the respective prameter followed by a call of the backbone. Write these range-checking functions (*SliceSpoilGraRangel* and the *SliceSpoilGradDurRange*) . They should give the respective parameters a default value if they have none, and constrain them. The minimum value of the gradient spoiler duration should be 2.0*CFG_GradientRiseTime().
5. Set the visibility of the new parameters in the *SliceSelectionParsVisibility* relation (in SliceSel.c) to make them appear in the protocol.
6. Call the range checking functions of the new parameters in the initialisation of the SliceSelection parameter group (*InitSliceSelection*, in SliceSel.c).
7. Add the spoiler duration to the minimum repetition time calculation (in the function *UpdateRepetitionTime* located in parsRelations.c).
8. Add the new trim value t9 for the SliceSpoilGrad parameter and the delay D[10]=(SliceSpoilGradDur – riseTime)/1000 in the function *SetBaseLevelParam* (in BaseLevelRelations.c).
9. Add the new line d10 grad{(0)|(0)|(t9)} in the spGre.ppg pulse program.

2.5 Proposed Solution

SliceSelPars.h define new parameters

```
double parameter
{
  display_name "Slice Spoiler";
  units "%";
  format "%f";
  relations SliceSpoilGradRel;
}SliceSpoilGrad;

double parameter
{
  display_name "Slice Spoiler Duration";
  units "ms";
  format "%f";
  relations SliceSpoilGradDurRel;
}SliceSpoilGradDur;
```

SliceSelLayout.h insert the new parameters in the SliceSelection class

```
parclass
{
  SliceGradStabTime;
  ExcSliceRephTime;
  ExcSliceGrad;
  ExcSliceGradLim;
  ExcSliceRephGrad;
  ExcSliceRephGradLim;
  SliceSpoilGrad;
  SliceSpoilGradDur;
```

```
}SliceSelection;
```

SliceSel.c parameter relations

```
/*---------------------------------------------------
 *  Default relations and range checker for the spoiler
 *---------------------------------------------------*/

void SliceSpoilGradRange()
{
  DB_MSG(("-->SliceSpoilGradRange"));

  if(!ParxRelsParHasValue("SliceSpoilGrad"))
  {
    SliceSpoilGrad = 40.0;
  }
  else
  {
    SliceSpoilGrad =
      MIN_OF(SliceSpoilGrad,100.0);
    SliceSpoilGrad =
      MAX_OF(SliceSpoilGrad,0.0);
  }
  DB_MSG(("<--SliceSpoilGradRange"));
}

void SliceSpoilGradRel(void)
{
  DB_MSG(("-->SliceSpoilGradRel"));

  SliceSpoilGradRange();
  backbone();

  DB_MSG(("<--SliceSpoilGradRel"));
}

void SliceSpoilGradDurRange()
{
  double min;

  DB_MSG(("-->SliceSpoilGradDurRange"));
  min = 2.0*CFG_GradientRiseTime();

  if(!ParxRelsParHasValue("SliceSpoilGradDur"))
  {
    SliceSpoilGradDur = MAX_OF(min,1.0);
  }
  else
  {
    SliceSpoilGradDur =
      MAX_OF(SliceSpoilGradDur,min);
  }
  DB_MSG(("<--SliceSpoilGradDurRange"));
}

void SliceSpoilGradDurRel(void)
{
  DB_MSG(("-->SliceSpoilGradDurRel"));

  SliceSpoilGradDurRange();
  backbone();
```

```
  DB_MSG(("<--SliceSpoilGradDurRel"));
}
```

SliceSel.c itialization of the parameters

```
void InitSliceSelection(YesNo showAllPars)
{
  DB_MSG(("-->InitSliceSelection"));

  SliceGradStabTimeRange();
  ExcSliceRephTimeRange();
  ExcSliceGradRange();
  ExcSliceGradLimRange();
  ExcSliceRephGradRange();
  ExcSliceRephGradLimRange();
  SliceSpoilGradRange();
  SliceSpoilGradDurRange();

  SliceSelectionParsVisibility(showAllPars);

  DB_MSG(("<--InitSliceSelection"));
}
```

parsRelations.c include SliceSpoilGradDur in the minimum repetition time

```
  PVM_MinRepetitionTime =
    nslices *
    (
      0.011                                                    +
      SliceGradStabTime                                        +
      SliceSpoilGradDur                                        +
      CFG_AmplifierEnable()                                    +
      ExcPulse.Length/2                                        +
      PVM_EchoTime                                             +
      PVM_AcquisitionTime *(1.0 - PVM_EchoPosition/100)        +
      ReadSpoilGradDur                                         +
      CFG_InterGradientWaitTime()
    );
```

BaseLevelRelations.c add new trim t9 and delay D[10]

```
ATB_SetAcqTrims( 10,
          ExcSliceGrad,                          /* t0 */
               -ExcSliceRephGrad,                /* t1 */
               -ReadDephGrad,                    /* t2 */
                Phase2DGrad,                     /* t3 */
                -Phase3DGrad,                    /* t4 */
                ReadGrad,                        /* t5 */
               -Phase2DGrad,                     /* t6 */
                Phase3DGrad,                     /* t7 */
                ReadSpoilGrad,                   /* t8 */
                SliceSpoilGrad                       /* t9 */
                );

(...)

D[10] = (SliceSpoilGradDur - riseT)/1000.0;
```

SpGre.ppg add the line with d10

```
start,      10u          fq8b:f1
            d10                                  grad{( 0)| (0) |(t9)}
            d1           fq1:f1                  grad{( 0)| (0) |(t0)}
            d2           gatepulse 1
            (p0:sp0 ph1):f1
            ...
```

3 Timing Control: Long echo-time version of gre (project ltGre)

3.1 Objectives

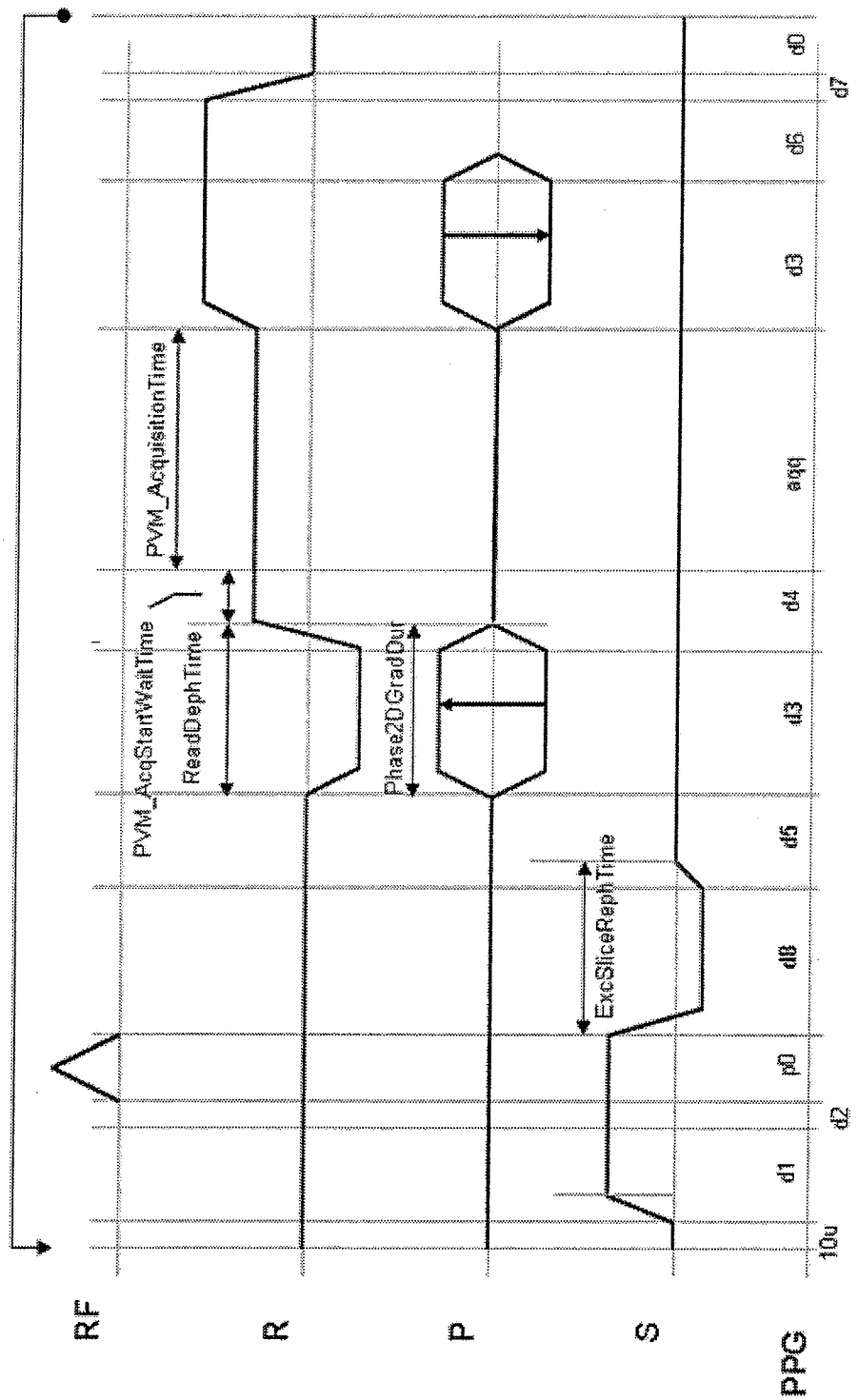
By developing this project you will learn how to

- modify the timing of the pulseprogram
- control the TE and TR calculation of the method
- control the maximum gradient settings dependent on the pulse program timing
- test the correct method timing by dedicated subroutines

3.2 Project description

We will start from the known single gradient echo method (*gre*) and modify it to get a different timing of the measurement sequence. We will separate the slice gradient rephasing from the readout dephasing and phase-encoding (in *gre* these three gradient pulses are simultaneous) and generate a TE filling delay at a different position. In the new sequence the readout dephasing will be done immediately before the read gradient is switched on. This will reduce the flow artifacts at long echo times. The necessary modification of the pulse program consists of a separate gradient switching command for the slice refocusing gradient and the introduction of a new delay for a separate control the duration of the slice refocusing.

The additional delay introduced for slice refocusing must be taken into account in the calculation of the echo time and the repetition time in the method code (TR and TE are calculated by subroutines called in the backbone). The slice refocusing time must be made independent of the 2D phase-encoding time. Additionally, since only two gradient channels are switched on simultaneously during phase-encoding (and not three as in gre) the maximum gradient amplitudes may be increased for the 2D phase-encoding gradient. This can be made depending on the dimensionality of the experiment (for 3D acquisition the 3D phase-encoding gradient is switched simultaneously).



Method "ItGre"

3.3 Specific method parameters

ExcSliceRephTime – the duration of the slice refocusing pulse

Phase2DGradLim, Phase3DGradLim, ReadDephGradLim, ExcSliceRephGradLim: Maximum gradient amplitude for the 2D, 3D read diphase and excitation slice rephase gradient respectively.

3.4 Special features

None in this project.

3.5 Realisation hints

3.5.1 Starting point

The method should be developed based on the *gre* method provided with ParaVision 3.0.2

3.5.2 Steps

10. Use copyMethod to copy *gre* to *ltGre*.
11. Modify the pulse program ltGre.ppg (created by copy method). In a first step, copy the line "`d3 grad{(t2)| r2d(t3)|(t1)+r2d(t4)}`" and paste it behind the "`d5 groff`" command. Remove the gradients which should not be switched at these two timepoints. Copy this pulse program to the pp directory and test it "manually".
12. Use a new delay (e.g. d8) for the slice rephasing gradient statement. The method ltGre should control this delay with parameter ExcSliceRephTime.
13. Check ExcSliceRephTime to appear editable in the method editor: The editabiliy of parameters are controlled by special functions for the different parameter groups. In case of the slice selection group the routine SliceSelectionParsVisibility (c module SliceSel.c) is used. Inspect the way the parameter editability is handled.
14. Find the place in the backbone routine (parsRelations.c) that connects the ExcSliceRephTime with the Phase2DgradDur and modify this line appropriately.
15. (Optional) Modify the setting of Phase2DGradLim, Phase3DGradLim, ReadDephGradLim, ExcSliceRephGradLim (initMeth.c parsRelations.c – backbone). The maximum gradient amplitude has 2 major constraints:
    - The number of gradient pulses switched simultaneously at different gradient channels. Note, the norm of the vector switched in read, phase and slice direction must be less or equal to 1 (one) to prevent components for the x y and z gradient coil to be above 100% gradient amplitude in case of oblique slices.
    - The amplitude difference of two adjacent gradient switching events switched with the system gradient rise time must not exceed 100% for each channel (r,p,s).
    - Consider differences of the maximum Phase2D gradient dependent on the imaging dimensionality.

$57\% = \frac{100}{\sqrt{3}}$ for oblique slices.

16. Modify the UpdateEchoTime routine called in backbone to include the new timing. Check the TR calculation in UpdateRepetitionTime: Is it necessary to change it too?
17. Inspect the structure of SetBaseLevelParam routine (BaseLevelRelations.c) and find out which subroutine is used to set up the delays. Set delay D[8] to the appropriate value.
18. Inspect the routine PrintTimingInfo. Adapt this testing routine and activate debug messages for the routines in BaseLevelRelations.c.

3.6 Possible pitfalls

The gradient amplitudes may exceed 100% if the limits calculation is not appropriate.

3.7 Possible extensions

None.

3.8 Proposed Solution

ltGre.ppg (complete listing; modified/new elements are printed boldface)

```
#include<Avance.incl>
#include <DBX.include>
preset off


;============================================================
;   definition of delays
;============================================================

define delay denab
"denab =  d4 - de + depa"

"l3 = l0 + ds"



;============================================================
;   declaration of 2d and 3d loop
;============================================================


lgrad r2d<2d> = L[0]
zgrad r2d
lgrad r3d<3d> = L[1]
zgrad r3d


if(DS>0)
{
  dsl,  dgrad r2d
  lo to dsl times DS
```

```
}

lgrad slice = NSLICES
zslice

#include <MEDSPEC.include>


;=================================================================
=
;     D/P                spec control          gradients
;=================================================================
=
start,      10u                fq8b:f1
      d1                fq1:f1              grad{(0)| (0) |(t0)}
      d2                gatepulse 1
      (p0:sp0 ph1):f1
      d8                                    grad{(0)| (0)|(t1)}
      d5                                    groff
      d3                                    grad{(t2)| r2d(t3)|
r3d(t4)}
      denab             REC_ENABLE          grad{(t5)| (0) | (0)}
                        ADC_INIT_B(ph1, ph0)
      aqq               ADC_START
      d3                                    grad{(t8)| r2d(t6) |
r3d(t7)}
      d6                                    grad{(t8)| (0) | (0) }
      d7                                    groff
        d0                ADC_END
      1u                                    islice
lo to start times NSLICES
      1u            ipp1                    zslice
lo to start times NA
      1u                rpp1                igrad r2d
lo to start times l3    ;2d loop
      1u                                    igrad r3d
      "l3=l0"
lo to start times l1    ;3d loop
lo to start times NAE
goto start
exit
;===============================================================
;phase lists

ph0 = 0
ph1 = 0 2 1 3
;===============================================================
```

initMeth.c - Necessary modification in the initMeth() function:

```
ReadDephGradLim =
      ExcSliceRephGradLim = ExcGradLim = 50.0;
```

(directly switching from positive to negative gradient, max. step is 100% during 1 risetime).

parsRelations.c

Necessary modification in the backbone routine:

```
(...)
ReadDephTime = Phase3DGradDur = Phase2DGradDur;
(...)
dim = PTB_GetSpatDim();

if(dim == 3)
{
      Phase2DGradLim = Phase3DGradLim = 61.0 /* close to 0.5*sqrt(3/2) */
}
else
{
      Phase2DGradLim = 86.0; /* close to 0.5*sqrt(3/2) */
      Phase3DGradLim = 0.1; /* not used in this case but > 0 for savety
reasons */
}
```

Necessary modification in the UpdateEchoTime routine:

```
(...)
*minte1 =
      ExcPulse.Length/2       +
      ExcSliceRephTime        +
      igwt                ;

*minte2 =
      riseTime                +
      Phase2DGradDur          +
      igwt ;
(...)
```

4 RF Pulses in PVM: A spin-echo method (project spinEcho)

4.1 Objectives

By developing this project you will learn the following topics:

- How to introduce new RF pulses into the sequence,
- How RF pulses are represented in PVM (struct RF_PULSE_PULSE_TYPE)
- How to handle RF pulses with toolbox functions,
- How to use RF pulse enums (struct PV_RF_PULSE_LIST) for an easy pulse shape selection,
- How to handle array parameters,
- How to calculate and use a new frequency list in the method code and in the pulse program.

4.2 Project description

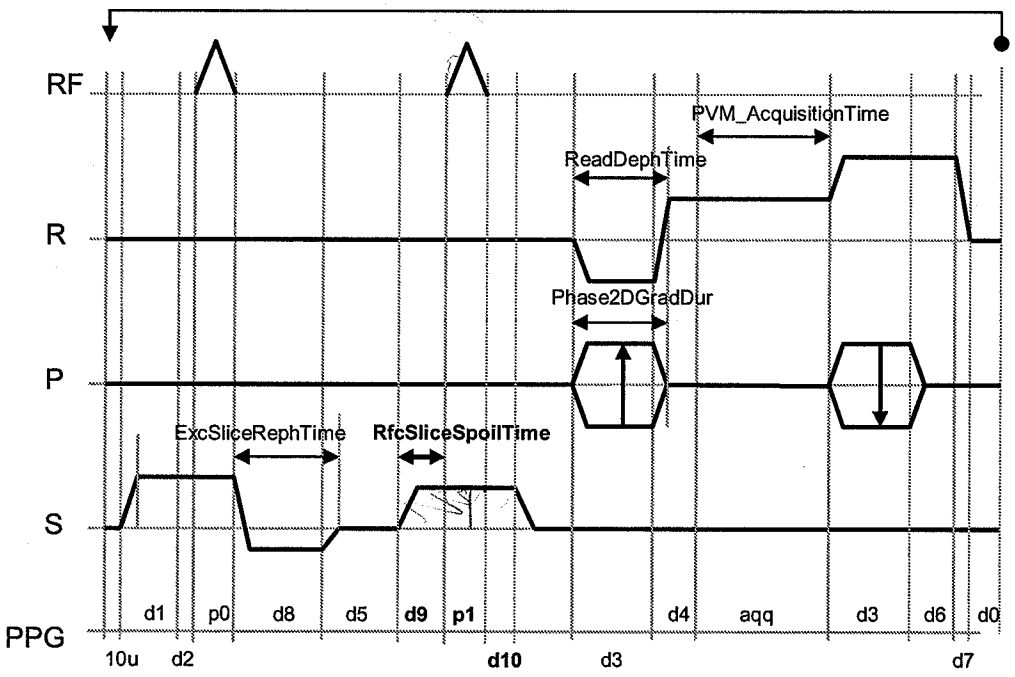
We will create a new method spinEcho based on ltGre, a variant of gre in which the slice rephasing and readout de-phasing periods are separated (see previous project). This arrangement is convenient for introducing the refocusing RF pulse since it reduces signal losses due to diffusion and flow. The project is organized in a series of short steps which can be programmed and tested progressively.

First, we introduce a refocusing group consisting of a refocusing RF pulse and a slice gradient of the same amplitude as used for the excitation. The RF pulse will be surrounded by fixed spoiling delays. We will focus on the RF pulse handling in this step and neglect the modified timing for a moment. Our goal will be to familiarize with the RF pulse tools provided in PVM.

In the second step we will introduce variable spoiler delays and take care of the correct echo positioning. This part will show you how timing issues are handled in PVM methods.

Finally, as and optional extension, we propose to introduce individual slice-selection gradient amplitudes for both RF pulses. This will require two separate frequency offset lists for both pulses – an occation to see how array parameters are used in the method code.

Method "spinEcho"



4.3 Specific method parameters

4.3.1 New method parameters

RfcPulse – a struct parameter of predefined type to be defined in RFPulsePars.h (to be implemented in the first step.

RfcPulseEnum – a pulse list parameter of predefined type to facilitate the selection of available refocusing pulses, controls the name of the pulseshape file.

RfcSliceGrad – a double parameter defining the refocusing slice selection gradient amplitude to be defined in SliceSelPars.h

RefSliceSpoilTime – a double parameter defining the time around the refocusing pulse to spoil unwanted coherences (FID of the excitation and refocusing pulse), to be defined in SliceSelPars.h

4.3.2 Important method parameters

ExcPulse – struct parameter already used to define the excitation pulse

ExcPulseEnum – a pulse list parameter of predefined type to facilitate the selection of available excitation pulses.

BwScale – a double parameter already defined to handle the transfer bandwidth of a multiple RF pulse slice selective preparation. Note that single RF pulse excitations as in case of the gradient echo variants of gre do not really need this parameter. It is predefined

for educational purposes to facilitate the gradient calculations in case additional RF pulses are implemented.

4.3.3 Important baselevel parameters

TPQQ – array of struct parameters to define RF pulse properties on the machine level

ACQ_O1_list ACQ_O2_list – array parameters holding the offset frequencies for slice selective excitation. Note that these parameters are bound to fq1 and fq2 commands in the pulseprogram.

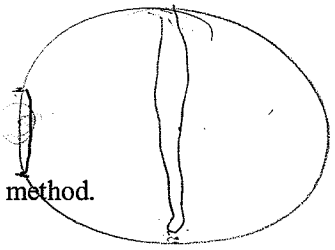
4.4 Important toolbox routines

The following toolbox routines are used to facilitate the handling of the different elements introduced in this project. The user may refer to the HTML toolbox documentation for detailed information:

STB_UpdateRfcPulseName – used to update the RF pulse name according to the state of the refocusing pulse list.

STB_InitRFPulse – type based initialisation routine for RF pulse struct parameters to assure valid ranges of the struct members.

4.5 Realisation hints



4.5.1 Starting point

The method should be developed based on the *ltGre* method.

4.5.2 Refocusing pulse in the pulse program

Use copyMethod to copy ltGre to spinEcho. Modify the pulse program spinEcho.ppg:

a. Introduce a new RF-Pulse command surrounded by two fixed delays of 1ms. Use P1 and sp1 and a new phase list ph2. Find out the correct place in the pulse program to do this.
b. Write the phase list ph2. One simple possibility is to keep the refocusing pulse orthogonal to the excitation pulse. Remaining lists will need no modification in this case.
c. During the first fixed delay switch on the slice gradient (t0). Switch it off after the second delay.
d. Copy the pulse program into the standard PP directory
e. Start ParaVision and create a scan with status "new", select the new spinEcho method.
f. Set the correct TPQQ entry to appropriate values in the ACQP editor and provide an appropriate pulse duration. Which TPQQ and P element needs to be modified?
g. Start a GSP and move the TX sliders to get the signal and to verify it is a spin echo.
h. Open the method editor again and have a look at the RF pulses parameter class: Open the RF pulse struct and have a look at the struct members. Modify the shapename and keep track to the reaction of the excitation pulse list parameter. You will access in the method code some of the struct elements. Modify the classification field and check what may be selected. This reaction is related to a switch statement in the range

checking routine of the ExcPulse parameter. You will need to modify this for the refocusing pulse.

4.5.3 Refocusing pulse in the method

Shut down ParaVision and introduce the new RF pulse into the method code by performing the following steps:

a. In RFPulsePars.h: Copy the ExcPulse parameter definition and call it RfcPulse. Remember the necessary renaming of display names and relations.

b. Make the new RF pulse visible by extension of the RF_pulses parameter class defined in RFPulseLayout.h.

c. In RFPulses.c, write the relations of the new pulse (RfcPulseRel) by copy-and-paste of the existing ExcPulseRel function. Remember about renaming of "ExcPulse" to "RfcPulse" in the new function!

d. Write the range checker (RfcPulseRange) by copy-and-paste of ExcPulseRange. Make sure this range checker is called in the relations of RfcPulse. Again, remember about renaming of ExcPulse to RfcPulse. Adapt the classification restriction (only LIB_REFOCUS, PVM_REFOCUS, and USER_PULSE should be allowed) as well as the flip angle limit.

e. Initialization RF Pulses Group (function InitRfPulses) – add a call of the range checker of RfcPulse. As you remember the range checkers also initialize parameters when they have no value.

f. Update of RF Pulses Group (function UpdateRFPulse). Add another call of the toolbox function STB_UpdateRfPulse with the new pulse as argument.

g. Transfer the state of RfcPulse to baselevel parameters. Find in BaseLevelRelations the subroutine which sets pulse delays and TPQQ settings and add the proper entries for the refocusing pulse.

Recompile the method and start ParaVision. The refocusing pulse should be editable and the correct flip-angle/gains and bandwidth/duration calculations should take place. You can try the sequence with signal, however, the image will still have mixed properties of spin- and gradient-echoes because both echoes are not yet centred – this will be corrected later.

4.5.4 Pulse Choice Enum (optional)

In addition to the struct parameter describing the refocusing pulse, a special enum parameter can be added to allow choosing between available pulse shapes (you can also select any shape by typing its name directly in the Fileneme field of the struct, but this is less convenient). Such an enum parameter already exists for the excitation pulse (ExcPulseEnum). We will add another one for the refocusing (RfcPulseEnum) by copying proper parts of the code:

a. In RFPulsePars.h: Add a definition of RfcPulseEnum,

b. In RFPulseLayout.h: Add the RfcPulseEnum to the definition of RF_Pulses parclass.

c. In RfPulses.c: Write relations of the new enum (RfcPulseEnumRel). In the relations, call STB_Update**Rfc**PulseName with proper arguments. This assures the change of the pulse's Filename when the enum is changed. Range-checking of the enum is not needed (because of the next step)

d. In the function UpdateRFPulses (file RFPulses.c): call STB_UpdateRfcPulseEnum with proper arguments. This changes the value of the enum when the Filename of the related pulse is changed.

Recompile the method and restart PV. Check the interaction between RfcPulse.Filename and RfcPulseEnum.

4.5.5 Controlling the spin-echo position

Our next goal is to get the gradient echo in the centre of the spin echo and to make sure the echo time is correctly calculated. We will also introduce a parameter to control the duration of the spoilers on both sides of the refocusing pulse.

a. in the pulse program use delays d9, d10, d11 in the refocusing group, as in the sequence diagram. The echo time and the centering will be controlled by d5 and d11.

b. Introduce a new parameter "RfcSliceSpoilerDuration" as a member of the SliceSelection group. Give it a minimum of CFG_GradientRiseTime() in its range check. It will affect delays d9 and d10. Call the range check in the InitRFPulses function.

c. In parsRelations.c, edit the funtion UpdateEchoTime. It calculates two variables *minte1*, and *minte2*, which hold the sum of all fixed delays from the excitation to the refocusing, and from the refocusing to the echo, respectively. You should add the spoiler duration and one half of the refocusing pulse to both of them. Additionally, one rise time should be added to *minte2* as a minimum value of d11. Finally, change the restriction of the echo time (PVM_EchoTime). The minimum is now given by two times *minte1* or two times *minte2* whichever one is longer. (Attention: these two parameters are passed by pointers; the asterisk is used to access their values).

d. Verify the function UpdateRepetitionTime. Is any modification needed?

e. Pass the values of minte1 and minte2 from the backbone via BaseLevelRelations() to SetPpgParameters(), where they will be needed to set delays d5 and d11. For this purpose, two double arguments must be added to the interfaces BaseLevelRelations and SetPpgParameters.

f. In SetPpgParameters, calculate d5/d11 as half the difference between PVM_EchoTime and minte1/minte2, plus rise time (remember about ms-to-sec conversion). Set d9 to RfcSliceSpoilerDuration and d10 to RfcSliceSpoilerDuration – CFG_GradientRiseTime(). This apparent asymmetry will compensate the delay of the gradient system.

Recompile the method and restart PV. Check the reaction of the echo time to modifications of spoiler duration, acquition size, pulse durations. Check in GSP mode whether the spin echo coincides with the gradient echo. For this purpose, turn the read gradient off by setting ACQ_scaling_read to zero. You should now see the pure spin-echo envelope and and its maximum should correspond to the gradient echo centre (you may make it narrower with shims).

4.6 Possible extensions

4.6.1 Different amplitudes of excitation and refocusing slice gradients

So far, both RF pulses are applied with the same amplitude of the slice selection gradient. The effective slice thickness is determined by the pulse of a narrower bandwidth. In the next step we will adjust the slice selection gradient individually to each pulse so that they always act at the same slice thickness. As a consequence both pulses will also need individual frequency offset lists.

1. Implementation of slice gradient and slice spoiling: (stop ParaVision)

- Define parameters RfcGrad in SliceSelPars.h. We use backbone as default relation of RfcGrad. Any input to this parameter should be replaced by a value calculated there. This is a common approach for parameters whose values are of interest but which should not be modified directly by user input.
- Handle visibility of RfcGrad. The visibility of all slice selection parameters is handled by the locally defined routine SliceSelectionParsVisibility. Modify this routine appropriately
- Consider the refocusing slice selection gradient for the calculation of the minimum slice thickness. You should:
  - provide an additional argument for routine SliceSelectionLimits, namely a pointer to an RF pulse (PVM_RF_PULSE_TYPE *const rfcPulse).
  - Declare a new double variable "min" to hold the minimum slice thickness with regard to the refocusing slice gradient
  - (optional) perform a trivial type based range check for the argument rfcPulse using STB_CheckRFPulse. If called appropriately in backbone, the argument will not be changed. For savety reasons (to avoid division by zero for illegal pulse bandwidth) this call is nevertheless a good practice.
  - Calculate the minimum slice thickness so that RfcSliceGrad does not exceed 90% gradient amplitude using routine MRT_MinSliceThickness (hint: the slice ratio is 1.0 in this example, why?). Store return value in variable "min"
  - Consider the "min" in the return value of SliceSelectionLimits
- Calculate the refocusing slice gradient:
  - Define a new argument const double rfcPulseBW (effective bandwidth used for the slicegradient calculation) for routine UpdateSliceSelectionGradients. Check this argument to be greater than zero (0).
  - Use MRT_SliceGrad to calculate RfcSliceGrad
- Adapt backbone routine
  - Adapt the call of SliceSelectionLimits (think about the new argument RfcPulse!)
  - Adapt the call of UpdateSliceSelectionLimits rfcPulseBW (here you may consider the bandwidth scaling factor!)
- Calculate an additional frequency list
  - Modify routine SetFrequencyParameters (BaseLevelRelations.c)
    - Change the dimension of ACQ_O2_list to the total number of slices (note the total number of slices has been calculated for the setting of the offsets for the slice excitation. Why do we have to set ACQ_O2_list (hint: see syntax in pulseprogram).
    - Calculate the ACQ_O2_list using the ACQ_O1_list. What is the relation between offsets for the excitation and for the refocusing slice? Use UT_ScaleDoubleArray to do the job.

Recompile the method, rstart ParaVision and check the extended version of spinEcho.

4.7 Possible pitfalls

- Forgotten UT_SetRequest in the relations of RF pulses – the backward requests (e.g. Bandwidth and Length, Attenuation and FlipAngle) will not be accepted. Bandwidth and Attenuation appear noneditable. The request mechanism will be explained in more detail in a later project (greTag).

- No signal for offcenter slice acquisition:
  - Wrong frequency list.
  - Frequency list not used (check pulseprogram)
- Wrong amplitudes for slice gradients:
  - error in handling of bandwidth scaling factor
  - wrong handling of minimum slice thickness: Correct return value of SliceSelectionLimits, Correct arguments for this routine (check call in backbone)
- Compiler errors concerning usage of unknown functions or wrong function prototypes:
  - Forgot to do a make cproto depend before the compile (especially if function arguments have been changed!)

4.8 Proposed Solution

spinecho.ppg (complete listing; modified/new elements are printed boldface)

```
#include<Avance.incl>
#include <DBX.include>
preset off


;===========================================================
;   definition of delays
;===========================================================

define delay denab
"denab =  d4 - de + depa"

"l3 = l0 + ds"

;===========================================================
;   declaration of 2d and 3d loop
;===========================================================

lgrad r2d<2d> = L[0]
zgrad r2d
lgrad r3d<3d> = L[1]
zgrad r3d

if(DS>0)
{
  dsl,  dgrad r2d
  l0 to dsl times DS
}

lgrad slice = NSLICES
zslice

#include <MEDSPEC.include>

;=============================================================================
;      D/P                spec control          gradients
;=============================================================================
start, 10u               fq8b:f1
       d1                fq1:f1                grad{( 0)| (0) |(t0)}
```

```
       d2                gatepulse 1
       (p0:sp0 ph1):f1
       d8                                    grad{(0)| (0)|(t1)}
       d5                groff fq2:f1
       d9                grad{(0)|(0)|(t9)}
       (p1:sp1 ph2):f1
       d10
       d11               groff
       d3                grad{(t2)| r2d(t3)| r3d(t4)}
       denab             REC_ENABLE          grad{(t5)| (0) | (0)}
                         ADC_INIT_B(ph1, ph0)
       aqq               ADC_START
       d3                                    grad{(t8)| r2d(t6) | r3d(t7) }

       d6                                    grad{(t8)| (0) | (0) }
       d7                                    groff
       d0                ADC_END
       lu                                    islice
lo to start times NSLICES
       lu             ipp1                   zslice
lo to start times NA
       lu                rpp1                igrad r2d
lo to start times l3
       lu                                    igrad r3d
   "l3=l0"
lo to start times l1
lo to start times NAE
SETUP_GOTO(start)
exit
;===============================================================
;phase lists

ph0 = 0
ph1 = 0 2 1 3
ph2 = 1 3 2 0
;===============================================================
```

RFPulsePars.h:

```
/*
 * refocusing pulse parameter
 */

PV_PULSE_LIST parameter
{
  display_name "Refocusing Pulse Choice";
  relations    RfcPulseEnumRel;
}RfcPulseEnum;


PVM_RF_PULSE_TYPE parameter
{
  display_name "Refocusing Pulse";
  relations    RfcPulseRel;
}RfcPulse;
```

RFPulseLayout.h:

```
parclass
{
  BwScale;
```

```
  ExcPulseEnum;
  ExcPulse;
  RfcPulseEnum;
  RfcPulse;
} RF_Pulses;
```

RFPulses.c

```
void InitRFPulses(void)
{
  DB_MSG(("-->InitRFPulses"));

  BwScaleRange();
  ExcPulseRange();
  RfcPulseRange();
  DeriveGainsRange();
  STB_InitExcPulseEnum("ExcPulseEnum");
  STB_InitRfcPulseEnum("RfcPulseEnum");

  DB_MSG(("<--InitRFPulses"));

  return;
}


YesNo UpdateRFPulses(YesNo deriveGains,char *nucleus)
{
  YesNo referenceAviable;
  double referenceAttenuation=30;

  DB_MSG(("-->UpdateRFPulses"));

  if(deriveGains == Yes)
    referenceAviable =
      CFG_GetGlobRefAtt(nucleus,&referenceAttenuation);
  else
    referenceAviable = No;

  STB_UpdateRFPulse("ExcPulse",
                    &ExcPulse,
                    referenceAviable,
                    referenceAttenuation);

  STB_UpdateExcPulseEnum("ExcPulseEnum",
                         &ExcPulseEnum,
                         ExcPulse.Filename,
                         ExcPulse.Classification);

  STB_UpdateRFPulse("RfcPulse",
                    &RfcPulse,
                    referenceAviable,
                    referenceAttenuation);

  STB_UpdateRfcPulseEnum("RfcPulseEnum",
                         &RfcPulseEnum,
                         RfcPulse.Filename,
                         RfcPulse.Classification);



  DB_MSG(("<--UpdateRFPulses"));
```

```
  return referenceAviable;

}

(...)

void RfcPulseEnumRel(void)
{
  DB_MSG(("-->RfcPulsesEnumRel"));

  /* set the name and clasification of RfcPulse: */

  STB_UpdateRfcPulseName("RfcPulseEnum",
                         &RfcPulseEnum,
                         RfcPulse.Filename,
                         &RfcPulse.Classification);

  /* call the method relations */
  backbone();

  DB_MSG(("<--RfcPulseEnumRel"));
}


void RfcPulseRange(void)
{
  DB_MSG(("-->RfcPulseRange"));

  if(ParxRelsParHasValue("RfcPulse") == No)
  {
    STB_InitRFPulse(&RfcPulse,
                    "gauss.rfc",
                    1.0,
                    180.0);
  }

  /* allowed clasification */

  switch(RfcPulse.Classification)
  {
    default:

      RfcPulse.Classification = LIB_REFOCUS;
      break;
    case LIB_REFOCUS:
    case PVM_REFOCUS:
    case USER_PULSE:
      break;
  }

  /* allowed angle for this pulse */

  RfcPulse.FlipAngle = MIN_OF(180.0,RfcPulse.FlipAngle);
  RfcPulse.FlipAngle = MAX_OF(90.0,RfcPulse.FlipAngle);

  /* general verifiation of all pulse atributes  */

  STB_CheckRFPulse(&RfcPulse);

  DB_MSG(("<--RfcPulseRange"));

}
```

```
void RfcPulseRel(void)
{
  DB_MSG(("-->RfcPulseRel"));

  /*
   * Tell the request handling system that the parameter
   * RfcPulse has been edited
   */

  UT_SetRequest("RfcPulse");

  /* Check the values of RfcPulse */

  RfcPulseRange();

  /*
   * call the backbone; further handling will take place there
   * (by means of STB_UpdateRFPulse)
   */

  backbone();

  DB_MSG(("<--RfcPulseRel"));
}
```

SliceSelPars.h

```
double parameter
{
  display_name "Ref. Slice Gradient";
  relations backbone;
  units "%";
  format "%f";
}RfcSliceGrad;

double parameter
{
  display_name "Ref Slice Spoiling Time";
  relations RefSliceSpoilTimeRel;
  units "ms";
  format "%f";
}RefSliceSpoilTime;
```

SliceSelLayout.h

```
parclass
{
  SliceGradStabTime;
  ExcSliceRephTime;
  RefSliceSpoilTime;
  ExcSliceGrad;
  RfcSliceGrad;
  ExcSliceGradLim;
  ExcSliceRephGrad;
  ExcSliceRephGradLim;
}SliceSelection;
```

SliceSel.c:

```
void SliceSelectionParsVisibility(YesNo showAllPars)
{
  const char *const editable = "ExcSliceRephTime,"
                               "RefSliceSpoilTime";


  const char *const nonedit  = "ExcSliceGrad,"
                               "SliceGradStabTime,"
                          "ExcSliceGradLim,"
                          "ExcSliceRephGrad,"
                               "ExcSliceRephGradLim,"
                               "RfcSliceGrad";


  DB_MSG(("-->SliceSelectionParsVisibility"));


(...)

void InitSliceSelection(YesNo showAllPars)
{
 ...
  ExcSliceRephGradLimRange();
  RefSliceSpoilTimeRange(1.0);

  SliceSelectionParsVisibility(showAllPars);

  DB_MSG(("<--InitSliceSelection"));
}

double SliceSelectionLimits(PVM_RF_PULSE_TYPE *const excPulse,
                        PVM_RF_PULSE_TYPE *const rfcPulse,
                        const double gradStabTime,
                        const double gradCalConst,
                        double *const sliceRatio)

{
 (...)

 /* range check of arguments */

  STB_CheckRFPulse(excPulse);
  STB_CheckRFPulse(rfcPulse);


(...)
  min      = MRT_MinSliceThickness(rfcPulse->Bandwidth,
                                   1.0,
                                   90.0,
                                   90.0,
                                   gradCalConst);


  DB_MSG(("<--SliceSelectionLimits"));
  return MAX_OF(minSlThk,min);
}

YesNo UpdateSliceSelectionGradients(const double slthk,
                                    const double sliceRatio,
                                    const double excPulseBW,
                                    const double rfcPulseBW,
                                    double gradCalConst)
```

```
{
  DB_MSG(("-->UpdateSliceSelectionGradients"));


  if(slthk <= 0.0)
  {
    UT_ReportError("UpdateSliceSelectionGradients: "
                   "Illegal value of argument 1\n");
    return No;

  }


  if(excPulseBW < 0.0 || rfcPulseBW <0.0)
  {
    UT_ReportError("UpdateSliceSelectionGradients: "
                   "Illegal value of argument 2,3\n");
    return No;

  }

(...)
  RfcSliceGrad      = MRT_SliceGrad(rfcPulseBW,slthk, gradCalConst);
(...)

  DB_MSG(("<--UpdateSliceSelectionGradients"));
  return Yes;
}


(...)

void RefSliceSpoilTimeRange(double gradstab)
{
  double min;
  DB_MSG(("-->RefSliceSpoilTimeRange"));

  min = CFG_GradientRiseTime();

  min += gradstab > 0.0 ? 0.0:gradstab;
  min =  MIN_OF(min,10.0);

  if(!ParxRelsParHasValue("RefSliceSpoilTime"))
  {
    RefSliceSpoilTime = min;
  }
  else
  {
    RefSliceSpoilTime = MAX_OF(MIN_OF(RefSliceSpoilTime,10.0),min);
  }


  DB_MSG(("<--RefSliceSpoilTimeRange"));
}

void RefSliceSpoilTimeRel(void)
{
  DB_MSG(("-->RefSliceSpoilTimeRel"));

  RefSliceSpoilTimeRange(0.0);
  backbone();

  DB_MSG(("<--RefSliceSpoilTimeRel"));
}
```

ParsRelations.c

```
void backbone(void)
{

(...)

  minSliceThick = SliceSelectionLimits(&ExcPulse,
                                       &RfcPulse,
                                            GradStabTime,
                                       PVM_GradCalConst,
                                       &sliceGradRatio);
  /*
   *  update geometry parameters
   */

(...)

  /*
   *  calculate gradients in logical directions
   */

  FreqEncodingGradients(PVM_Fov[0],readGradRatio,PVM_GradCalConst);
  UpdatePhase2DGradients(PVM_Matrix[1],PVM_Fov[1],PVM_GradCalConst);

  if(dim == 3)
  {
    UpdatePhase3DGradients(PVM_Matrix[2],PVM_Fov[2],PVM_GradCalConst);
  }
  else
  {
    Phase3DGrad =0.0;
  }

  UpdateSliceSelectionGradients(PVM_SliceThick,
                                sliceGradRatio,
                                ExcPulse.Bandwidth*BwScale/100.0,
                           RfcPulse.Bandwidth*BwScale/100.0,
                           PVM_GradCalConst);



  /*
   *  calculate frequency offsets
   */

  LocalFrequencyOffsetRels();

  /*
   *  update sequence timing
   */


  UpdateEchoTime(&minte1,&minte2);
  UpdateRepetitionTime();

  PVM_NEchoImages = 1;


  SetBaseLevelParam(minte1, minte2);
  SetRecoParam();
```

```
  DB_MSG(("<--backbone"));
  return;
}


void UpdateEchoTime(double *const minte1, double *const minte2)
{
  double riseTime, igwt;


  DB_MSG(("-->UpdateEchoTime"));

  riseTime = CFG_GradientRiseTime();
  igwt     = CFG_InterGradientWaitTime();

  *minte1 =
    ExcPulse.Length/2                    +
    ExcSliceRephTime                     +
    igwt                               +  /* min TE1/2 filling delay */
    RefSliceSpoilTime                    +
    RfcPulse.Length/2;

  *minte2 =
    RfcPulse.Length/2                      +
    RefSliceSpoilTime                      +
    igwt                               + /* min TE2/2 filling delay */
    Phase2DGradDur                         +
    PVM_AcqStartWaitTime                   +
    PVM_AcquisitionTime * PVM_EchoPosition/100;


  PVM_MinEchoTime = 2*MAX_OF(*minte1, *minte2);
  PVM_EchoTime    = MAX_OF(PVM_MinEchoTime,PVM_EchoTime);
  PVM_EchoTime1   = PVM_EchoTime2 = PVM_EchoTime;



  DB_MSG(("<--UpdateEchoTime"));
  return;
}
```

(...)

BaseLevelRelations.c

```
void SetBaseLevelParam(double minte1, double minte2)
{

  DB_MSG(("-->SetBaseLevelParam"));

  SetBasicParameters();
```

(...)

```
  SetPpgParameters(minte1,minte2);
```

(...)

```
  DB_MSG(("<--SetBaseLevelParam"));

}


void SetFrequencyParameters( void )
{
  int nslices;

  DB_MSG(("-->SetFrequencyParameters"));

  ATB_SetNuc1(PVM_Nucleus1);

(...)

  nslices = GTB_NumberOfSlices( PVM_NSPacks, PVM_SPackArrNSlices );
  ATB_SetAcqO1List( nslices,
                    PVM_ObjOrderList,
                    PVM_SliceOffsetHz );

  PARX_change_dims("ACQ_O2_list",nslices);

  UT_ScaleDoubleArray(nslices,
                      ACQ_O1_list,
                      RfcSliceGrad/ExcSliceGrad,
                      ACQ_O2_list);

  ATB_SetAcqO1BList( nslices,
                     PVM_ObjOrderList,
                     PVM_ReadOffsetHz);


(...)

  DB_MSG(("<--SetFrequencyParameters"));
}


void SetGradientParameters( void )
{
  int spatDim, dim, i;

  DB_MSG(("-->SetGradientParameters"));

(...)

  ATB_SetAcqTrims( 10,
                    ExcSliceGrad,                    /* t0 */
                   -ExcSliceRephGrad,                /* t1 */
                   -ReadDephGrad,                    /* t2 */
                   Phase2DGrad,                      /* t3 */
                   -Phase3DGrad,                     /* t4 */
                   ReadGrad,                         /* t5 */
                   -Phase2DGrad,                     /* t6 */
                   Phase3DGrad,                      /* t7 */
                   ReadSpoilGrad,                    /* t8 */
                   RfcSliceGrad                      /* t9 */
                    );

(...)

  DB_MSG(("<--SetGradientParameters"));

}
```

```
void SetPpgParameters(double minte1,double minte2)
{
  double riseT,igwT;
  int slices;
  DB_MSG(("-->SetPpgParameters"));

  (...)

  D[0]  = ((PVM_RepetitionTime - PVM_MinRepetitionTime)/slices
          + igwT)/1000.0;
  D[1]  = (SliceGradStabTime + riseT)/1000.0;
  D[2]  = CFG_AmplifierEnable()/1000.0;
  D[3]  = (Phase2DGradDur - riseT) / 1000.0;
  D[5]  = (PVM_EchoTime/2 -minte1  + riseT + igwT)/1000.0;
  D[11] = (PVM_EchoTime/2 -minte2  + riseT + igwT)/1000.0;


  D[4]  = (riseT + PVM_AcqStartWaitTime)/1000.0;
  D[6]  = (ReadSpoilGradDur - Phase2DGradDur)/1000.0;
  D[7] = riseT/1000.0;
  D[8]  = (ExcSliceRephTime - riseT)/1000.0;

  D[9] = RefSliceSpoilTime/1000.0;
  D[10]= (RefSliceSpoilTime-riseT)/1000.0;



  /* set shaped pulses     */
  sprintf(TPQQ[0].name,ExcPulse.Filename);
  if(PVM_DeriveGains == Yes)
  {
    TPQQ[0].power  = ExcPulse.Attenuation;
  }
  TPQQ[0].offset = 0.0;

  sprintf(TPQQ[1].name,RfcPulse.Filename);
  if(PVM_DeriveGains == Yes)
  {
    TPQQ[1].power  = RfcPulse.Attenuation;
  }
  TPQQ[1].offset = 0.0;


  ParxRelsParRelations("TPQQ",Yes);

  /* set duration of pulses          */
  P[0] = ExcPulse.Length * 1000;
  P[1] = RfcPulse.Length * 1000;
  ParxRelsParRelations("P",Yes);

  DB_MSG(("<--SetPpgParameters"));
}
```

Method Programming in PVM Part 2

ParaVision Programming Course
April 3 – 7, 2006

Authors:

Franciszek Hennel

Sascha Kőhler

Markus Wick

Bruker BioSpin MRI GmbH

Table of Contents

Introduction

The first project for today (*satGre*) demonstrates the **usage of modules** in PVM. We will extend the gradient echo method *gre* by a series of slice-selective saturation RF pulses. Saturation slices are commonly used to avoid alising of signals from outside the field of view. Positions, orientation and thickness of the saturation slices will be controlled by the Geometry editor. All we will need to add this quite complex functionality to *gre* is to include a special module in the pulse program and provide its support in the PVM code. As you will see, a pulse program module comes along with a dedicated group of PVM parameters and with toolbox functions for initializing and updating of the group. A similar modularity is proposed in PVM for several other applications: the echo-planar imaging readout, diffusion wighting, water or fat suppression, and several other preparation sequences.

The goal of the next project (*greTag*) is to show how you can **develop your own PVM modules**. We will program a pulse program module for spin tagging and include it in the gradient echo sequence. The group of parameters controlling this module will be handled in a similar way as the predefined groups used for the saturation or geometry. We will see that this modular design of method elements is very useful when similar features need to be implemented in various methods.

As already mentioned, every parameter group has an updating function and we will also write one for our tagging module. The essential problem of such functions is to react properly to modifications of group members. When members A and B are to be related, should A be derived from B, or the other way round? This should be dependent on which parameter has last been modified by the user. We will make use of the **request handling mechanism** which provides exactly this type of information.

Finally, we will explore a new feature of PVM introduced in ParaVision 4.0 allowing a simple implementation of **accelerated imaging methods**. A variant of gradient echo will be programmed (*encGre*) using the *Encoding* parameter group to handle all common schemes of accelerated phase encoding, such as parallel imaging, partial-FT and zero-filling. We will also show how this parameter group can be used for a segmented k-space scanning.

1 Using Modules: Saturation Slices (satGre)

1.1 Objectives

By developing this project you will learn how to

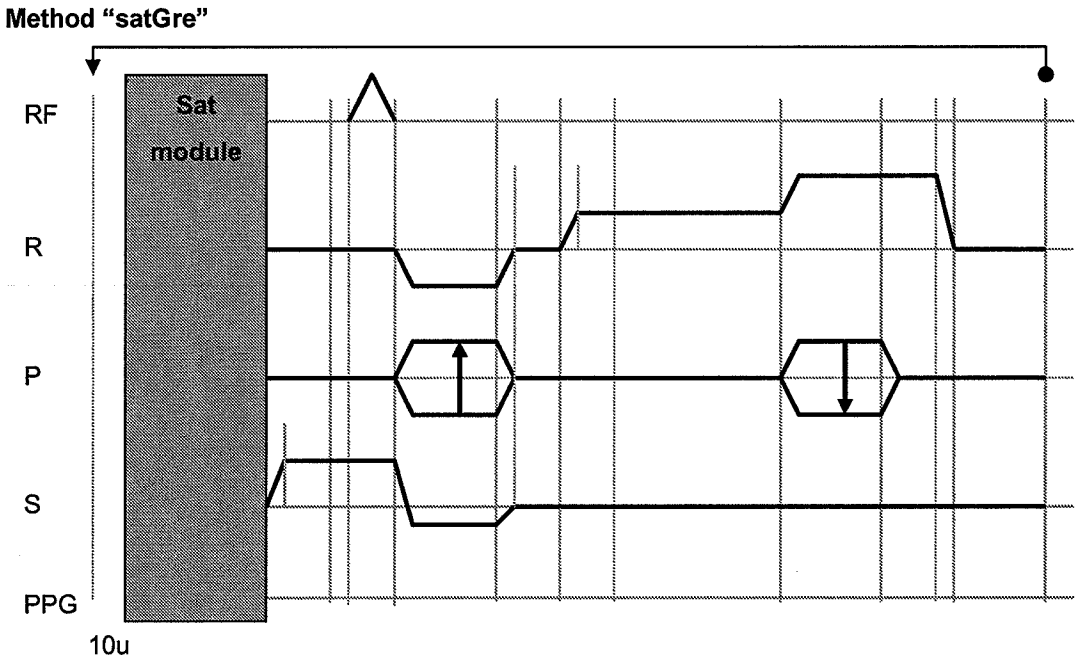
- use a preparation module based on a globally defined parameter class
- modify the functionality of a preparation module
- handle multidimensional array parameters
- use a pulseprogram subroutine

1.2 Project description

The template gre will be copied to a new method satGre. First the saturation slices module will be introduced into the method. This module is implemented as a standard preparation module consisting of a dedicated set of PVM parameters and a parclass providing a user interface. In addition this module is connected to the routine geometry editor to assure a saturation slice placement based on reference images.

Parameter initialisation, -update and the setting of the baselevel parameters that are associated to the pulseprogram subroutine are based on toolbox functions. These functions are used also to control the visibility of the parameters as shown in the method editor and the geometry editor respectively.

In a second step the saturation slices are constrained to lie always parallel to the imaging slices. Here the gradient matrix (as defined in the standard slice geometry parameter group) is used to provide a gradient vector that is transferred to the related parameter of the saturation slices group.



1.3 The Saturation Slices Parameter Group

Parameters, that define the Saturation Slices are grouped in the Sat_Slices_Parameters parclass:

PVM_FovSatNSlices: The number of saturation slices represented as int.

PVM_FovSatSliceOrient: The major orientation of saturation slice (axial, sagittal, coronal) implemented as array of corresponding enums.

PVM_FovSatThick: The thickness (defined by the FWHM of the excitation/inversion profile), implemented as double array with PVM_FovSatNSlices entries.

PVM_FovSatOffset: The distance to the gradient isocenter, implemented as double array with PVM_FovSatNSlices entries.

PVM_FovSatSliceVec: The slice orientation for each saturation slice, implemented as an array of 3 element vectors with PVM_FovSatNSlices entries.

PVM_SatSlicesPulseEnum: The pulse list parameter for the RF pulse used for slice selective saturation/inversion.

PVM_SatSlicesPulse: The pulse struct parameter to define the specifications of the shaped RF pulse. Dependent on the Classification field it may be specified whether an excitation pulse (for saturation purposes, LIB_EXCITATION) or a slice selective inversion pulse is used. The pulse list parameter will represent either a list of excitation pulses or inversion pulses according to the value specified in this field.

PVM_SatSlicesDeriveGainMode: Dependent on this parameter, the derivation of a RF pulse amplitude is handled according to the state of PVM_DeriveGains (YesNo parameter in the main

method parclass) or according to the availability of a reference attenuation to force RF amplitude derivation independent on the state of PVM_DeriveGains.

PVM_FovSatGrad: The amplitude of the slice gradient implemented as a double array with PVM_FovSatNSlices entries.

PVM_FovSatSpoilTime: Duration of the spoiling gradient used to suppress unwanted coherences invoked by the slice selective excitation/inversion.

PVM_FovSatSpoilGrad: Gradient amplitude of the spoiling gradient.

PVM_FovSatModuleTime: Total duration of the SatSlices module (needed to calculate the repetition time of the sequence).

The group handler of the saturation slices parameter group is PVM_SatSlicesHandler.

Parameter PVM_FovSatOnOff is used to switch saturation slices on or off. Only if set to On, the parameter group is visible in the method editor and saturation slices may be defined in the geometry editor.

1.4 Toolbox functions associated to the Saturation Slices Parameter Group

Parameter initialisation: STB_InitSatSlicesModule, used to provide legal starting values. This function is intended to be called in the initMeth routine of the method.

Parameter update: STB_UpdateSatSlicesModule, used to provide a consistent state of the parameter group. This function must be called in the backbone of the method.

Baselevel parameter setting: ATB_SetSatSliceBaseLevel, used to derive the baselevel parameters used in the pulse program part according to the state of the saturation slice parameter group.

1.5 Pulse Program Subroutine SatSlices()

A new mechanism to handle common used functionality on pulse program level is the concept of subroutines. Subroutines are defined in the header of a pulse program and may be called inside the pulseprogram body. The related pulseprogram part for this project is available after an include of PrepModulesHead.mod in the pulseprogram header. This is a common include file for most of the preparation modules delivered with the software.

To apply the saturation slices, the subroutine SatSlices() must be called at a dedicated part of the major pulse program.

1.6 Realization Hints: Usage of Sat_Slices_Parameters parclass.

1.6.1 Creation of satGre

To perform the exercises create your version of method satGre and a reasonable protocol of this method by following these steps:

- Copy method gre from the /opt`/PV4.0/prog/parx/src directory in your method development environment.
- Install the pulse program satGre.ppg as created by copyMethod into the pulse program directory of your ParaVision® installation (/opt/PV4.0/exp/stan/nmr/lists/pp).

1.6.2 Modification of Method Sources

The following approach is typical for the handling of modules defined by a global parameter group:

1. Define a parclass "Preparations" in parsLayout.h. Members of this class are PVM_FovSatOnoff and the parclass Sat_Slices_Parameters. Place this parclass inside the methodClass. The parclass is now visible in the method editor.
2. Initialize the parameter group by calling STB_InitSatSliceModule (please refer to the related online documentation) in initMeth (before the backbone call).
3. Update the parameter group with STB_UpdateSatSlicesModule. This toolbox function requires the final value of PVM_Nucleus1 (to derive the gyromagnetic ratio for gradient calculations) and must be called in the backbone routine (parsRelations.c) after the update of the nuclei. Since the minimum TR depends now on the duration of this module, the update must be called before the TR calculation. To prepare also the second part (modification of module functionality), the slice geometry parameters should have been updated before the call of STB_UpdateSatSlicesModule.
4. Consider the duration of the module in the TR calculation (modification of local routine UpdateRepetitionTime).
5. Use ATB_SetSatSlicesBaseLevel in routine SetBaseLevelParam to derive the acquisition parameters for the pulseprogram include.
6. Modify callbackDefs.h to assure a redirection of the group handler PVM_SatSlicesHandler to the methods backbone routine.
7. Modify the pulseprogram satGre.ppg: include PrepModulesHead.mod and call the subroutine SatSlices in the pulseprogram body.
8. Test the method: Restart ParaVision, switch on saturation slices and place them inside the geometry editor. Acquire Images to assure the saturation is done as planned in the geometry editor

1.7 Realization Hints: Modification of Global Module Functionality

In this second part of the project the direction of the saturation slice vectors should be constrained to be along the orientation of the first slice package. This functionality should be dependent on a new (locally defined) parameter SatParallel of type YesNo. This parameter should be visible in the Preparations parclass only if the saturation slices module is switched on. Before the parameter group is updated, the last entry of the gradient matrix (parameter

PVM_SpackArrayGradOrient[0][2]) is copied in each entry of PVM_SatSliceVec in case SatParallel is set to Yes.

1.7.1 Definition of a parameter controlling the orientation constrain

- Define a YesNo parameter SatParallel (parsDefinition.h), the default relation should be SatParallelRel.
- Make it visible in the new Preparations subclass of the method (parsLayout.h)
- Write a standard range checking function SatParallelRange and the default relation SatParallelRel in parsRelations.c
- Call the range checking function in the default relation as well as in initMeth() of the method (force parameter initialization)

1.7.2 Modifications in the backbone

These modifications are related to the handling of multidimensional double array parameters. Parameter PVM_SpackArrayGradOrient is implemented as double [i][j][k], where i is an index running from 0 to PVM_NSPacks-1, j and k is running from 0 to 2. Whereas j relates to the read (j=0), phase (j=1) and slice (j=2) direction. The index k denotes the x (k=0), y (k=1) and z (k=2) component of the gradient vector in the patient coordinate system. The implementation of multidimensional arrays allows an easy access to the row vectors of the gradient matrix:

PVM_SpackArrGradOrient[0][2] is the slice orientation vector of the first slice package. Parameter PVM_FovSatSliceVec is implemented as double [i][j] array, where i is an index running from 0 to PVM_FovSatNslices-1, j is the index for the components of the saturation slice vectors.

The c-function memcpy may be used to transfer the slice vector defined in the gradient matrix to the slice vectors of the saturation slices as follows:

```
memcpy(PVM_FovSatSliceVec[i],PVM_SpackArrGradOrient[0][2],3*sizeof(double));
```

The orientation transfer should be done before STB_UpdateSatSlicesModule is called (why?):

1. Write an if-statement, testing whether SatParallel==Yes
2. Define in the body of the if-statement int variables to hold the actual dimension of the saturation slices gradient vector array (dim) as well as a running index i to implement a loop. Define a double array vec[3] to save the slice vector defined by the orientation matrix.
3. Copy the slice vector of the orientation matrix into the local variable vec (using memcpy).
4. Use PARX_get_dim("PVM_FovSatSliceVec",1) to get the dimension of this array.
5. Loop over all vectors of PVM_FovSatSliceVec and copy the orientation stored in the local variable vec into each vector.

6. Control the visibility of SatParallel according to the state of PVM_FovSatOnOff. (use function ParxRelsHideInEditor)

1.8 Possible Pitfalls:

- The parameter group does not react on user input: Relation redirection in callbackDefs.h is missing?
- TR is not calculated correctly: Adaption of UpdateRepetitionTime is missing?
- Saturation is performed always perpendicular to the slice orientation: Is the correct index of PVM_SpackArrGradOrient used to provide the saturation orientation?

1.9 Proposed Solution

(modifications in boldface):

File parsLayout.h:

```
parclass
{
  PVM_FovSatOnOff;
  SatParallel;
  Sat_Slices_Parameters;
}Preparation;

parclass
{
  Method;
  PVM_EchoTime;
  PVM_MinEchoTime;
  PVM_RepetitionTime;
  PVM_NAverages;
  PVM_ScanTimeStr;
  PVM_DeriveGains;
  RF_Pulses;
  Nuclei;
  SequenceDetails;
  Preparation;
  StandardInplaneGeometry;
  StandardSliceGeometry;
} MethodClass;
```

File parsDefinition.h :

```
YesNo parameter
{
  display_name "Sat. Parallel to Slice";
  relations SatParallelRel;

}SatParallel;
```

File initMeth.c

```
(...)
  /*
   *  initialize sat slices module
   */
  SatParallelRange();
  STB_InitSatSlicesModule();
```

File callbackDefs.h

```
/* saturation slices */
relations PVM_SatSlicesHandler backbone;
```

File parsRelations.c

```
void backbone(void)
{
  (...)

  /*
   *  calculate frequency offsets
   */

  LocalFrequencyOffsetRels();

  /*
   *  update saturation slices
   */

  if(PVM_FovSatOnOff==Off)
  {
    ParxRelsHideInEditor("SatParallel");
  }
  else
  {
    ParxRelsShowInEditor("SatParallel");
  }

  if(SatParallel==Yes)
  {
    int i,siz;
    double vec[3];

    memcpy(vec,PVM_SPackArrGradOrient[0][2],3*sizeof(double));
    dim=(int)PARX_get_dim("PVM_FovSatSliceVec",1);
    for(i=0;i<dim;i++)
    {
      memcpy(PVM_FovSatSliceVec[i],vec,3*sizeof(double));
    }
  }
```

```
  STB_UpdateSatSlicesModule(PVM_Nucleus1);

(...)
}

void UpdateRepetitionTime(void)
{
  int nslices, dim;
  double TotalTime,mindur,riset;

  DB_MSG(("-->UpdateRepetitionTime"));
  (...)
  PVM_MinRepetitionTime =
    nslices *
    (
      0.011                                              +
      PVM_FovSatModuleTime                               +
      SliceGradStabTime                                  +
      CFG_GradientRiseTime()                             +
      CFG_AmplifierEnable()                              +
      ExcPulse.Length/2                                  +
      PVM_EchoTime                                       +
      PVM_AcquisitionTime *(1.0 - PVM_EchoPosition/100)+
      ReadSpoilGradDur                                   +
      CFG_InterGradientWaitTime()
    );
(...)
}

void SatParallelRange(void)
{
  if(!ParxRelsParHasValue("SatParallel"))
  {
    SatParallel = No;
  }
  else
  {
    if(SatParallel != Yes && SatParallel != No)
    {
      SatParallel = No;
    }
  }

}


void SatParallelRel(void)
{
  SatParallelRange();
  backbone();
}
```

File BaseLevelRelations.c:

```
void SetBaseLevelParam()
{

  DB_MSG(("-->SetBaseLevelParam"));
  (...)

  ATB_SetSatSlicesBaseLevel();


  PrintTimingInfo();

  DB_MSG(("<--SetBaseLevelParam"));


}
```

2 Creating modules: Tagging (greTag)

2.1 Objectives:

By developing this project you will learn to

- program method elements in a modular manner,
- include a module in an existing method,
- use the Request Handling mechanism for a group of parameters.

2.2 Project description

We will develop a pulse program module for making a periodical pattern of stripes (tags) on the imaging object. Such a module may find application in dynamical studies of moving objects (e.g. in cardiac movies). The module will consist of a pair of identical non-selective RF pulses separated by a gradient pulse on the readout channel. Depending on the position along the readout axis, and on the precession caused by the gradient pulse, the flip angles of the RF pulses will either add or cancel. This will produce a periodic pattern of saturation tags perpendicular to the readout axis with the spacing given by the relation

$$\text{gradient\_amplitude} * \text{gradient\_time} * \text{tag\_spacing} = 1 \qquad [1]$$

with the gradient amplitude expressed in Hz/mm. We will also add a spoiling gradient pulse to disperse the residual transverse magnetisation. The module will be inserted in the gre.ppg pulse program in front of the excitation pulse.

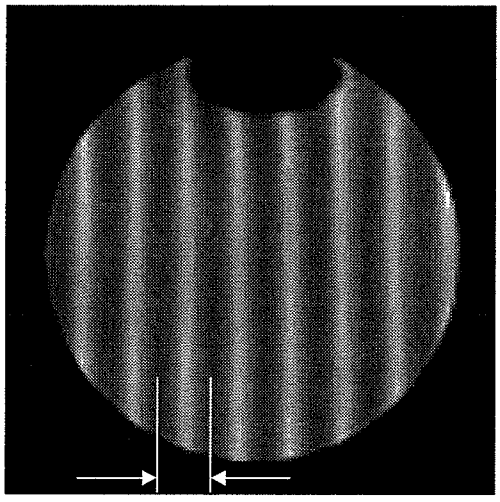


Fig. 1: Definition of Tag_Spacing

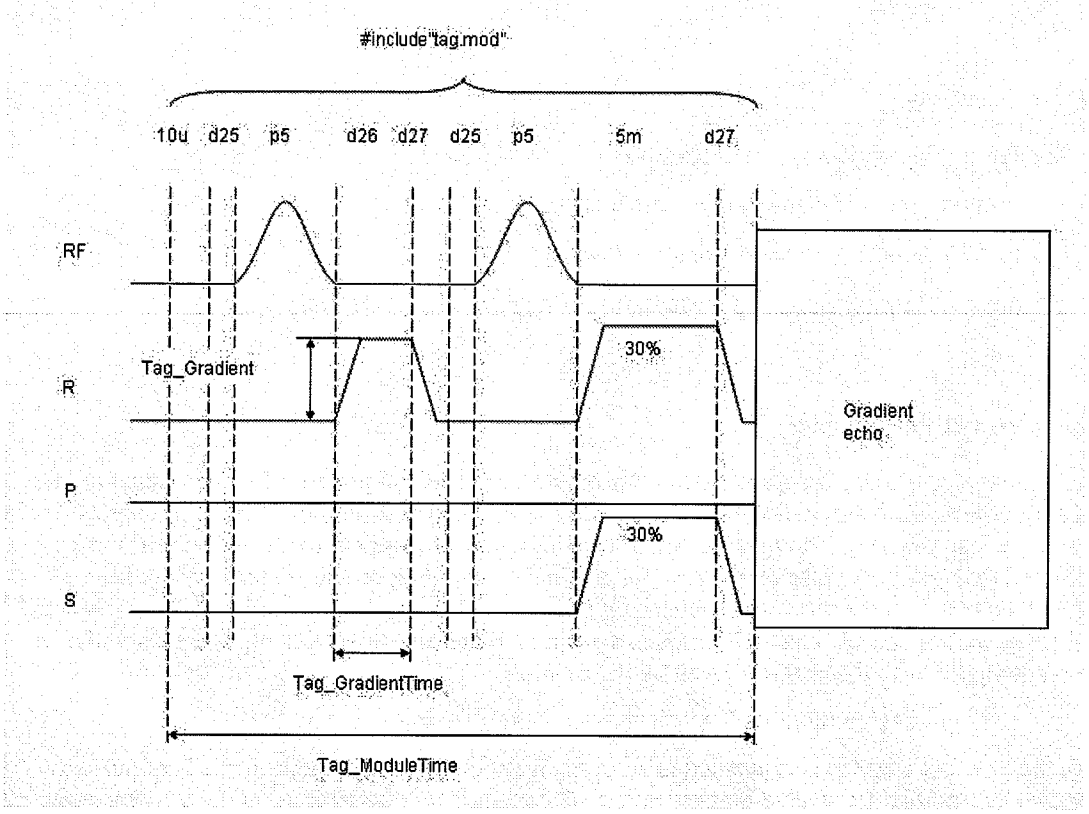


Fig. 2: The tagging module.

The module will be controlled by a dedicated group of PVM parameters handled in the same way as the predefined parameter groups (Diffusion, contrast modules, etc.). This modular design of sequence elements has the advantage of being easy to transfer to different methods.

The parameters describing the gradient pulse and the tag spacing will be inter-related using the request handling mechanism. It will be possible to change the value of any of the three parameters and the method will react by adapting the remaining ones to keep the relation [1].

Your task will consist of modifying the *greTag* method, in which a simple form of the tagging module is already included. You will add a new parameter to the module and change the group behavior. The module will then be transferred to the spinEcho method.

2.3 The greTag method

The greTag method that you can find in the `/opt/PPC/PVM/src` directory already includes a simplified version of the Tagging module. The module allows specifying the gradient duration (*Tag_GradientTime*) and the tag spacing (*Tag_Spacing*) but the gradient amplitude is fixed at 5%. Your task will be to introduce a new parameter for the gradient amplitude and to produce specified changes in the module behavior.

The module is represented by the following files:

- The pulse program section (Tag.mod) to be included in the main ppg,

- TagPars.h containing the definitions of the module parameters,
- TagLayout.h containing the definition of the module parameters class Tagging,
- Tag.c containing the relations of the module parameters as well as the initialization, update and base-level-setting functions.

The *.h files of the module are included in the corresponding *.h of the method. The *.c file is listed in the Makefile. You should remember this arrangement for transferring the module to a different method.

By working with this method you will notice that the parameters *Tag_Spacing* and *Tag_GradientTime* are related. This relation is programmed in the *Tag_Update* function using the ***UT_GetRequest*** function. This function allows checking which parameter has last been edited. The switch statement inside *Tag_Spacing* provides different reactions to the editing of these parameters:

- **Tag_Spacing changed**: Calculate Tag_GradientTime correspondingly: if too short, restrict it and readjust Tag_Spacing.
- **Tag_GradientTime (or any other parameter) changed**: Calculate the corresponding Tag_Spacing.

Note that the UT_GetRequest function detects parameter changes only if the relations of these parameters include calls of ***UT_SetRequest***. You can also use this mechanism to detect edited structure fields and array elements, as described in ParaVision® Advanced Users Manual. This, however is out of scope of our project.

2.4 Specific method parameters

Parameters of the Tagging module:

Tag_RfPulse –	RF pulse used (twice) in the module.
Tag_Spacing –	Tag pattern spacing (period) in mm.
Tag_GradientTime –	Effective duration of the gradient pulse
Tag_Gradient –	Amplitude of the gradient pulse in % of max, to be introduced in this project.
Tag_ModuleTime –	Duration of the tagging module

2.5 Special features

When the tag spacing is set too narrow (below one pixel) the pattern disappears. The depth of the pattern (saturation strength) depends on the flip angle of the tag pulses, which should be typically in the range from 15 to 45 degree.

The value of Tag_GradientTime must not be shorter than the gradient rise time.

The method must prevent zero value for any of the parameters involved in Eq. [1]: all of them are used in denominators in the code!

2.6 Your task

Review the code of the greTag method. In particular, look at the structure of Tag_ parameters relations and at the usage of the request handling in the Tag_Update function.

Modify the greTag method (locally on your computer, not in PPC/PVM) in the following way:

- Add a new parameter *Tag_GradientTime* to control the amplitude of the tag gradient pulse
- Modify the *Tag_Update* function so that it reacts to the to the changes of all three parameters involved in Eq. [1] in the following way:
    - **Tag_Gradient changed**: Calculate *Tag_GradientTime* to keep the current *Tag_Spacing*; if *Tag_GradientTime* becomes too short, restrict it and readjust Tag_Spacing.
    - **Tag_Spacing changed**: Calculate Tag_Gradient to keep the current Tag_Spacing. Check the new value of Tag_Gradient. If too low or too high, restrict it and readjust Tag_Spacing.
    - **Tag_GradientTime (or any other parameter) changed**: Calculate the corresponding Tag_Spacing.
- Transfer the modified tagging module to the spinEcho method (call it spinEchoTag).

2.7 Realisation hints

2.7.1 Starting point

The method should be developed based on the *greTag* method .

2.7.2 Steps

1. Copy *greTag* from the /opt/methPool to your local methods/src directory
2. Add a definition of *Tag_Gradient* in TagPars.h. Declare its relations as *Tag_GradientRels.*
3. Add *Tag_Gradient* to the *Tagging* parclass in TagLayout.h.
4. Write the relations (*Tag_GradientRels*) and range checker (*Tag_GradientRange*) of *Tag_Gradient.* You can take the ones of *TagGradientTime* as a basis. *Tag_Gradient* should be restricted to [0.1 .. 100]. Remember about UT_SetRequest in the relations.
5. Initialise *Tag_Gradient* by calling its range checker in *Tag_Init*().
6. Modify *Tag_Update*():
    - Replace the fixed amplitude (variable gradient) by the new parameter.
    - Add the parameter name to the list used by UT_GetRequest.

- Modify the switch statement: add a case for Tag_Gradient (5, if you have placed it at the end of the list) and change the case for Tag_Spacing, as described in 2.6.

7. Modify Tag_BaseLevel() to set the gradient amplitude based on Tag_Gradient.
8. Test the modified greTag.
9. **Optional**: You can restrict the tag spacing to at least twice the pixel resolution and at most a quarter of the FOV (so that the tagging pattern never disappears). This will require two additional arguments (fov, resolution) in the Tag_Update function. There are different ways for this restriction. You can, for example, adjust the Tag_Gradient to set the Tag_Spacing within limits.
10. Introduce the same module in spinEcho (call it spinEchoTag):
    - copy all Tag.* files to spinEchoTag
    - add Tag$(OBJEXT) to the OBJLIST definition in the Makefile
    - include Tag.mod in the pulse program in front of the excitation.
    - include the Tag*.h files in the corresponding method's files.
    - call Tag_Init() in initMeth()
    - call Tag_Update() in the backbone() – when?
    - call Tag_BaseLevel in SetBaseLevel().

2.8 Proposed solution

File Tag.c (modifications in boldface)

```
/* relations of Tag_Gradient */
void Tag_GradientRels(void)
{
  DB_MSG(("--> Tag_GradientRels()"));

  UT_SetRequest("Tag_Gradient");
  Tag_GradientRange();
  ParxRelsParRelations("Tag_Handler", No);

  DB_MSG(("<-- Tag_GradientRels()"));
}

/* range check of Tag_Gradient */
void Tag_GradientRange(void)
{
  DB_MSG(("--> Tag_GradientRange()"));

  if(ParxRelsParHasValue("Tag_Gradient") == 0)
    Tag_Gradient = 5.0;

  Tag_Gradient = MAX_OF(Tag_Gradient, 0.1);
  Tag_Gradient = MIN_OF(Tag_Gradient, 100.0);

  DB_MSG(("<-- Tag_GradientRange()"));
}
```

```
/* ------------------------------------------------
 * Tag_Init
 * initialisation of the Tag module
 *
 * ------------------------------------------------ */
void Tag_Init(void)
{
  DB_MSG(("--> Tag_InitTagging()"));

  Tag_RfPulseRange();
  Tag_ModuleTimeRange();
  Tag_GradientTimeRange();
  Tag_GradientRange();
  Tag_SpacingRange();

  DB_MSG(("<-- Tag_InitTagging()"));
}


/* ------------------------------------------------
 * Tag_Update
 * Updates the Tag module.
 * Arguments:
 * nucleus: name of the nucleus (PVM_Nucleus1)
 * deriveGains - if Yes, he gains for tagging pulses
 *               will be derived.
 *-----------------------------------------------*/
void Tag_Update(char *nucleus, YesNo deriveGains)
{

  YesNo  referenceAviable;
  double referenceAttenuation=30;
  YesNo  requestDetected;
  int    dummy, parN;
  double gcc; /* gradient removed */
  const char *parList =
    "Tag_Spacing,"
    "Tag_GradientTime,"
    "Tag_RfPulse,"
    "Tag_ModuleTime,"
    "Tag_Gradient";


   DB_MSG(("--> Tag_Update()"));


  /* gradient calibration const, for later use */

  gcc = CFG_GradCalConst(nucleus);


  /* update the RF pulse */

  if(deriveGains == Yes)
    referenceAviable =
      CFG_GetGlobRefAtt(nucleus,&referenceAttenuation);
  else
    referenceAviable = No;
```

```
  STB_UpdateRFPulse("Tag_RfPulse",
                    &Tag_RfPulse,
                    referenceAviable,
                    referenceAttenuation);

  /* react to requests */
  /* we do not test structure fields, therefore "" and dummy: */
  requestDetected = UT_GetRequest(parList, &parN, "", &dummy);


  if(requestDetected == No)
    parN = 0;  /* Lack of request is treated as default; see switch below */

  DB_MSG(("request: %d",parN));

  switch(parN)
  {

    case 1:  /* Tag_Spacing requested: Derive Tag_Gradient */
      Tag_Gradient = 1e5/(Tag_Spacing * Tag_GradientTime * gcc);
      if(Tag_Gradient < 0.1)
      {
        Tag_Gradient = 0.1;
        Tag_Spacing = 1e5 / (Tag_GradientTime * Tag_Gradient * gcc);
      }
      else if(Tag_Gradient > 100.0)
      {
        Tag_Gradient = 100.0;
        Tag_Spacing = 1e5 / (Tag_GradientTime * Tag_Gradient * gcc);
      }
      break;

    case 5: /* Tag_Gradient requested: Derive Tag_GradientTime */
      Tag_GradientTime = 1e5/(Tag_Spacing * Tag_Gradient * gcc);
      if(Tag_GradientTime < CFG_GradientRiseTime())
      {
        Tag_GradientTime =  CFG_GradientRiseTime();
        Tag_Spacing = 1e5 / (Tag_GradientTime * Tag_Gradient * gcc);
      }
      break;

    case 2:  /* Tag_GradientTime requested, */
    case 0:  /* no request -                */
    default: /*   - derive Tag_Spacing:     */
      Tag_Spacing = 1e5 / (Tag_GradientTime * Tag_Gradient * gcc);
  }

  /* module duration */
  Tag_ModuleTime =
    0.01                      +
    2 * Tag_RfPulse.Length    +
    2 * CFG_AmplifierEnable() +
    Tag_GradientTime          +      /* tag gradient pulse */
    CFG_GradientRiseTime()    +
    5.0                       +      /* spoiler */
    CFG_GradientRiseTime();

  DB_MSG(("<-- Tag_Update()"));
}


/*---------------------------------------------------------------------
```

```
 * Tag_BaseLevel
 * Sets ACQP parameters used by the Tag module
 *-----------------------------------------------------------------*/
void Tag_BaseLevel(void)
{

  /* delays */
  D[25] = CFG_AmplifierEnable()/1000.0;
  D[26] = Tag_GradientTime/1000.0;
  D[27] = CFG_GradientRiseTime()/1000.0;

  /* gradient amplitude */
  PVM_ppgGradAmp5 = Tag_Gradient/100.0;

  /* rf pulse length */
  P[5] = Tag_RfPulse.Length*1000.0;

  /* rf pulse */
  strcpy(TPQQ[5].name, Tag_RfPulse.Filename);
  if(PVM_DeriveGains == Yes)
    TPQQ[5].power =  Tag_RfPulse.Attenuation;
  TPQQ[5].offset = 0.0;

  /* frequency list */
  ACQ_O2_list_size = 1;
  ParxRelsParRelations("ACQ_O2_list_size",Yes);
  ACQ_O2_list[0] = 0.0;

}
```

3 Accelerated Imaging (encGre)

3.1 Objectives

By developing this project you will learn how to

- implement a new parameter group called *Encoding*.

This new parameter group can handle all common encoding schemes including Parallel Imaging (PI), linear and centric k-space sampling, block-wise and interleaved segmentation, partial Fourier encoding and zero filling. Its purpose is to allow the user specify the required encoding scheme and to calculate the corresponding acquisition matrix size as well as the table of phase encoding steps.

With implementation of the *Encoding* group different mechanisms for accelerated imaging are provided.

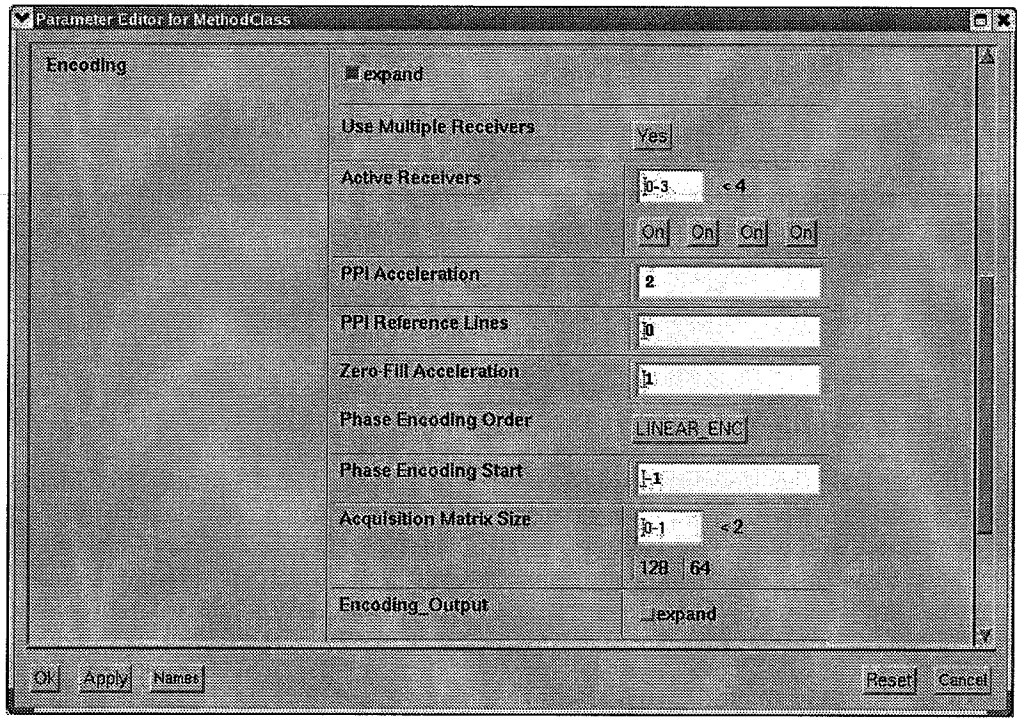
3.2 Project description

We will start from the *gre* sequence and create a new method *encGre* using copyMethod. You will learn to perform the necessary steps to make the new parameter group visible in the encGre method with correct initialisation of all *Encoding* parameters. Afterwards you will learn how to update the Encoding parameters and how to set all relevant base-level parameters for controlling the phase encoding and the reconstruction of multi-channel acquisitions.

For experienced programmers there exists an additional exercise described in chapter 3.6. In this exercise you will use the implemented *Encoding* group to control the phase encoding schema of a segmented acquisition. For this purpose *PVM_RareFactor* must be activated in the method and must be used as an update of *segment size* in the *Encoding* group.

3.3 Specific method parameters

3.3.1 New parameter group: Encoding



- **Use Multiple Receivers (PVM_EncUseMultiRec)** – A Yes/No parameter deciding whether the acquisition should work on a single or multiple channels. It can be switched to Yes only if the system contains multiple receive channels.
- **Active Receivers (PVM_EncActiveReceivers)** – An array of On/Off parameters to activate/deactivate the data acquisition on selected channels.
- **PPI Acceleration (PVM_EncPpiAccel1)** – An integer defining the acceleration by means of Parallel Imaging. This parameter introduces an undersampling of the k-space, i.e., it removes a periodic pattern of steps from the encoding scheme. The maximum value of PPI Acceleration is given by the number of active channels. Parallel Imaging preserves the resolution of the image but reduces the SNR by at least the square root of PVM_EncPpiAccel1.
- **PPI Reference Lines (PVM_EncPpiRefLInes1)** – Number of nominally sampled lines in the k-space centre to be taken disregarding the PPI acceleration. The reference lines are needed for the auto-calibrating PI reconstruction methods, like the one currently implemented in ParaVision.
- **Zero-Fill Acceleration (PVM_EncZfAccel1)** – A number telling how much the experiment is accelerated by a symmetric truncation of the k-space. Zero-filling should be used with caution since it reduces the effective resolution of the image (causes blurring). A similar parameter exists for the 3rd spatial dimension (Zero-Fill Acceleration 3d, PVM_EncZfAccel2).

- **Phase Encoding Order (PVM_EncOrder1)** – A symbol taking values ENC_LINEAR or ENC_CENTRIC and determining the order of acquisition of k-space lines (from edge to edge, or from centre outwards, respectively). The centric order allows achieving minimum effective echo times in multiple spin-echo methods such as RARE. It can also increase the signal-to-noise ratio in fast gradient echo methods (e.g. FISP) by effectively using the approach to the steady state. However, the centric scheme usually introduces a blurring effect.
- **Phase Encoding Start (PVM_EncStart1)** – A number indicating the starting value of the phase encoding, ranging from -1 (lower edge of k-space) to +1 (upper edge). When the encoding start is higher than -1, the k-space is sampled cyclically. This parameter may be used to shorten the effective echo time in segmented experiments, such as RARE. This parameter is not available with the centric encoding and with partial-FT. A similar parameter exists for the third dimension (**Phase Encoding Start 3D**)
- **Acquisition Matrix Size (PVM_EncMatrix).** The effective size of the acquisition matrix taking into account the geometry parameters and all acceleration methods. Non-editable. When all acceleration parameters are set to 1.0 (no acceleration), the acquisition matrix size is a product of the image size (PVM_Matrix, see In-plane geometry) and the anti-aliasing factor (PVM_AntiAlias). Selecting one or more acceleration schemes (parallel, partial-FT, zero-filling) causes the acquisition matrix size to be reduced.
- **Encoding_Output.** This subclass contains non-editable information parameters calculated by the Encoding mechanism.

3.4 Realization hints

3.4.1 Starting point

The method should be developed based on the *gre* method.

3.4.2 Steps

1. Use copyMethod to copy *gre* to *encGre*.
2. Like any other parameter group, *Encoding* must be included in the definition of MethodClass in the file **parsLayout.h** to be visible in the protocol.
3. In the **callbackDefs.h** file the group handler must be directed to the method's *backbone* function.
   - relations PVM_EncodingHandler backbone;
4. The group should also be initialized in **initMeth.c** by call of
   - STB_InitEncoding();
5. In the *backbone* function in **parsRelations.c** the *Encoding* group must be updated:

```
• STB_UpdateEncoding(PTB_GetSpatDim(),  /* dimension */
                     PVM_Matrix,        /* image size */
                     PVM_AntiAlias,     /* a-alias */
                     &seg_size,         /* segment size */
                     SEG_SEQUENTIAL,    /* segmenting mode */
                     Yes,             /* PI in 2nd dim allowed */
                     Yes,  /* PI ref lines in 2nd dim allowed */
                     No);/* partial ft in 2nd dim NOT allowed */
```

Note that this update requires information about the dimension, image size and anti-aliasing and should therefore be made after the update of the in-plane geometry. The fourth argument of STB_UpdateEncoding is a pointer to a parameter defining the segment size in a multi-shot experiment, which typically is the RARE-factor (PVM_RareFactor). This parameter may change as a result of this function call. The next argument specifies the way of k-space segmenting. SEG_SEQUENTIAL means that consecutive scans should sample adjacent blocks of k-space lines. SEG_INTERLEAVED used for this argument would cause the consecutive k-space scans to be interleaved as in the RARE method. The final Yes/No arguments tell the group to activate or deactivate the possibility of PI, reference lines for PI, and partial (asymmetric) Fourier encoding.

Following the update, all parameters of the Encoding group have legal and consistent values.

6. Use *PVM_EncMatrix* for experiment time calculation in **parsRelations.c**, because PVM_EncMatrix describes the acquisition matrix and is defined as product of image size and anti-aliasing factor:

   - TotalTime = PVM_EncMatrix[1] * PVM_RepetitionTime * PVM_NAverages;

7. Use *PVM_EncMatrix* for setting *ACQ_size* in **BaseLevelRelations.c**

   - ATB_SetAcqSize(Spatial, spatDim, PVM_EncMatrix, NULL, No);

8. Set ACQP parameters for *User Encoding* in **BaseLevelRelations.c**

   - Use the switch statement: *switch(spatDim){....}*

9. Setting base-level parameters for controlling a multiple receiver acquisition:

   This is done in the **BaseLevelRelations.c** file using the toolbox function *ATB_SetMultiRec()*. If this function returns *Yes*, a special 4-channel version of the pulse program needs to be declared. This takes place on older versions of the spectrometer hardware. Starting from Avance II this function always returns *No* and no special pulse program is needed.

   - If(Yes == ATB_SetMultiRec())

     ATB_SetPulprog("encGre.4ch");

10. Multi receiver data require a special reconstruction mode which is defined in the **RecoRelations.c**:

    - To make things easier, copy RecoRelations.c from the master solution (/opt/PPC/PVM/src/encGre/) into your method directory.

3.5 Proposed Solution

parsLayout.h

```
parclass
{
  Method;
  PVM_EchoTime;
  PVM_MinEchoTime;
  PVM_RepetitionTime;
  PVM_NAverages;
  PVM_ScanTimeStr;
  PVM_DeriveGains;
  RF_Pulses;
  Nuclei;
  Encoding;
  SequenceDetails;
  StandardInplaneGeometry;
  StandardSliceGeometry;
} MethodClass;
```

callbackDefs.h

```
/* ************************************************************
 * redirection of global parameter groups
 * ************************************************************/

/* Encoding */
relations PVM_EncodingHandler backbone;

/* inplane geometry */
relations PVM_InplaneGeometryHandler  InplaneGeometryRel;
```

initMeth.c

```
  STB_InitStandardInplaneGeoPars(2,dimRange,lowMat,upMat,No);

  /*
   *  init slice geometry parameters
   */
  STB_InitSliceGeoPars(0,0,0);

  /*
   *  init encoding parameters
  */
  STB_InitEncoding();

  /*
   * init spectroscopy parameters (no csi)
   */
```

```
PTB_SetSpectrocopyDims( 0, 0 );
```

parsRelations.c

```
STB_UpdateStandardInplaneGeoPars(minFov,2);

/* calculate gradients in logical directions */

if(dim == 3)
{
  /* Connect slice thickness to FOV in 3rd direction. */
  PVM_SliceThick = PVM_Fov[2];

  /* constrain maximum slices per package to 1 */
  STB_UpdateSliceGeoPars(0,0,1,minSliceThick);
}
else
{
  /* no constrain to slices in 2D mode */
  STB_UpdateSliceGeoPars(0,0,0,minSliceThick);
}
FreqEncodingGradients(PVM_Fov[0],readGradRatio,PVM_GradCalConst);
UpdatePhase2DGradients(PVM_Matrix[1],PVM_Fov[1],PVM_GradCalConst);

if(dim == 3)
{
  UpdatePhase3DGradients(PVM_Matrix[2],PVM_Fov[2],PVM_GradCalConst);
}
else
{
  Phase3DGrad =0.0;
}

/* once the image size is decided (after first update of inpl geo),
we can update the Encoding parameters, to get the acquisition size
(PVM_EncMatrix). */

int seg_size;
seg_size = 1;

STB_UpdateEncoding(PTB_GetSpatDim(),  /* total dimensions */
                PVM_Matrix,       /* image size */
                PVM_AntiAlias,    /* a-alias */
                &seg_size,        /* segment size */
                SEG_SEQUENTIAL,   /* segmenting mode */
                Yes,              /* ppi in 2nd dim allowed */
                Yes,              /* ppi ref lines in 2nd dim allowed*/
                No);              /* partial ft in 2nd dim allowed */

UpdateSliceSelectionGradients(PVM_SliceThick,
                              sliceGradRatio,
                              ExcPulse.Bandwidth*BwScale/100.0,
                      PVM_GradCalConst);
```

```
(...)

PVM_RepetitionTime=MAX_OF(PVM_MinRepetitionTime,PVM_RepetitionTime);

dim = PTB_GetSpatDim();

TotalTime = PVM_RepetitionTime
          * PVM_EncMatrix[1]
          * PVM_NAverages;

if(dim == 3)
{
  TotalTime *= PVM_EncMatrix[2];
}

UT_ScanTimeStr(PVM_ScanTimeStr,TotalTime);
ParxRelsShowInEditor("PVM_ScanTimeStr");
ParxRelsMakeNonEditable("PVM_ScanTimeStr");
```

BaseLevelRelations.c

```
SetMachineParameters();

if( PVM_ErrorDetected == Yes )
{
  UT_ReportError("SetBaseLevelParam: In function call!");
  return;
}

/* ------------------------------------------------------------------
    Sets parameters needed for multi-receiver acq. Overrides some
    previously set parameters such as NUCn Must be called at the end
    of SetBaseLevel.
   ------------------------------------------------------------------ */
if(Yes==ATB_SetMultiRec())
{
  ATB_SetPulprog("encGre.4ch");
}


PrintTimingInfo();

DB_MSG(("<--SetBaseLevelParam"));
```

(...)

```
/* ACQ_dim_desc */

ATB_SetAcqDimDesc( specDim, spatDim, NULL );
if( PVM_ErrorDetected == Yes )
{
  UT_ReportError("SetBasicParameters: In function call!");
  return;
```

```
  }

  /* ACQ_size */
  /* With the Encoding group, this call
     ATB_SetAcqSize(Spatial, spatDim, PVM_Matrix, PVM_AntiAlias, No);
     is replaced by: */
  ATB_SetAcqSize( Spatial, spatDim, PVM_EncMatrix, NULL, No );

  if( PVM_ErrorDetected == Yes )
  {
    UT_ReportError("SetBasicParameters: In function call!");
    return;
  }
```

(...)

```
void SetGradientParameters( void )
{
  int spatDim, dim;

  DB_MSG(("-->SetGradientParameters"));


  ATB_SetAcqPhaseFactor( 1 );
  if( PVM_ErrorDetected == Yes )
  {
    UT_ReportError("SetGradientParameters: In function call!");
    return;
  }

  spatDim = PTB_GetSpatDim();

  dim = PARX_get_dim("ACQ_phase_encoding_mode", 1 );
  PARX_change_dims("ACQ_phase_encoding_mode", spatDim );
  PARX_change_dims("ACQ_phase_enc_start", spatDim );
  switch(spatDim)
  {
    case 3:
      ACQ_phase_encoding_mode[2] = User_Defined_Encoding;
      ACQ_phase_enc_start[2] = -1; /* set, but no used */
      ACQ_spatial_size_2 = PVM_EncMatrix[2];
      ParxRelsCopyPar("ACQ_spatial_phase_2","PVM_EncValues2");
      /* no break */
    case 2:
      ACQ_phase_encoding_mode[1] = User_Defined_Encoding;;
      ACQ_phase_enc_start[1] = -1.0; /* set, but no used */
      ACQ_spatial_size_1 = PVM_EncMatrix[1];
      ParxRelsCopyPar("ACQ_spatial_phase_1","PVM_EncValues1");
      /* no break */
    default:
      ACQ_phase_encoding_mode[0] = Read;
      ACQ_phase_enc_start[0] = -1;
  }
```

```
ATB_SetAcqGradMatrix( PVM_NSPacks, PVM_SPackArrNSlices,
                PtrType3x3 PVM_SPackArrGradOrient[0],
                PVM_ObjOrderList );

if( PVM_ErrorDetected == Yes )
{
  UT_ReportError("SetGradientParameters: In function call!");
  return;
}
```

RecoRelations.c

```
int dim,i;
int size,ftSize[3];

DB_MSG(("-->SetRecoParam"));

/* set baselevel reconstruction parameter */
/* default initialization of reco based on acqp pars allready set */

ATB_InitDefaultReco();

for(i=0; i<PTB_GetSpatDim(); i++)
  ftSize[i] = PVM_Matrix[i]*PVM_AntiAlias[i];

if(PVM_EncUseMultiRec == Yes)
{
 /* select method specific reconstruction method */
 RECO_mode = USER_MODE;
 ParxRelsParRelations("RECO_mode",Yes);
 ATB_InitUserModeReco(ACQ_dim, PVM_EncMatrix, ftSize, PVM_EncSteps1,
                PVM_EncSteps2,PVM_EncNReceivers, PVM_EncPpiAccel1,
                PVM_EncPpiRefLines1, NI, ACQ_obj_order,
                ACQ_phase_factor, PVM_EchoPosition);

}


/* set reco rotate according to phase offsets     */
dim = PTB_GetSpatDim();


/* set reco sizes and ft_mode (dim 2&3) */
/* (dim 1 is kept as it was set by ATB_InitDefaultReco) */
for(i=1; i<dim; i++)
{
  size = PVM_Matrix[i]*PVM_AntiAlias[i];
  RECO_ft_mode[i] = (size == PowerOfTwo(size)) ?
                    COMPLEX_FFT:COMPLEX_FT;
  RECO_ft_size[i] = size;
  RECO_size[i] = PVM_Matrix[i];
}

ParxRelsParRelations("RECO_ft_mode",Yes);
```

```
  ParxRelsParRelations("RECO_ft_size",Yes);
  ParxRelsParRelations("RECO_size",Yes);


  ATB_SetRecoRotate(PVM_EffPhase1Offset,
                    PVM_Fov[1]*PVM_AntiAlias[1],
                    NSLICES,
                    PVM_NEchoImages,
                    1) ;          /* phase1 direction*/

(...)

  ATB_SetRecoTranspositionFromLoops(PtrType3x3 ACQ_grad_matrix[0],
                              NSLICES,
                              1,
                              NI,
                              ACQ_obj_order);

  RECO_bc_mode[0] = AUTO_OFFSET_BC;

  DB_MSG(("<--SetRecoParam"));

}

int PowerOfTwo(int x)
/* returns next power of two */
{

 return (1<<(int)ceil(log((double)x)/log(2.0)));
}
```

3.6 Additional Exercise: Segmented acquisition

3.6.1 Objectives

- Use *PVM_RareFactor* to define the segment size in the Encoding group.

The purpose of this exercise is to demonstrate the functionality of the new introduced *Encoding* group: only a few steps are necessary in order to implement the functionality of a segmented acquisition.

3.6.2 Project description

We will use *PVM_RareFactor* as an argument of the update of the Encoding group. The fourth argument of *STB_UpdateEncoding* is a pointer to a parameter defining the segment size in a multi-shot experiment, which typically is the *PVM_RareFactor*.

To use *PVM_RareFactor* in encGre, we have to make this parameter visible in parsLayout.h, to initialize it in initMeth.c, to use it as an update of the Encoding group in parsRelations.c and some base-level parameters must be set in BaseLevelRelations.c.

A new parameter *Seg_DelayTime* will be defined in order to control segment repetition time.

3.6.3 Realization hints

1. Include *PVM_RareFactor* in the definition of MethodClass in the file **parsLayout.h** to be visible in the protocol.
2. Initialize *PVM_RareFactor* in **initMeth.c**:
   - if(ParxRelsParHasValue("PVM_RareFactor") == No)

     PVM_RareFactor = 8;
3. In **callbackDefs.h** the relation of PVM_RareFactor must be redirected:
   - relations PVM_RareFactor backbone;
4. In **parsRelations.c** the Encoding group must be updated with a pointer argument of PVM_RareFactor as segment size. Change the segmenting mode from SEG_SEQUENTIAL to SEQ_INTERLEAVED.
5. Define a new parameter *Seg_DelayTime* in **parsDefinition.h** and make this parameter visible in the MethodClass in the file **parsLayout.h**.
6. *Seg_DelayTime* should be restricted in **parsRelations.c**:
   - Seg_DelayTime = MAX_OF(PVM_RepetitionTime, Seg_DelayTime);
7. Set base-level parameters in the file **BaseLevelRelations.c**:
   - ATB_SetAcqPhaseFactor(PVM_RareFactor);
   - ACQ_rare_factor = PVM_RareFactor;
   - L[0] = ACQ_size[1]/PVM_RareFactor;
   - L[4] = PVM_RareFactor;
   - D[8] = (Seg_DelayTime – PVM_RepetitionTime)/1000.0;
8. Add *Seg_DelayTime d8* and *loop l4* to the pulse program **encGre.ppg**.
9. Incorporate the *Seg_DelayTime* into the calculation of the estimated total scan time (*PVM_ScanTimeStr*) in **parsRelations.c**.

3.6.4 Proposed Solution

parsLayout.h

```
parclass
{
  Method;
  PVM_EchoTime;
  PVM_MinEchoTime;
```

```
  PVM_RepetitionTime;
  Seg_DelayTime;
  PVM_NAverages;
  PVM_ScanTimeStr;
  PVM_DeriveGains;
  RF_Pulses;
  Nuclei;
  Encoding;
  PVM_RareFactor;
  SequenceDetails;
  StandardInplaneGeometry;
  StandardSliceGeometry;
} MethodClass;
```

initMeth.c

```
  STB_InitEncoding();

  if(ParxRelsParHasValue("PVM_RareFactor") == No)
    PVM_RareFactor = 8;

  /*
   * init spectroscopy parameters (no csi)
   */
```

callbackDefs.h

```
/* ************************************************************
 *
 * relation redirection of single global parameters used in
 * this method:
 *
 * ************************************************************/

relations PVM_RepetitionTime          RepetitionTimeRel;
relations PVM_NAverages               AveragesRel;
relations PVM_EchoTime                EchoTimeRel;
relations PVM_MinEchoTime             backbone;

relations PVM_AcquisitionTime         backbone;
relations PVM_DeriveGains             DeriveGainsRel;
relations PVM_EffSWh                  EffSWhRel;
relations PVM_RareFactor              backbone;
```

parsRelations.c

```
  /* once the image size is decided (after first update of inpl geo),
  we can update the Encoding parameters, to get the acquisition size
  (PVM_EncMatrix). */

  STB_UpdateEncoding(PTB_GetSpatDim(),  /* total dimensions */
                PVM_Matrix,        /* image size */
```

```
                PVM_AntiAlias,      /* a-alias */
                &PVM_RareFactor,    /* segment size */
                SEG_INTERLEAVED,    /* segmenting mode */
                Yes,                /* ppi in 2nd dim allowed */
                Yes,              /* ppi ref lines in 2nd dim allowed */
                No);                /* partial ft in 2nd dim allowed */
```

(...)

```
PVM_RepetitionTime=MAX_OF(PVM_MinRepetitionTime,PVM_RepetitionTime);

Seg_DelayTime = MAX_OF(PVM_RepetitionTime, Seg_DelayTime);

dim = PTB_GetSpatDim();

TotalTime = PVM_RepetitionTime
          * PVM_EncMatrix[1]
          * PVM_NAverages;
```

parsDefinition.h

```
int parameter
{
  display_name "Number of dummy scans";
  relations dsRelations;
} NDummyScans;

double parameter
{
  display_name "Segment Delay Time";
  units        "ms";
  format       "%.2f";
  relations    backbone;
} Seg_DelayTime;
```

BaseLevelRelations.c

```
void SetGradientParameters( void )
{
  int spatDim, dim;

  DB_MSG(("-->SetGradientParameters"));


  ATB_SetAcqPhaseFactor( PVM_RareFactor );
  if( PVM_ErrorDetected == Yes )
  {
    UT_ReportError("SetGradientParameters: In function call!");
    return;
  }
```

(...)

```
  ACQ_scaling_read  = 1.0;
  ACQ_scaling_phase = 1.0;
  ACQ_scaling_slice = 1.0;

  ACQ_rare_factor = PVM_RareFactor;

  ACQ_grad_str_X = 0.0;
  ACQ_grad_str_Y = 0.0;
  ACQ_grad_str_Z = 0.0;
```

(...)

```
  sprintf(PULPROG,"encGre.ppg");

  L[0] = ACQ_size[1]/PVM_RareFactor;
  L[4] = PVM_RareFactor;

  if( PTB_GetSpatDim() == 3)
  {
    L[1] = ACQ_size[2];
  }
  else
  {
    L[1] = 1;
  }

  slices = GTB_NumberOfSlices( PVM_NSPacks, PVM_SPackArrNSlices );
  igwT   = CFG_InterGradientWaitTime();
  riseT  = CFG_GradientRiseTime();


  D[0]  = ((PVM_RepetitionTime - PVM_MinRepetitionTime)/slices
            + igwT)/1000.0;
  D[1]  = (SliceGradStabTime + riseT)/1000.0;
  D[2]  = CFG_AmplifierEnable()/1000.0;
  D[3]  = (Phase2DGradDur - riseT) / 1000.0;
  D[5]  = (PVM_EchoTime - PVM_MinEchoTime + riseT + igwT)/1000.0;


  D[4]  = (riseT + PVM_AcqStartWaitTime)/1000.0;
  D[6]  = (ReadSpoilGradDur - Phase2DGradDur)/1000.0;
  D[7] = riseT/1000.0;
  D[8] = (Seg_DelayTime - PVM_RepetitionTime)/ 1000.0;
```

encGre.ppg

```
;============================================================
;   definition of delays
;============================================================

define delay denab
"denab =  d4 - de + depa"

"l3 = l0 + ds"
"l8 = l4 * ds"
```

```
;=============================================================
;   declaration of 2d and 3d loop
;=============================================================


lgrad r2d<2d> = ACQ_size[1]
zgrad r2d
lgrad r3d<3d> = L[1]
zgrad r3d


if(DS>0)
{
  dsl,  dgrad r2d
  lo to dsl times l8

}

(...)
lo to start times NSLICES
      1u            ipp1                    zslice
lo to start times NA
      1u                rpp1                    igrad r2d
lo to start times l3
      1u                                        igrad r3d
      d8                  ;Seg_DelayTime
lo to start times l4      ;PVM_RareFactor
lo to start times l1
lo to start times NAE
SETUP_GOTO(start)
exit
```

Pipeline Filter

ParaVision Programming Course
April 03-07, 2006

Arno Nauerth

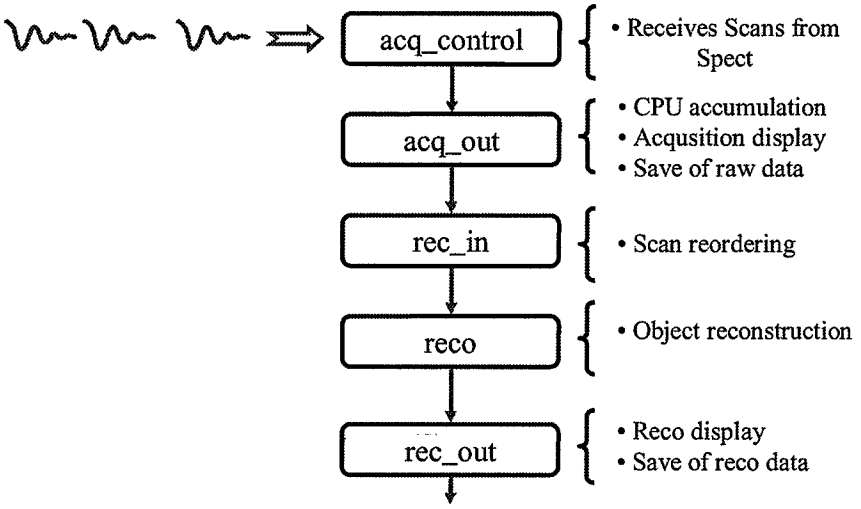
Pipeline Filter

- Introduction
- Overview of tutorial Examples
- Basics
- Examples: Step by Step
- Discussions

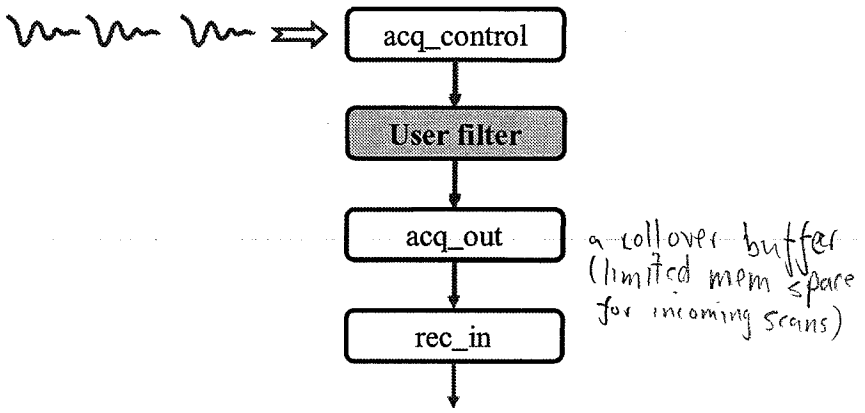
Introduction: Usage

- Integration of an user program into an acquisition pipeline
- Sophisticated, user-defined raw data manipulations, such as
  - K-space corrections
  - K-space weighted accumulation
  - Keyhole imaging
  - Navigator techniques
  - ...

Introduction: Acquisition Pipeline



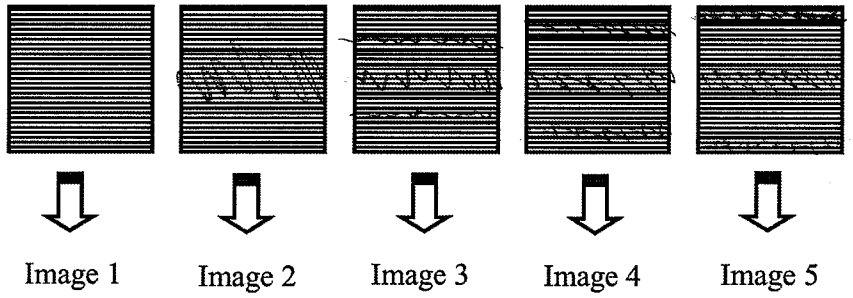
Introduction: User Filter Program



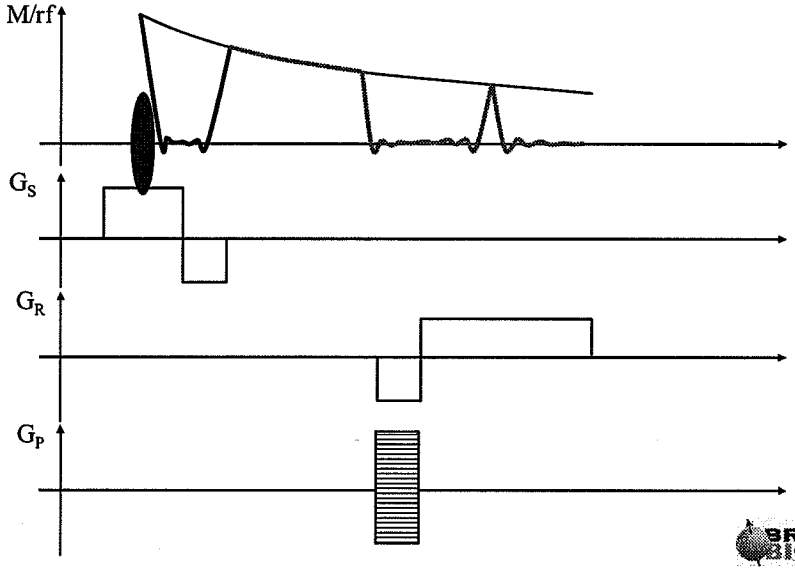
Overview: Examples

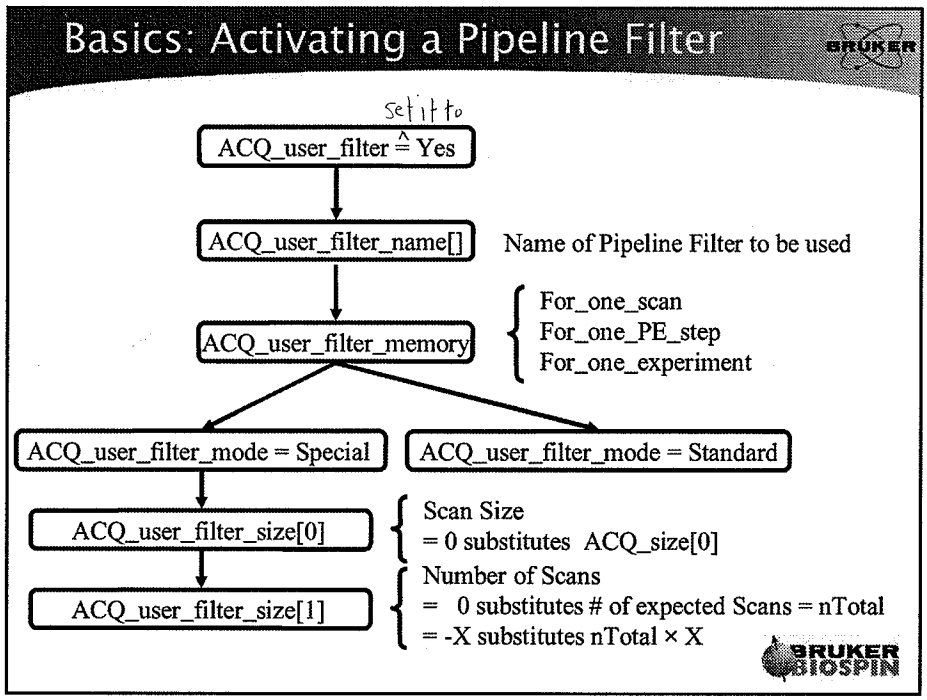
- NoOperation – pass data without processing
- NoOpFast – optimized NoOperation
- SyncScan – Synchronized Experiment Start
- CartToPolar – Cartesian to Polar conversion of Scans
- KeyHole – Sequence acceleration
- MagControl – Navigator Corrected Magnetization

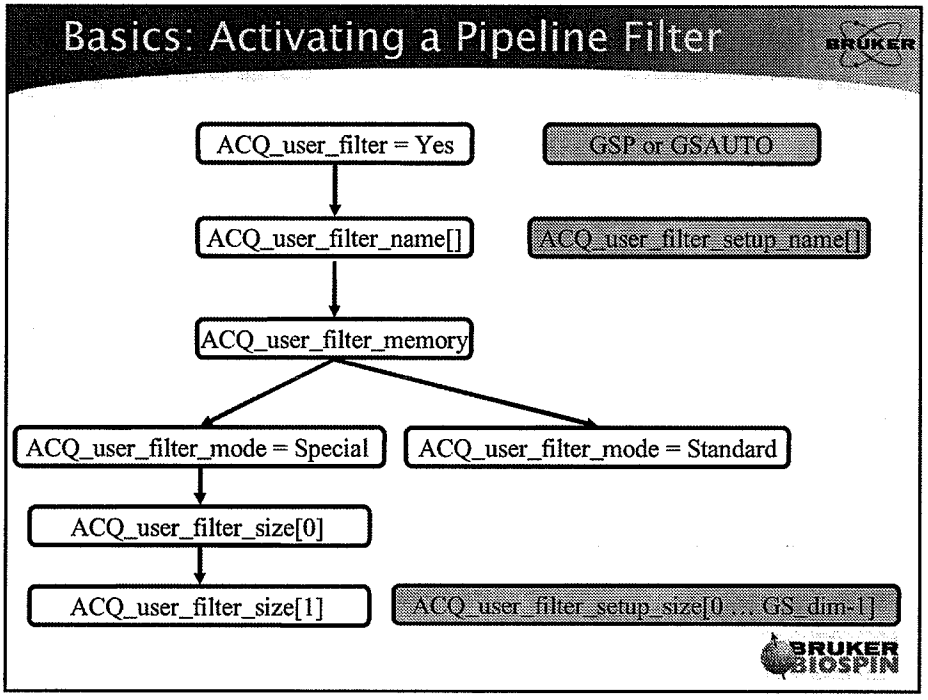
Pipeline Filter: KeyHole



Pipeline Filter: MagControl




Basics: Activating a Pipeline Filter
BRUKER
set it to
ACQ_user_filter = Yes
ACQ_user_filter_name[]
Name of Pipeline Filter to be used
ACQ_user_filter_memory
For_one_scan
For_one_PE_step
For_one_experiment
ACQ_user_filter_mode = Special
ACQ_user_filter_mode = Standard
ACQ_user_filter_size[0]
Scan Size
= 0 substitutes ACQ_size[0]
ACQ_user_filter_size[1]
Number of Scans
= 0 substitutes # of expected Scans = nTotal
= -X substitutes nTotal × X
BRUKER BIOSPIN

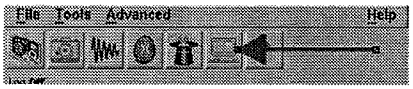

Basics: Activating a Pipeline Filter
BRUKER
ACQ_user_filter = Yes
GSP or GSAUTO
ACQ_user_filter_name[]
ACQ_user_filter_setup_name[]
ACQ_user_filter_memory
ACQ_user_filter_mode = Special
ACQ_user_filter_mode = Standard
ACQ_user_filter_size[0]
ACQ_user_filter_size[1]
ACQ_user_filter_setup_size[0 .. GS_dim-1]
BRUKER BIOSPIN

Basics: General Form of a Pipeline Filter

Declarations (optional)
PIPE_INIT
PARX_INIT
Declarations (optional)
Program code (optional)
PIPE_PROC
Declarations (optional)
Program code (required)
PIPE_WRAPUP
Program code (optional)
PIPE_End

Pipeline Filter: NoOperation

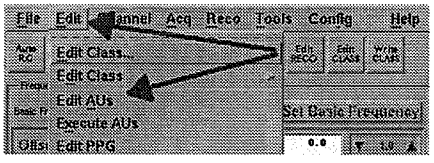
Open a ParaVision® Shell



Enter

cd $XWINNMRHOME/exp/stan/nmr/au/src
cp /opt/PPC/AU/src/NoOperation .

Edit AU NoOperation



Pipeline Filter: NoOperation

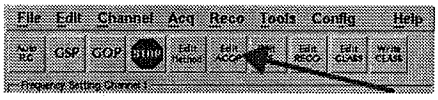
```
PIPE_INIT
PARX_INIT
PIPE_PROC
int scan_cnt;
int dummy;
while(1)
{
    for (scan_cnt=0; scan_cnt < scans_per_exp;scan_cnt++)
    {
      PIPE_WAIT_INPUT_OUTPUT(1,1,dummy,dummy)
      memcpy(out_ptr, in_ptr, sizeof_scan);
      PIPE_SIGNAL_OUTPUT(1,1);
    }
    if (acq_scan_type == Scan_Experiment) break;
    scans_in = 0;
} /* while (1) */
PIPE_WRAPUP
PIPE_END
```

Exit the Editor and let it be compiled

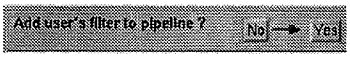
Pipeline Filter: NoOperation

Load a Single Slice Gradient Echo Sequence with short TR

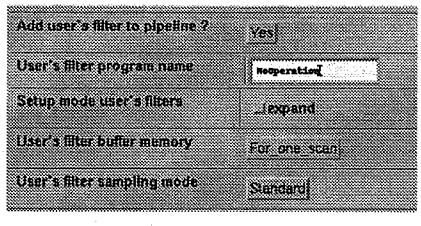
Edit ACQP Parameters



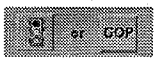
Enable Pipeline Filters



Activate NoOperation



Run Acquisition



Pipeline Filter: NoOperation

Add a print statement to NoOperation

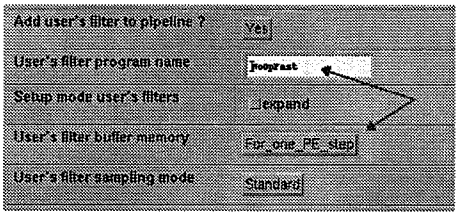
```
PIPE_INIT
PARX_INIT
PIPE_PROC
int scan_cnt;
int dummy;
while(1)
{
    for (scan_cnt=0; scan_cnt < scans_per_exp;scan_cnt++)
    {
      PIPE_WAIT_INPUT_OUTPUT(1,1,dummy,dummy)
      memcpy(out_ptr, in_ptr, sizeof_scan);
      PIPE_SIGNAL_OUTPUT(1,1);
      printf("%d scans transferred\n",scan_cnt);
    }
    if (acq_scan_type == Scan_Experiment) break;
    scans_in = 0;
} /* while (1) */
PIPE_WRAPUP
PIPE_END
```

Pipeline Filter: NoOpFast

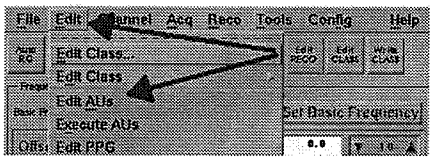
Enter within the ParaVision® Shell

{ cd $XWINNMRHOME/exp/stan/nmr/au/src }
cp /opt/PPC/AU/src/NoOpFast .

Activate NoOpFast



Edit AU NoOpFast



Pipeline Filter: NoOpFast

```
PIPE_INIT
PARX_INIT
PIPE_PROC
int scan_cnt;
int inCnt,outCnt,useCnt;
while(1)
{
  for (scan_cnt=0; scan_cnt < scans_per_exp;)
  {
    PIPE_WAIT_INPUT_OUTPUT(1,1,inCnt,outCnt)
    useCnt = MIN_OF(inCnt,outCnt);
    memcpy(out_ptr, in_ptr, sizeof_scan*useCnt);
    PIPE_SIGNAL_OUTPUT(useCnt,useCnt);
    scan_cnt += useCnt;
  }
  if (acq_scan_type == Scan_Experiment) break;
  scans_in = 0;
} /* while (1) */
PIPE_WRAPUP
PIPE_END
```

#scans input

#scans passed to output (ACQ - out)

faster b/c if more than 1 scan available, can transfer more than 1 scan.

Pipeline Filter: NoOpFast

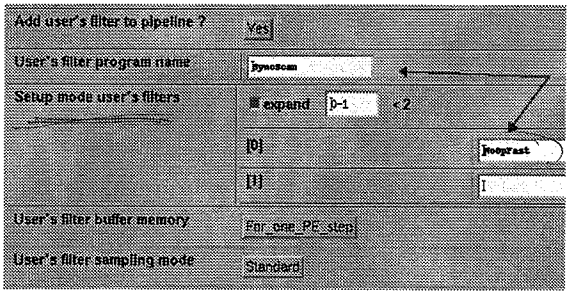

```
FILE *fdTty=NULL;
PIPE_INIT
fdTty = fopen("/dev/tty","w");
if (NULL == fdTty) EXCEPT_printf("cannot fopen /dev/tty");
PARX_INIT
PIPE_PROC
int scan_cnt;
int inCnt,outCnt,useCnt;
while(1)
{
  for (scan_cnt=0; scan_cnt < scans_per_exp;)
  {
    PIPE_WAIT_INPUT_OUTPUT(1,1,inCnt,outCnt)
    useCnt = MIN_OF(inCnt,outCnt);
    memcpy(out_ptr, in_ptr, sizeof_scan*useCnt);
    PIPE_SIGNAL_OUTPUT(useCnt,useCnt);
    scan_cnt += useCnt;
    fprintf(fdTty,"in=%d, out=%d, use=%d\n",inCnt,outCnt,
useCnt);
  }
  if (acq_scan_type == Scan_Experiment) break;
  scans_in = 0;
} /* while (1) */
PIPE_WRAPUP
if (NULL != fdTty) fclose(fdTty);
fdTty = NULL;
PIPE_END
```

Pipeline Filter: SyncScan

Enter within the ParaVision® Shell

```
cp /opt/PPC/AU/src/SyncScan .
pvWish
wm withdraw .
tk_messageBox –message Hello
exit
```

Activate NoOpFast for Setup an SyncScan for Scan experiments



← traffic light RXnotj. won't do SyncScan. will do NoOpFast

Pipeline Filter: SyncScan

```
const char sysStr[] = {"echo \"wm withdraw .;tk_messageBox -message \
                \\\"Start Scan\\\" -type ok -icon info;exit\"|pvWish"};

PIPE_INIT
PARX_INIT

system((const char *)sysStr);

PIPE_PROC
...
PIPE_WRAPUP
PIPE_END
```

prepares all pipelines

command prompt shell commands.

Use GSP and GDP

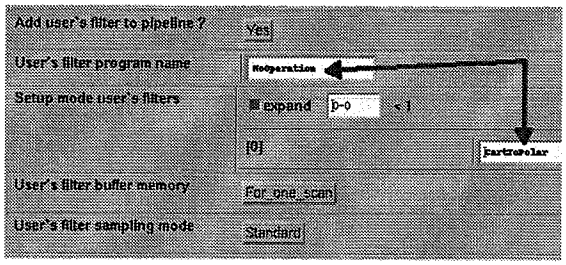
Pipeline Filter: CartToPolar

Load SETUP_SHIM_bas protocol, location B_BASIC_PVM

Enter within the ParaVision® Shell

cp /opt/PPC/AU/src/CartToPolar .

Activate CartToPolar for Setup Experiments



Pipeline Filter: CartToPolar

```
double scale;
int dr;
PIPE_INIT
dr = GETIPAR("DR");
scale = (1<<(dr-1))/M_PI;
PARX_INIT
PIPE_PROC
int               scan_cnt, dummy, *ip, *op;
int               mag_int, phase_int;
double   Re, Im;
while (1)
{
  for (scan_cnt=0; scan_cnt < scans_per_exp; scan_cnt++)
  {
    PIPE_WAIT_INPUT_OUTPUT(1,1,dummy,dummy)
    ip = (int *) in_ptr;
    op = (int *) out_ptr;
    for (i1=0; i1 < scan_word_size/2 ; i1++)
    {
      Re = *ip++ ;
      Im = *ip++ ;
      mag_int   = sqrt(Re*Re + Im*Im);
      phase_int = scale * atan2(Im,Re);
      *op++ = mag_int;
      *op++ = phase_int;
    }
    PIPE_SIGNAL_OUTPUT(1,1);
  }
  if (acq_scan_type == Scan_Experiment) break;
  scans_in = 0;
} /* while (1) */
        PIPE_WRAPUP
PIPE_END
```

complex #

useful for, preemph adj.

mag → adj grad. preemph

phase → adj B_0 comp.

shift Scan by = -1 to remove adj filter

Pipeline Filter: KeyHole

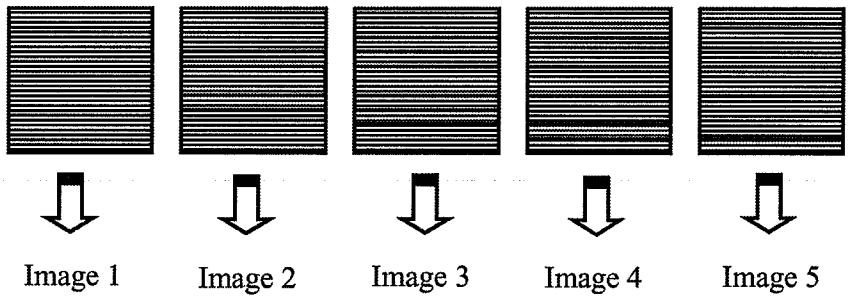
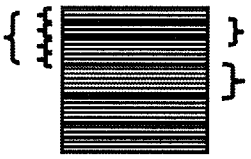


Image 1 Image 2 Image 3 Image 4 Image 5

KeyHole: Parameters



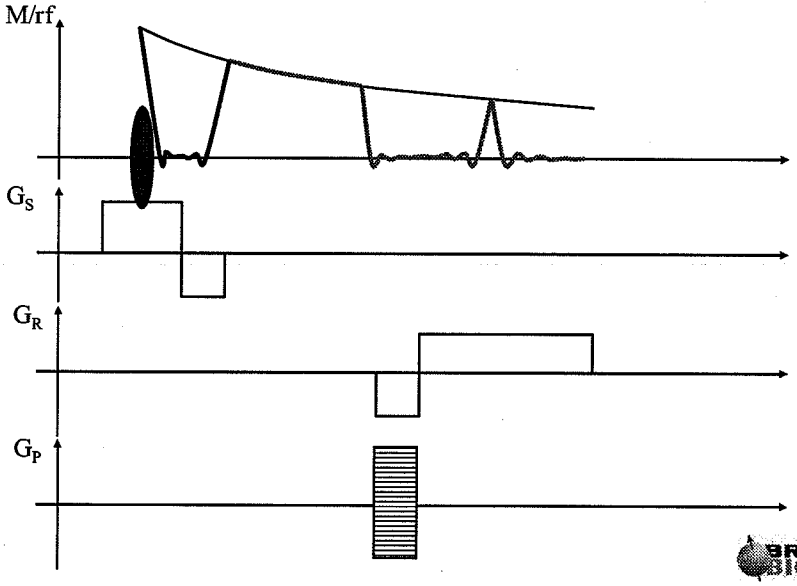
NumberOfSegments **SegmentSize** **CenterProjections**

ArrayIndices[] specifies array position for each Scan

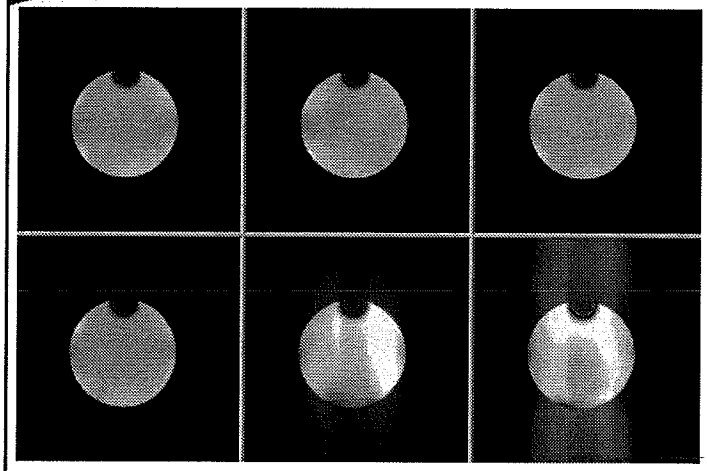
ACQ_spatial_phase_1[] specifies the K-space position for each Scan

arraySize = acq_size[1] +
(CenterProjections + NumberOfSegments) * SegmentSize * 2

Pipeline Filter: MagControl



MagControl: Results



MagControl[led]

uncontrolled

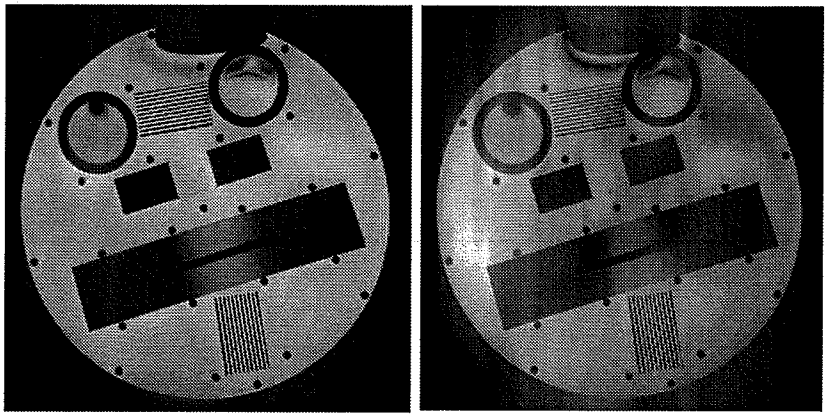
ACQ Start at K = -1

ACQ Start at K = 0

Flip-Angle 90 deg

MagControl: Results

ACQ Start at K = 0, Flip Angle 30 deg



MagControl[led]

uncontrolled

Pipeline Filter

ParaVision Programming Course
April 03-07, 2006

Author:

Arno Nauerth

Bruker BioSpin MRI GmbH

Table of Content

Pipeline Filter

1 NoOperation

Pipeline Filter which passes through all Scans without any processing of them.

1.1 Objectives

The purpose of this Pipeline Filter is to

- understand the basic principle of a Pipeline Filter
- have a simple template for the development of a new Pipeline Filter which does neither a data reduction nor a data expansion
- have a no-operation pipeline filter wherever it is needed

1.2 Implementation

1.2.1 Parameters and their Settings

- ACQ_user_filter = Yes
- ACQ_user_filter_name = ACQ_user_filter_setup_name = NoOperation
- ACQ_user_filter_memory = For_one_scan
- ACQ_user_filter_mode = Standard

1.2.2 Realization Hints

- The predefined integer variable **scans_per_exp** is initialized to the total number of scans expected by the pipeline filters succeeding the user filter
- **PIPE_WAIT_INPUT_OUTPUT(x1, x2, y1, y2)** – Wait for at least **x1** scans to be available in the input buffer, where at least **y1** scans are contiguous, and wait until there is space in the output buffer for at least **x2** scans, where at least **y2** scans are contiguous.
- After the return from PIPE_WAIT_INPUT_OUTPUT(), the predefined char pointers **in_ptr** and **out_ptr** are pointing to the input buffer and the output buffer respectively
- **PIPE_SIGNAL_OUTPUT(x1,x2)** – Signal the system that **x1** input scans and **x2** output scans have been processed
- **void *memcpy(void *dest, const void *src, size_t n)** – copy **n** bytes from memory area **src** to memory area **dest**

1.2.3 Creating the Pipeline Filter NoOperation

Use the Menu function ***Edit AUs*** of the ***Spectrometer Control Tool*** in order to create and compile the program to be executable within the acquisition pipeline environment.

```
PIPE_INIT
PARX_INIT
PIPE_PROC
int scan_cnt;
int dummy;
while(1)
{
    for (scan_cnt=0; scan_cnt < scans_per_exp; scan_cnt++)
    {
      PIPE_WAIT_INPUT_OUTPUT(1,1,dummy,dummy)
      memcpy(out_ptr, in_ptr, sizeof_scan);
      PIPE_SIGNAL_OUTPUT(1,1);
    }
    if (acq_scan_type == Scan_Experiment) break;
} /* while (1) */
PIPE_WRAPUP
PIPE_END
```

The source of NoOperation will be found within the directory $XWINNMRHOME/exp/stan/nmr/au/src. During the development of a Pipeline Filter, it is often more efficient to have the source code permanently open with an arbitrary editor. When doing so, any change to the file needs to be manually compiled, e.g. by

```
$XWINNMRHOME/exp/stan/nmr/au/makeau NoOperation
```

1.3 Exercises

- comment out the memcpy() function
- add printf() statements for test purposes

2 NoOpFast

Variation of the Pipeline Filter NoOperation for maximum throughput

2.1 Objectives

The purpose of this Pipeline Filter is to

- optimize the data throughput for experiments with an very high effective sampling rate
- have a simple template for an optimized data throughput

- have an efficient no-operation pipeline filter wherever it is needed

2.2 Implementation

2.2.1 Parameters and their Settings

- ACQ_user_filter = Yes
- ACQ_user_filter_name = ACQ_user_filter_setup_name = NoOpFast
- ACQ_user_filter_memory = {For_one_PE_step, For_one_Experiment}
- ACQ_user_filter_mode = Standard

2.2.2 Realization Hints

The macro ***MIN_OF(a,b)*** returns the minimum value of the two operands ***a*** and ***b***.

2.2.3 The Pipeline Filter NoOpFast

The modified lines to NoOperation are printed in bold face:

```
PIPE_INIT
PARX_INIT
PIPE_PROC
int scan_cnt;
int inCnt,outCnt,useCnt;
while(1)
{
  for (scan_cnt=0; scan_cnt < scans_per_exp;)
  {
    PIPE_WAIT_INPUT_OUTPUT(1,1,inCnt,outCnt)
    useCnt = MIN_OF(inCnt,outCnt);
    memcpy(out_ptr, in_ptr, sizeof_scan*useCnt);
    PIPE_SIGNAL_OUTPUT(useCnt,useCnt);
    scan_cnt += useCnt;
  }
  if (acq_scan_type == Scan_Experiment) break;
} /* while (1) */
PIPE_WRAPUP
PIPE_END
```

2.3 Exercises

- use **FILE *fdTty, fdTty=fopen("/dev/tty","w")** and **fprintf(fdTty,"...")** in order to get a kind of real-time debug information within the ParaVision startup terminal window
- modify sequence in order to have a high effective sampling rate
- set NI > 1 and ACQ_user_filter_size = For_one_PE_step

3 SyncScan

Pipeline Filter which raises a top-level window to control the start of the acquisition (modified NoOpFast)

3.1 Objectives

The purpose of this Pipeline Filter is to

- understand the synchronization of the initialization- proceccing- and wrapup sections of Pipeline Filters respectively
- usage of **system()** commands within the different sections

3.2 Implementation

3.2.1 Parameters and their Settings

- ACQ_user_filter = Yes
- ACQ_user_filter_name = ACQ_user_filter_setup_name = SyncScan
- ACQ_user_filter_memory = x
- ACQ_user_filter_mode = Standard

3.2.2 Realization Hints

- **system(const char *string)** – executes a command in **string** by calling /bin/sh –c **string**
- **pvWish** – TclTk window shell for *ParaVision*®
  - tk_messageBox – display a toplevel message box within pvWish
  - enter the follwing commands within a *ParaVision*® shell

```
pvWish
wm withdraw .
tk_messageBox -message {Hello World}
exit
```

3.2.3 The Pipeline Filter SyncScan

```
const char sysStr[] = {"echo \"wm withdraw .;tk_messageBox -message \
                \\\"Start Scan\\\" -type ok -icon
info;exit\"|pvWish"};

PIPE_INIT
PARX_INIT

system((const char *)sysStr);
PIPE_PROC
int scan_cnt;
int inCnt,outCnt,useCnt;
int *ip, *op;
while(1)
{
  for (scan_cnt=0; scan_cnt < scans_per_exp;)
  {
    PIPE_WAIT_INPUT_OUTPUT(1,1,inCnt,outCnt)
    useCnt = MIN_OF(inCnt,outCnt);
    memcpy(out_ptr, in_ptr, sizeof_scan*useCnt);
    PIPE_SIGNAL_OUTPUT(useCnt,useCnt);
    scan_cnt += useCnt;
  }
  if (acq_scan_type == Scan_Experiment) break;
  scans_in = 0;
} /* while (1) */
PIPE_WRAPUP
PIPE_END
```

3.3 Exercises

- Add a system call to signalize the end of the experiment
- What does it mean to put a system call outside the *while (1) {...}* body but before the *PIPE_WRAPUP* statement?

4 CartToPolar

Reorganize raw data from Cartesian Real/Imaginary pairs into Magnitude/Phase pairs.

4.1 Objectives

The purpose of this Pipeline Filter is to

- get a better idea of the power of Pipeline Filters
- have a PREEMPHASIS tool available for separating gradient from B0 eddy current effects

4.2 Implementation

4.2.1 Parameters and their Settings

- ACQ_user_filter = Yes
- ACQ_user_filter_name = ACQ_user_filter_setup_name = CartToPolar
- ACQ_user_filter_memory = x
- ACQ_user_filter_mode = Standard

4.2.2 Realization Hints

- **M_PI** is a constant definition for π
- **GETIPAR("parName")** – returns the value of the PARX integer parameter **parName**
- **double atan2(double Y, double X)** – calculates the arc tangent of the two variables **X** and **Y**

4.2.3 The Pipeline Filter CartToPolar

```
double scale;
int dr;

PIPE_INIT
dr = GETIPAR("DR");
scale = (1<<(dr-1))/M_PI;
PARX_INIT
PIPE_PROC
int       scan_cnt, dummy, *ip, *op;
int       mag_int, phase_int;
double    Re, Im;

while (1)
{
  for (scan_cnt=0; scan_cnt < scans_per_exp; scan_cnt++)
  {
    PIPE_WAIT_INPUT_OUTPUT(1,1,dummy,dummy)
    ip = (int *) in_ptr;
    op = (int *) out_ptr;
    for (i1=0; i1 < scan_word_size/2 ; i1++)
    {
      Re = *ip++ ;
      Im = *ip++ ;
      mag_int   = sqrt(Re*Re + Im*Im);
      phase_int = scale * atan2(Im,Re);
      *op++ = mag_int;
      *op++ = phase_int;
    }
    PIPE_SIGNAL_OUTPUT(1,1);
  }
  if (acq_scan_type == Scan_Experiment) break;
```

```
  scans_in = 0;
} /* while (1) */
    PIPE_WRAPUP
PIPE_END
```

4.3 Exercises

- Remove the scaling by the variable **scale**

5 KeyHole Imaging

Measuring method keyHole using a Pipeline Filter for speeding up an imaging sequence

5.1 Objectives

The purpose of this Pipeline Filter is to

- understand the pipeline mechanism for data reduction or data expansion

↳ needed to fill in the kspace lines that we kept from the previous scans (i.e. Centre projections)

5.2 Implementation

5.2.1 Parameters and their Settings

- ACQ_user_filter = Yes
- ACQ_user_filter_name = ACQ_user_filter_setup_name = KeyHoleAu
- ACQ_user_filter_memory = x
- ACQ_user_filter_mode = Special
- ACQ_user_filter_size[0] = ACQ_size[0]
- ACQ_user_filter_size[1] < scans_per_exp

5.2.2 Realization Hints

- **ACQ_user_filter_size[0] = 0** is a substitution for **ACQ_size[0]**
- **ACQ_phase_encoding_mode[1] = User_Defined_Encoding** can be used to have a user defined **r2d** phase encoding function
  - **ACQ_spatial_size_1** defines the size of the gradient function **r2d**
  - **ACQ_spatial_phase_1[]** contains the phase encoding values for **r2d**
- Apply the command **copyMethod** within a *ParaVision*® shell in order to create a private measuring method
- The environment variable **ParxMethodSearchPath** can be used to invoke private user-created measuring method
- **PIPE_WAIT_OUTPUT(x,y)** Wait for at least **x** scans to be available in the output bffer, where at least **y** scans are contiguous

5.2.3 Creating the Measuring Method keyhole

5.2.3.1 Using copyMethod to copy gre to keyHole

```
>copyMethod ($XWINNMRHOME/prog/service/copyMethod)
 Enter name of source method : gre
    Enter name of target method : keyHole
    Copy source method e.g. to /home/$USER/pv302/methods
>cd /home/$USER/pv302/methods/keyHole
```

5.2.3.2 *Modification of the Pule Program*

There is no modification of the pulse program required when using **ACQ_phase_encoding_mode[1] = User_Defined_Encoding.**

5.2.3.3 *Externsion to parsDefinition.h*

```
/*****************************************************************
 *  PARAMETER DEFENITION FOR KEY-HOLE
 *****************************************************************/
int parameter
{
  display_name "nuber of segments";
  relations KeyHoleRelations;
}NumberOfSegments;
int parameter
{
  display_name "segment size";
  relations KeyHoleRelations;
}SegmentSize;
int parameter
{
  display_name "number of center projections";
  relations KeyHoleRelations;
} CenterProjections;
int parameter
{
  display_name "array indices";
} ArrayIndices[];
```

5.2.3.4 *Externsion to parsLayout.h*

```
/*---------------------------------------------------------------*
 * PV class...Extension for KeyHole
 *---------------------------------------------------------------*/

parclass
{
  NumberOfSegments;
  CenterProjections;
  SegmentSize;
  ArrayIndices;
}KeyHole;

parclass
{
  Method;
  KeyHole;
...
} MethodClass;
```

5.2.3.5 *Extension of initMeth.c*

```
int dimRange[2] = {2,2}; //limit experiment to two dimensions
```

5.2.3.6 *Extension of parsRelations.c*

```
/*:=MPB=:=========================================================*
 *
 * Global Function:  KeyHoleRelations()
 * Description:    calculate gradient- and index array for Keyhole
 *                 imaging.
 * Preconditons:  - linear phase encoding, starting at -1.0    */

void  KeyHoleRelations ( void )

/*:=MPE=:=========================================================*/
{
  int i,j, idx;
  int arraySize;
  int matrixSize = ACQ_size[1];

  DB_MSG(("-->KeyHoleRelations"));
  if (NumberOfSegments < 1)
    NumberOfSegments = 2;
  if (NumberOfSegments >= matrixSize/4)
    NumberOfSegments = matrixSize/4;
  if (CenterProjections >= matrixSize/4)
    CenterProjections = matrixSize / 4;
  if (CenterProjections < 2)
    CenterProjections = matrixSize / 8;
  CenterProjections = (CenterProjections + 1) /2 * 2;
  NumberOfSegments  = (NumberOfSegments  + 1) /2 * 2;
  SegmentSize = (matrixSize - CenterProjections) / NumberOfSegments;
  CenterProjections = matrixSize - SegmentSize * NumberOfSegments;
  arraySize = matrixSize +
              (CenterProjections + SegmentSize) * NumberOfSegments;
  NR = NumberOfSegments + 1;
  ParxRelsParRelations("NR", Yes);
  PARX_change_dims("ArrayIndices",arraySize);
  ACQ_phase_encoding_mode[1] = User_Defined_Encoding;
  ParxRelsParRelations("ACQ_phase_encoding_mode", Yes);
  ACQ_spatial_size_1 = L[0] = arraySize;
  ParxRelsParRelations("ACQ_spatial_size_1", Yes);
  ACQ_user_filter = Yes;
  ParxRelsParRelations("ACQ_user_filter", Yes);
  ACQ_user_filter_mode = Special;
  ParxRelsParRelations("ACQ_user_filter_mode", Yes);
  ACQ_user_filter_memory = For_one_scan;
  sprintf(ACQ_user_filter_name,"KeyHoleAu");
  ACQ_user_filter_size[0] = 0;
  ACQ_user_filter_size[1] = arraySize;
  sprintf(ACQ_user_filter_setup_name,"NoOperation");
```

```
  ACQ_user_filter_setup_size[0] = 1;
  GO_reco_each_nr = Yes;

  for (i=0; i<matrixSize; i++)
  {
    ArrayIndices[i] = i - matrixSize/2;
  }
  idx = matrixSize;
  for (i=0; i<NumberOfSegments; i++)
  {
    for (j=0; j<SegmentSize/2; j++)
    {
      ArrayIndices[idx++] = j - CenterProjections/2 -
(i+1)*SegmentSize/2;
    }
    for (j=0; j<CenterProjections; j++)
    {
      ArrayIndices[idx++] = - CenterProjections/2 + j;
    }
    for (j=0; j<SegmentSize/2; j++)
    {
      ArrayIndices[idx++] = j + CenterProjections/2 +
(i)*SegmentSize/2;
    }
  }
  for (i=0; i<arraySize; i++)
    ACQ_spatial_phase_1[i] = 2.0 * ArrayIndices[i] / matrixSize;
  DB_MSG(("<--KeyHoleRelations"));
}
...
void backbone( void )
{
...
  // make sure to  get NI set to 1 just to make it fast
  STB_UpdateSliceGeoPars(1,1,1,minSliceThick);
...
 KeyHoleRelations();
}
```

5.2.3.7 *The Pipeline Filter KeyHoleAu*

```
#define DEBUG
#ifdef DEBUG
#define DB(P1)        fprintf(fdTty,"%s",P1); (void)fflush(fdTty);
#define DB2(P1,P2)    fprintf(fdTty,P1,P2);   (void)fflush(fdTty);
#define DB3(P1,P2,P3) fprintf(fdTty,P1,P2,P3);(void)fflush(fdTty);
#else
#define DB(P1)
#define DB2(P1,P2)
#define DB3(P1,P2,P3)
#endif

/* global variables */
```

```
ACQ_SCAN_TYPE scan_type;
int *Image;
int *ArrayIndices;
int NumberOfArrayIndices;
int CenterProjections;
int SegmentSize;

#ifdef DEBUG
FILE *fdTty=NULL;
#endif

PIPE_INIT
PARX_INIT
{
#ifdef DEBUG
  fdTty = fopen("/dev/tty","w");
#endif

  /*-------------------------------------------------------*/
  /* allocate memory for image buffer and the array indices */
  /*-------------------------------------------------------*/
  DB2("allocating %d bytes for image buffer\n",scan_byte_size *
acq_size[1]);
  Image = malloc(scan_byte_size * acq_size[1]);
  if (NULL == Image)
    EXCEPT_printf("Cannot allocate memory for image buffer");
  NumberOfArrayIndices =
PARX_get_nth_dim(PARX_psid,"ArrayIndices",1);
  DB2("allocaling %d integers for array
indices\n",NumberOfArrayIndices);
  ArrayIndices = malloc(NumberOfArrayIndices * sizeof(int));
  if (NULL == ArrayIndices)
    EXCEPT_printf("Cannot allocate memory for array indices");

  /*-----------------------------------------------*/
  /* retreive some parameters from parameter space */
  /*-----------------------------------------------*/
  PARX_get_all_values(PARX_psid,
                "ArrayIndices",             ArrayIndices,
                "CenterProjections", &CenterProjections,
                "SegmentSize",              &SegmentSize,
                "ACQ_scan_type",            &scan_type,
                NULL_PTR );
}
PIPE_PROC

int scanCnt;
int outCnt;
int idxImageIn;
int idxImageOut;
int dummy,pipeOutCnt;
idxImageOut = 0;
```

```
for (scanCnt=0; scanCnt <NumberOfArrayIndices ; scanCnt++)
{
  idxImageIn = *(ArrayIndices + scanCnt) + acq_size[1] / 2;
  outCnt = idxImageIn - idxImageOut;
  if (0 < outCnt)
  {
    /*---------------------------------------------------------------*/
     DB3("\n transfer %d scans from 1st image at
%d\n",outCnt,idxImageOut);
    /*---------------------------------------------------------------*/
    do
    {
      PIPE_WAIT_OUTPUT(outCnt,pipeOutCnt);
      memcpy(out_ptr,
           (char *)(Image + idxImageOut * acq_size[0]) ,
           pipeOutCnt * scan_byte_size);
           PIPE_SIGNAL_OUTPUT(0,pipeOutCnt);
      idxImageOut += pipeOutCnt;
      outCnt -= pipeOutCnt;
    } while (outCnt);
  }
  /*---------------------------------------------------------------*/
  /* wait for next scan from acq_control and pass it to next filter
*/
  /*---------------------------------------------------------------*/
  DB2(" %d ",idxImageOut);
  PIPE_WAIT_INPUT_OUTPUT(1,1,dummy,dummy);
  memcpy((char *)(Image + idxImageOut*acq_size[0]),in_ptr
,scan_byte_size);
   memcpy(out_ptr,(char *)(Image + idxImageOut*acq_size[0]),scan_byte_size);
  PIPE_SIGNAL_OUTPUT(1,1);
  idxImageOut++;
  if (scanCnt < (NumberOfArrayIndices - 1))
  {
    if (idxImageIn > *(ArrayIndices + scanCnt + 1) + acq_size[1] / 2)
    {
      outCnt = acq_size[1] - idxImageIn - 1;
      if (outCnt > 0)
      {
        /*-----------------------------------------------------------*/
        DB3("\n transfer %d scans from 1st image at %d
\n",outCnt,idxImageOut);
        /*-----------------------------------------------------------*/
        do
        {
          PIPE_WAIT_OUTPUT(outCnt,pipeOutCnt);
          memcpy(out_ptr,
              (char *)(Image + idxImageOut*acq_size[0]) ,
               pipeOutCnt * scan_byte_size);
          PIPE_SIGNAL_OUTPUT(0,pipeOutCnt);
            idxImageOut += pipeOutCnt;
            outCnt -= pipeOutCnt;
        } while (outCnt);
      }
```

```
      idxImageOut = 0;
      /*-------------------------------------*/
      DB("\n **** Image transferred **** \n");
      /*-------------------------------------*/
    }
  }
}
PIPE_WRAPUP
If (NULL != fdTty)
{
  /*-----------------------------------------*/
  DB("\n ****  Experiment Completed **** \n");
  /*-----------------------------------------*/
  (void)fclose(fdTty);
  fdTty = NULL;
}
PIPE_END
```

5.2.3.8 *Making keyHole ready to use*

```
make clean
make cproto
make depend
make install

cp keyhole.ppg $XWINNMRHOME/exp/stan/nmr/lists/pp
cp KeyHoleAu $XWINNMRHOME/exp/stan/nmr/au/src
$XWINNMRHOME/exp/stan/nmr/au/makeau KeyHoleAu
```

5.3 Exercises

- Setup and run keyhole experiments with different keyhole- setting

6 Magnetization Control

Measuring method greMc using a Pipeline Filter for the online correction of scans having different magnetization amplitudes due to

- ***variable TR's (e.g. due to ECG- gating)***
- ***non-steady-state (e.g. DS = 0)***
- ***variable RF power***

6.1 Objectives

The purpose of this Pipeline Filter is to

- understand the pipeline mechanism for data reduction or data expansion
- implement Navigator techniques by the help of a Pipeline Filter

6.2 Implementation

6.2.1 Parameters and their Settings

- ACQ_user_filter = Yes
- ACQ_user_filter_name = ACQ_user_filter_setup_name = MagControl
- ACQ_user_filter_memory = x
- ACQ_user_filter_mode = Special
- ACQ_user_filter_size[0] > ACQ_size[0]
- ACQ_user_filter_size[1] = 0

6.2.2 Realization Hints

- The number of complex data points acquired during a delay is given by the equation **nx = SW_h * delay**
- **NI** is copied to the predefined parameter **ni**
- **ACQ_size[]** is copied into predefined array parameter **acq_size[]**
- The array parameter **filter_size[]** is adapted to **ACQ_user_filter_size[]**
- **ACQ_scan_type** is copied to the predefined parameter **acq_scan_type**
- **ACQ_scan_type** may have the values **Scan_Experiment** or **Setup_Experiment** during the execution time of an acquisition
- Use the construct

  If (ACQ_scan_type == Setup_Experiment)

  { ...

  }

  in order to have a special pulse program section for a setup experiment.

6.2.3 Creating the measuring method greMc

6.2.3.1 Using copyMethod to copy gre to greMC

```
>copyMethod ($XWINNMRHOME/prog/service/copyMethod)
 Enter name of source method : gre
    Enter name of target method : greMc
    Copy source method e.g. to  /home/$USER/pv302/methods
>cd /home/$USER/pv302/methods/greMc
```

6.2.3.2 Modification of the Pulse Program

Oiginal ppg section:

```
...
        (p0:sp0 ph1):f1
    d3                                  grad{(t2)| r2d(t3)|(t1)+r3d(t4)}
    d5                                  groff
    denab               REC_ENABLE      grad{(t5)| (0) | (0)}
                        ADC_INIT_B(ph1, ph0)
    aqq                 ADC_START
    d3                                   grad{(t8)| r2d(t6) | r3d(t7) }

    d6                                   grad{(t8)| (0) | (0) }
    d7                                   groff
    d0                  ADC_END
...
```

Modified ppg section

```
define delay aqScan
"aqScan = aqq - d5 - d3 - d4"
...
        (p0:sp0 ph1):f1
    d3                                   grad{(0)| (0)|(t1)+r3d(t4)}
    d3                  REC_ENABLE       groff
                        ADC_INIT_B(ph1, ph0)
    d5                  ADC_START
    d3                                   grad{(t2)| r2d(t3)|(0)}
    denab                                grad{(t5)| (0) | (0)}
    aqScan
    d3                                   grad{(t8)| r2d(t6) | r3d(t7)}

    d6                                   grad{(t8)| (0) | (0) }
    d7                                   groff
    d0                  ADC_END
    if (ACQ_scan_type == Setup_Experiment)
    {
          3s             ; additional T1 recovery for Setup and Auto_RG
    }
    ...
```

The scan time has been extended by d3+d5+d4, since denab is just a compensation for d4 for the hidden delay de-depa.

6.2.3.3 *Extensions to parsDefinition.h and parsLayout.h*

None.

6.2.3.4 *Extension of initMeth.c*

```
int dimRange[2] = {2,2}; //limit experiment to two dimensions
```

6.2.3.5 *Extensions to parsRelations.c*

```
void backbone( void )
{
...
  // make sure to  get NI set to 1
  STB_UpdateSliceGeoPars(1,1,1,minSliceThick);
...
}
```

6.2.3.6 *Extensions to BaseLevelRelations.c*

```
void SetBaseLevelParam()
{ ...
  SetPipeFilterParameters();
  DB_MSG(("<--SetBaseLevelParam"));
}
void SetPipeFilterParameters( void )
{
  ACQ_user_filter = Yes;
  ParxRelsParRelations("ACQ_user_filter", Yes);
  sprintf(ACQ_user_filter_name,"MagControl");
  sprintf(ACQ_user_filter_setup_name,"");
  ACQ_user_filter_memory = For_one_scan;
  ACQ_user_filter_mode = Special;
  ParxRelsParRelations("ACQ_user_filter_mode", Yes);
  ACQ_user_filter_size[0] = ACQ_size[0] +
                            (int)(SW_h * (D[3] + D[5] + D[4]) + 0.5)*2;
  ACQ_user_filter_size[1] = 0;
  DS = NDummyScans = 0; // Disable dummy scans at all
}
```

to round up additional points

6.2.3.7 *The Pipeline Filter MagControl*

```
// Magnetization Control for non-steady-state acquisition

double d5, sw_h;
double *meanRef;
int magSize;

PIPE_INIT
PARX_INIT
/*-------------------------------------------------*/
```

```
/* retreive some parameters from parameter space */
/*-----------------------------------------------*/
PARX_get_all_values(PARX_psid,
              "D[5]",                &d5,
              "SW_h",                &sw_h,
              NULL_PTR );

magSize = (int)(sw_h * d5)*2; // # of points recording the
Magnetization
if (magSize < 2) magSize = 2;
if (acq_scan_type == Setup_Experiment)
     if (magSize > acq_size[0]/4) magSize = acq_size[0]/4;
meanRef = malloc(ni * sizeof(double));
if (NULL == meanRef) EXCEPT_printf("Cannot allocate memory for mean
buffer");

PIPE_PROC
int     niStep, niIdx, curNi, scan_cnt, dummy, *ip, *op;
double  mean, Re, Im;

while (1)
{
  for (scan_cnt=0; scan_cnt < scans_per_exp; scan_cnt++)
  {
    PIPE_WAIT_INPUT_OUTPUT(1,1,dummy,dummy);
    ip = (int *) in_ptr;
    op = (int *) out_ptr;
    if (acq_scan_type == Setup_Experiment)
    {                                   // Pass Magnetization during
Setup
      for (i1=0; i1 < magSize; i1++)
          *op++ = *(ip+i1);
      ip = ip + filter_size[0] - acq_size[0];
      for (i1=0; i1 < scan_word_size - magSize; i1++)
          *op++ = *ip++;
    }
    else
    {                                   // Calculate mean during Scan
      niStep = scan_cnt / ni;
      niIdx = scan_cnt % ni;
      mean = 0.0;
      for (i1=0; i1 < magSize / 2; i1++)
      {
          Re = *(ip+i1*2);
          Im = *(ip+i1*2+1);
          mean += sqrt(Re*Re + Im*Im);
      }
      mean = mean / magSize / 2;
      if (niStep == 0)
      meanRef[niIdx] = mean;            // meanRef[niIdx] = mean of
first NI Scans
      ip = ip + filter_size[0] - acq_size[0];
      for (i1=0; i1 < scan_word_size; i1++) // Pass normalized Scan
          *op++ = (int)( meanRef[niIdx] / mean * *ip++);
    }
    PIPE_SIGNAL_OUTPUT(1,1);
  }
  if (acq_scan_type == Scan_Experiment) break;
```

```
  scans_in = 0;
} /* while (1) */
PIPE_WRAPUP
PIPE_END
```

6.2.3.8 Making greMc ready to use

```
make clean
make cproto
make depend
make install

cp greMc.ppg $XWINNMRHOME/exp/stan/nmr/lists/pp
cp MagControl $XWINNMRHOME/exp/stan/nmr/au/src
$XWINNMRHOME/exp/stan/nmr/au/makeau MagControl
```

6.2.4 Exercises

- Run a GSP experiment and make a SnapShot of the acquisition display
- Set manually **ACQ_size[0] = ACQ_user_filter_size[0]**, deactivate the pipeline filter by setting **ACQ_user_filter=No** and run a GSP experiment in order to have the whole Scan displayed
- Compare the SnapShot with the whole Scan
- **Experiment 1**: Setup a **short TR** and **a 30 degree excitation pulse** and run the sequence **without dummy scans** by setting PVM_RepetitionTime = min and ExcPulse.FlipAngle = 30.0 deg
- **Experiment 2**: **Move the K-space center to the beginning of the acquisition** by setting ACQ_phase_enc_start[1] = 0
- **Experiment 3**: Apply a **90 degree excitation pulse** instead of an alpha pulse by setting ExcPulse.FlipAngle = 90.0 deg
- Disable the normalization in **MagControl** and run the experiments again
- Compare the scaled with the un-scaled experiments

7 References

7.1 Advanced Users Manual, Chapter D-9.3: AU filters for Pipelined Acquisition

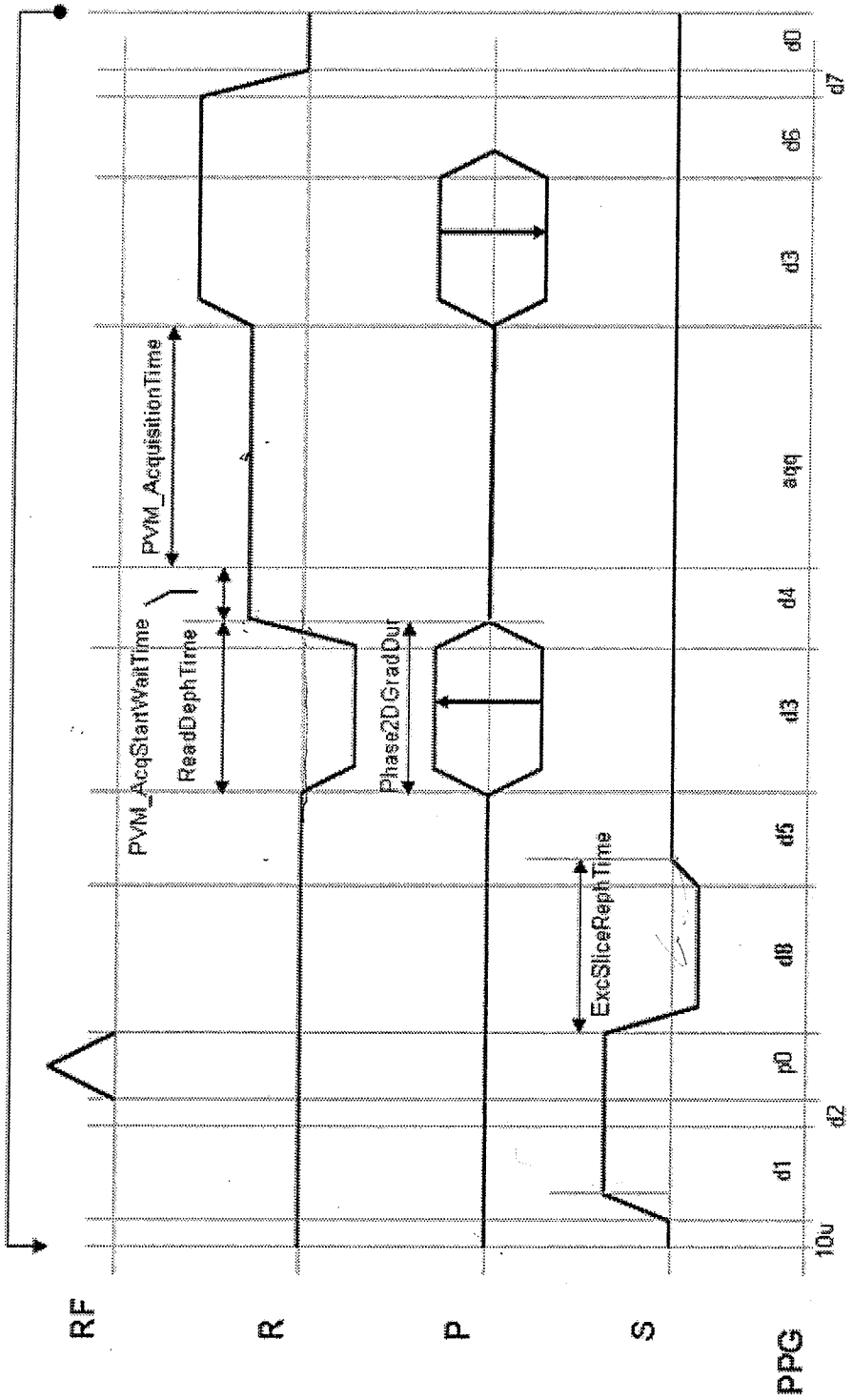
7.2 Advanced Users Manual, Chapter D-9: Automation Programming

7.3 $XWINNMRHOME/exp/stan/nmr/au/vorspann_f

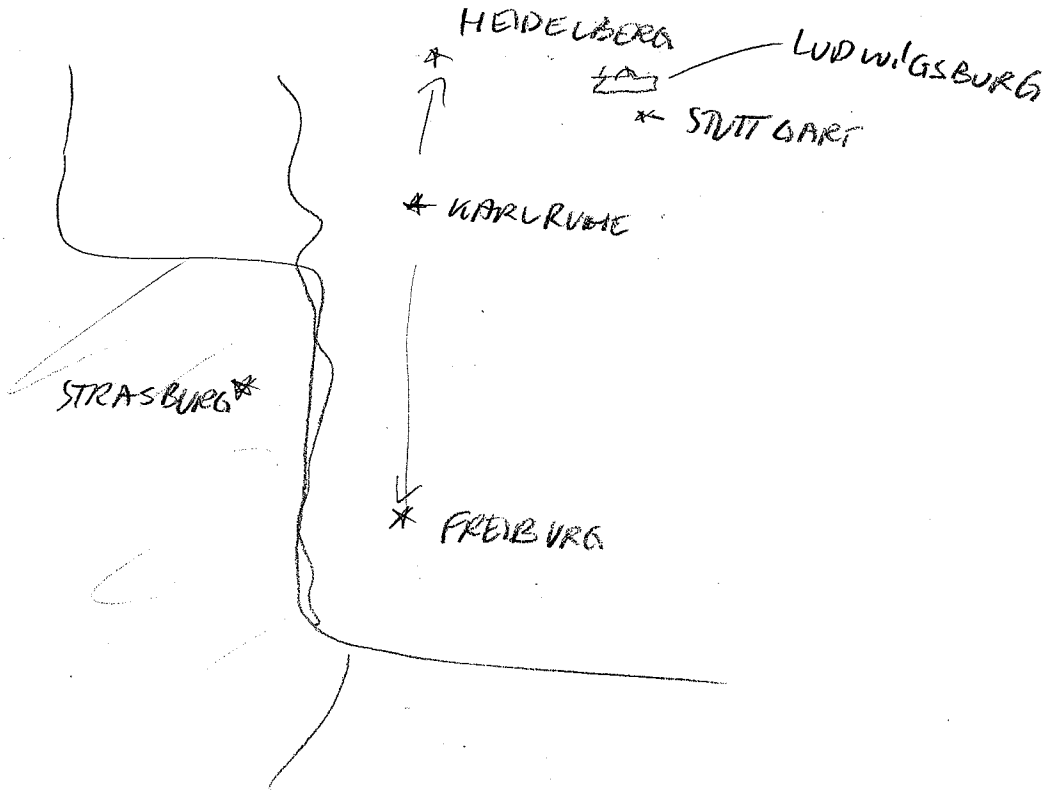
Definition of the "Convenience Parameters"

7.4 $XWINNMRHOME/prog/include/aupipe.h

Macro definitions and initialization of the "Convenience Parameters"



Method "ItGre"


HEIDELBERG
LUDWIGSBURG
STUTTGART
KARLSRUHE
STRASBURG
FREIBURG