# Cored Programming

Building Systems on Your Own Soft Core

OR

Blurring the

Hardware / Software Interface

Klaus.Schleisiek AT freenet.de

www.microcore.org

# Topics

Architecture

Opcodes

Co-Design Environment

Debug Environment

Geolon-MCS (a concrete product)

Interrupts and Exceptions

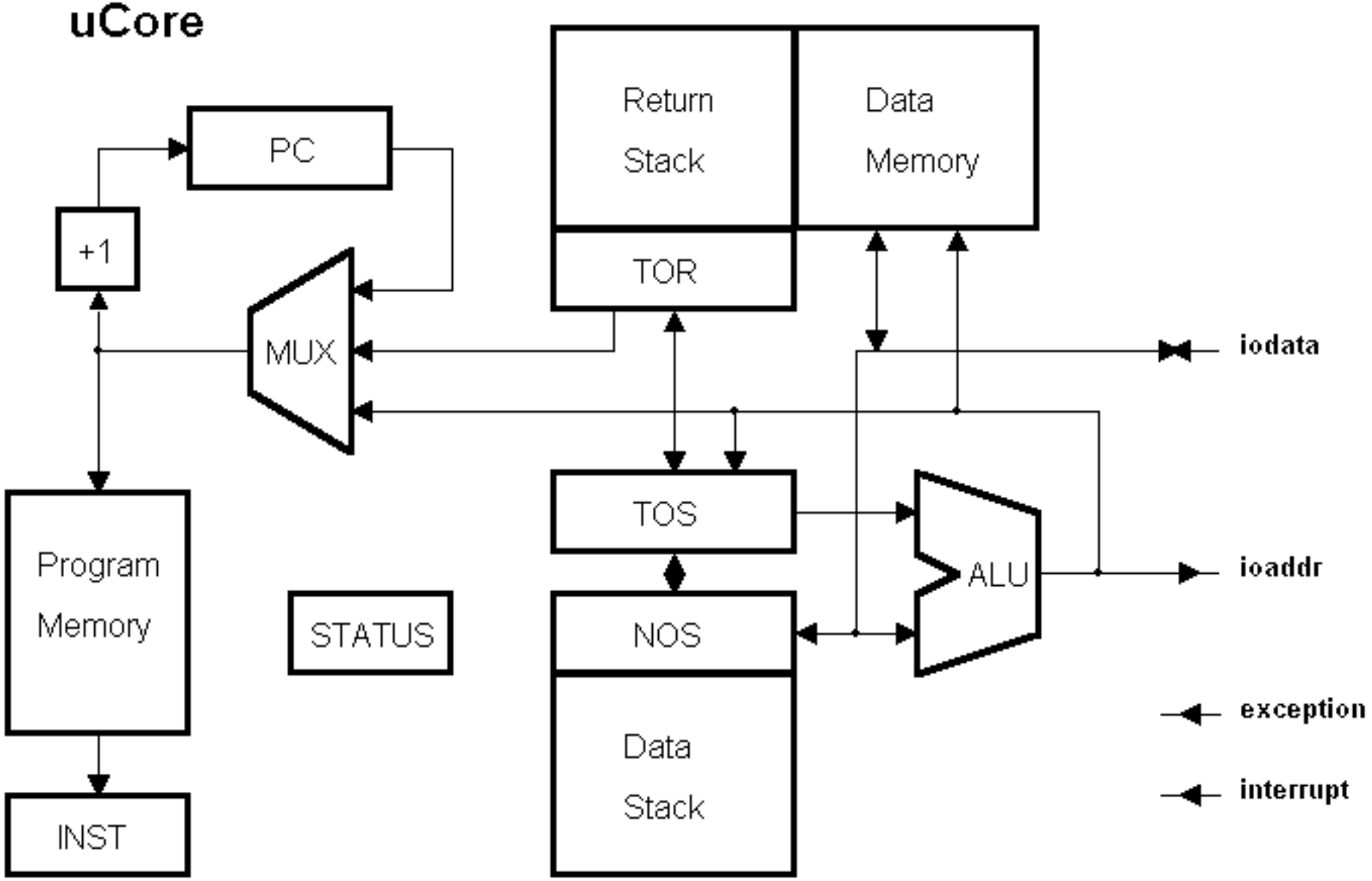
Special Instructions

Scalability

Prototyping Board

Licensing

Next Steps


uCore
Return
Stack
Data
Memory
PC
+1
TOR
MUX
iodata
TOS
ALU
ioaddr
Program
Memory
STATUS
NOS
exception
Data
Stack
interrupt
INST

# uCore Architecture

- Dual Stack
  - Forth oriented instruction set
- Harvard
- Prefix Code (Transputer Style)
  - Scalable data width
    - Object code compatibility
- Interrupt
  - after every instruction
- Exception
  - hardware mechanism for multi tasking

# uCore 1.20 Opcodes by Functionality

```
NEVER  NONE BRA Op: nop    ( -- )
```

## Unary Operators

```
NOT    BOTH ALU Op: invert ( n1 -- n2 )
SL     BOTH ALU Op: 2*     ( n1 -- n2 )
ASR    BOTH ALU Op: 2/     ( n1 -- n2 )
LSR    BOTH ALU Op: u2/    ( n1 -- n2 )
ROR    BOTH ALU Op: ror    ( n1 -- n2 )
ROL    BOTH ALU Op: rol    ( n1 -- n2 )
ZEQU   BOTH ALU Op: 0=     ( n1 -- flag )
CC     BOTH ALU Op: cc     ( -- )
```

# uCore 1.20 Opcodes by Functionality

## Binary operators

```
ADD    POP  ALU Op: +           ( n1 n2 -- n1+n2 )
ADC    POP  ALU Op: +c          ( n1 n2 -- n1+n2+carry )
SUB    POP  ALU Op: -           ( n1 n2 -- n1-n2 )
SSUB   POP  ALU Op: swap-       ( n1 n2 -- n2-n1 )
AND    POP  ALU Op: and         ( n1 n2 -- n1_and_n2 )
OR     POP  ALU Op: or          ( n1 n2 -- n1_or_n2 )
XOR    POP  ALU Op: xor         ( n1 n2 -- n1_xor_n2 )
ADD    PUSH ALU Op: 2dup +      ( n1 n2 -- n1 n2 n1+n2 )
ADC    PUSH ALU Op: 2dup +c     ( n1 n2 -- n1 n2 n1+n2+carry )
SUB    PUSH ALU Op: 2dup -      ( n1 n2 -- n1 n2 n1-n2 )
SSUB   PUSH ALU Op: 2dup swap-  ( n1 n2 -- n1 n2 n2-n1 )
AND    PUSH ALU Op: 2dup and    ( n1 n2 -- n1 n2 n1_and_n2 )
OR     PUSH ALU Op: 2dup or     ( n1 n2 -- n1 n2 n1_or_n2 )
XOR    PUSH ALU Op: 2dup xor    ( n1 n2 -- n1 n2 n1_xor_n2 )
```

# uCore 1.20 Opcodes by Functionality

## Complex Math Steps

```
MULTS  NONE ALU Op: mults  ( u1 u2 -- u1 u3 )
0DIVS  NONE ALU Op: 0divs  ( ud1 u2 -- u2 u3 )
UDIVS  NONE ALU Op: udivs  ( u2 u3 -- u2 u3' )
LDIVS  NONE ALU Op: ldivs  ( u2 u3 -- u2 u3' )
```

## Stack manipulation

```
DUP    PUSH BRA Op: dup    ( n -- n n )
QDUP   PUSH BRA Op: ?dup   ( n -- n n | 0 )
NEVER  BOTH BRA Op: drop   ( n -- )
SWAPS  NONE ALU Op: swap   ( n1 n2 -- n2 n1 )
NOS    PUSH ALU Op: over   ( n1 n2 -- n1 n2 n1 )
NOS    POP  ALU Op: nip    ( n1 n2 -- n2 )

RSTACK BOTH MEM Op: r>     ( -- n )
RSTACK NONE MEM Op: >r     ( n -- )
TOR    BOTH MEM Op: r@     ( -- n )
TOR    NONE MEM Op: r!     ( n -- )
```

# uCore 1.20 Opcodes by Functionality

## Registers

```
STATUS BOTH MEM Op: status@        ( -- status )
STATUS NONE MEM Op: status!        ( status -- )
RSP    BOTH MEM Op: rsp@           ( -- rstack_addr )
RSP    NONE MEM Op: rsp!           ( rstack_addr -- )
DSP    BOTH MEM Op: dsp@           ( -- dstack_ptr )
DSP    NONE MEM Op: dsp!           ( dstack_ptr -- ? )
```

## Data Memory access

```
N      PUSH MEM Op: +ld            ( addr -- n addr+i )
N      POP  MEM Op: +st            ( n addr -- addr+i )
LOCAL  BOTH MEM Op: +lld           ( i -- n rstack+i )
LOCAL  NONE MEM Op: +lst           ( n i -- rstack+i )
TASK   BOTH MEM Op: +tld           ( i -- n task+i )
TASK   NONE MEM Op: +tst           ( n i -- task+i )
```

# uCore 1.20 Opcodes by Functionality

## Exits

```
ALWAYS NONE BRA Op: exit          ( -- )
ZERO   NONE BRA Op: z-exit        ( flag -- )
NZERO  NONE BRA Op: nz-exit       ( flag -- )
SIGN   NONE BRA Op: s-exit        ( -- )
NSIGN  NONE BRA Op: ns-exit       ( -- )
NOVL   NONE BRA Op: no-exit       ( -- )
NCARRY NONE BRA Op: nc-exit       ( -- )
```

## Branches

```
ALWAYS POP  BRA Op: branch        ( offset -- )
ZERO   POP  BRA Op: z-branch      ( offset -- )
NZERO  POP  BRA Op: nz-branch     ( offset -- )
SIGN   POP  BRA Op: s-branch      ( offset -- )
NSIGN  POP  BRA Op: ns-branch     ( offset -- )
NOVL   POP  BRA Op: no-branch     ( offset -- )
NCARRY POP  BRA Op: nc-branch     ( offset -- )
```

# uCore 1.20 Opcodes by Functionality

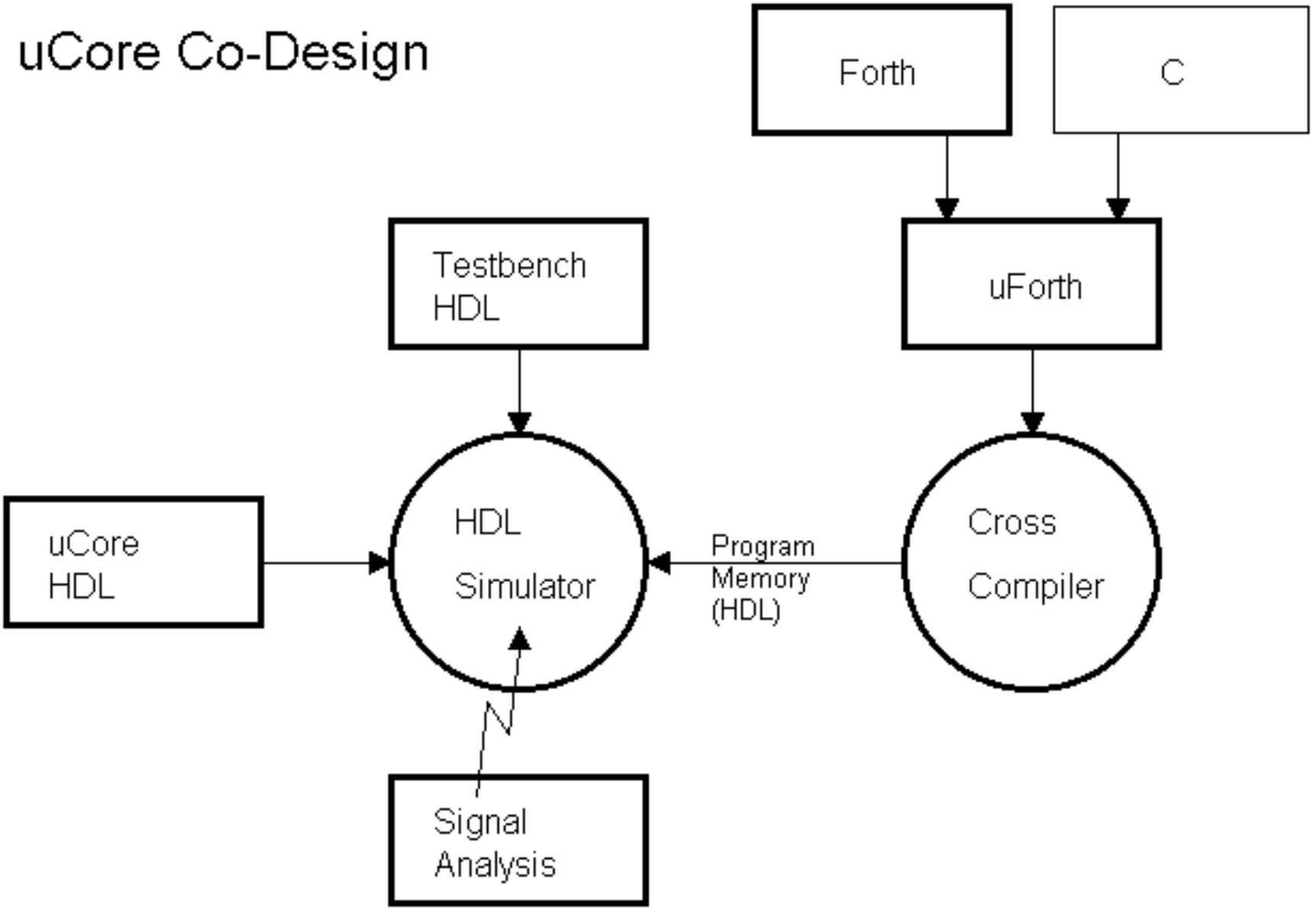
## Calls

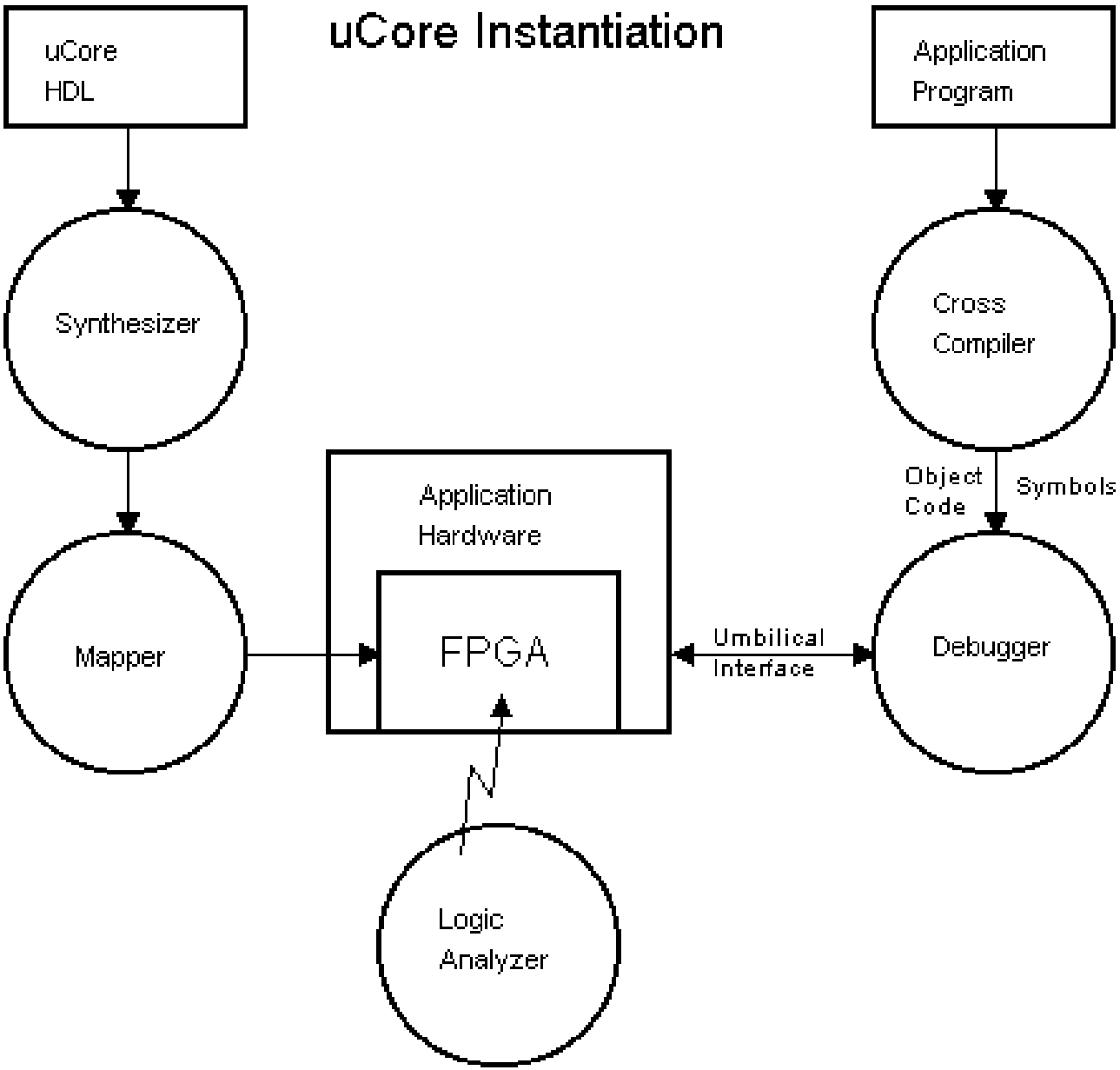
```
ALWAYS BOTH BRA Op: call          ( offset -- )
ZERO   BOTH BRA Op: z-call        ( offset -- )
NZERO  BOTH BRA Op: nz-call       ( offset -- )
SIGN   BOTH BRA Op: s-call        ( offset -- )
NSIGN  BOTH BRA Op: ns-call       ( offset -- )
NOVL   BOTH BRA Op: no-call       ( offset -- )
NCARRY BOTH BRA Op: nc-call       ( offset -- )
QOVL   PUSH BRA Op: ?ovl          ( -- )
          N USR Op: user_call     32 uncommitted instructions
```

## Complex Branches

```
INT    PUSH BRA Op: int           ( -- status )
IRET   PUSH BRA Op: iret          ( status -- )
EXC    PUSH BRA Op: exc           ( -- )
TIMES  PUSH BRA Op: times         ( n-1 -- )
NEVER  POP  BRA Op: drop_flag     ( flag offset -- offset )
```

# uCore Co-Design



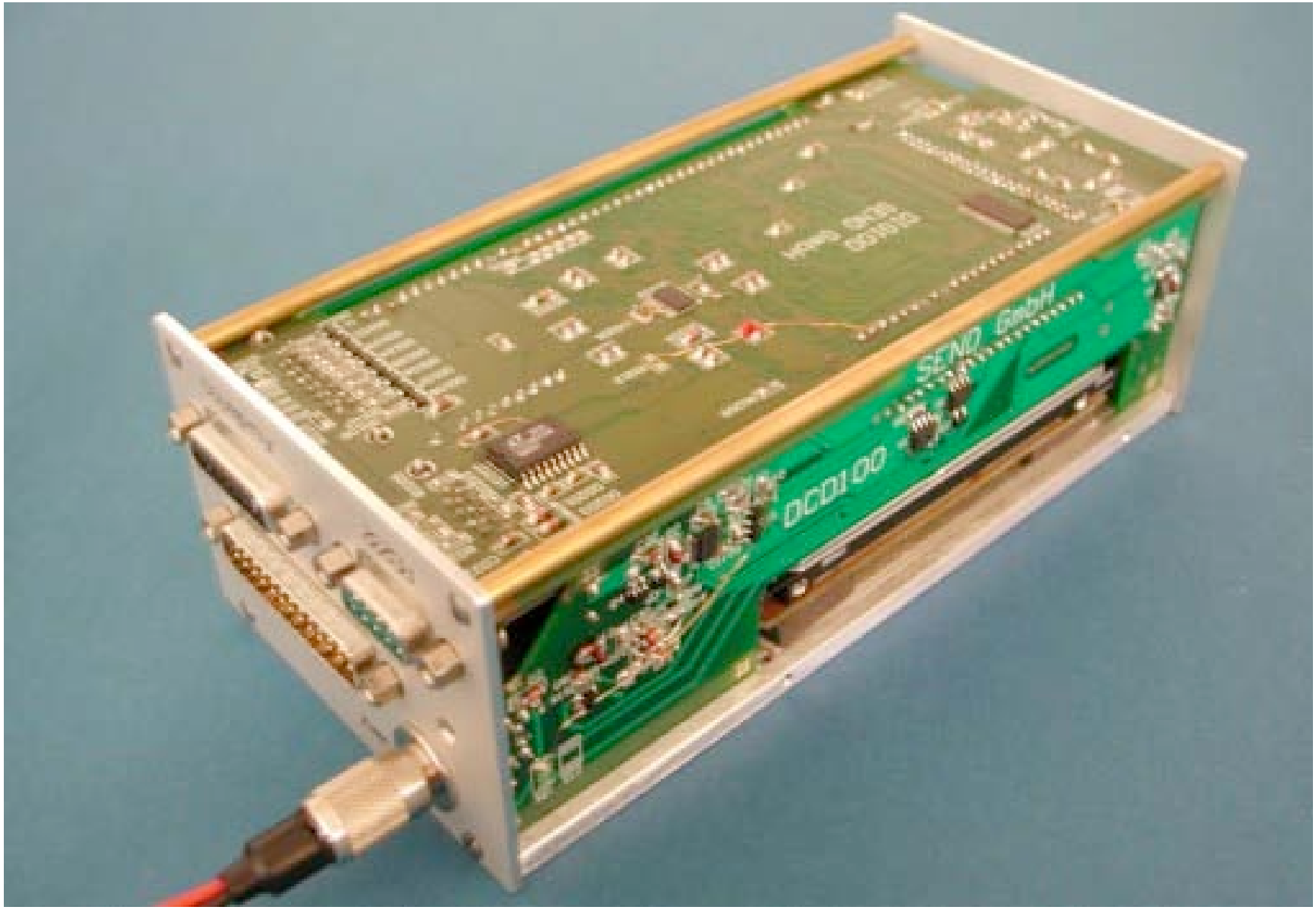

uCore Instantiation
uCore
HDL
Application
Program
Synthesizer
Cross
Compiler
Object
Code
Symbols
Application
Hardware
Mapper
FPGA
Umbilical
Interface
Debugger
Logic
Analyzer

# uCore instantiation for Geolon-MCS

| Data path width32 bits | Program memory256k external RAM, 2k internal "cache" | Data memory128k ex |
|---|---|---|
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

Booting

FPGA Technology Issues


SIEMO GmbH

# The Interrupt and Exception Duality

**Interrupt**

An event did happen that was **not** expected by software

**Exception**

An event did **not** happen that was hoped for by software

# Time Services using Exceptions

## Traditional Method using a Timer Interrupt

```
: ahead    ( ticks -- time.ahead )  Timer @ + ;

: timeout? ( ticks -- ticks f )     dup Timer @ - 0< ;

: continue ( ticks -- )     BEGIN  pause timeout? UNTIL drop ;

: sleep    ( ticks -- )     ahead continue ;
```

## Using an Exception

```
: continue ( ticks -- )     Timer ! ;
```

and now we can afford sub millisecond resolution as well

# Pseudo DMA

Transferring 16 bit data between IDE interface and Buffer memory

```
0 PUSH USR Opcode: ide@   ( bufaddr -- bufaddr+1 )

0 POP  USR Opcode: ide!   ( bufaddr -- bufaddr+1 )
```

and these types of instructions can be iterated using TIMES:

```
$FF times ide@
```

loads a sector into the buffer at one transfer/cycle plus 3 cycles overhead. Repeated instructions are fully interruptible without latency.

# Data Encoding and Buffering, soft encoded

(just appreciate its long winded complications)

```
: split  ( 32b -- l16b h16b ) dup $FFFF and swap u256/ u256/ ;
: wsplit ( 16b -- l8b h8b )   dup $FF and swap u256/ ;

: buf8! ( 8bit -- )
  >r idebuf_ptr @ 2/
  carry IF    ld swap r> 256* or swap !
        ELSE  r> $FF and swap !
        THEN
  idebuf_ptr ld swap 1+
  dup [ #idebuf_top 2* ] literal u>
  IF  idebuf_flush drop #idebuf_addr 2*  THEN
  swap !
;
: buf16! ( 16bit -- ) wsplit buf8!  buf8!  ;
: buf24! ( 24bit -- ) split  buf8!  buf16! ;
: buf32! ( 32bit -- ) split  buf16! buf16! ;
: abs  ( n -- u )   neg IF  0 swap-  THEN ;

: encode ( sample channel# -- )
  under           \ val val ch
  samples +       \ val val addr
  ld >r swap r> ! \ val old    store val in samples buffer
  under -         \ val difference
  dup abs         \ val difference |difference|
  dup $40 <   IF drop $7f and buf8!
                 drop EXIT
              THEN
  dup $1000 < IF drop $1FFF and $C000 or buf16!
                 drop EXIT
              THEN
  $80000 <    IF $FFFFF and $E00000 or buf24!
                 drop EXIT
              THEN
  drop $7FFFFFF and $F0000000 or buf32!
;
```

# Data Encoding and Buffering, hardware supported

```
1 POP  USR Opcode: buf8!   ( 8b -- )
2 POP  USR Opcode: buf16!  ( 16b -- )
1 NONE USR Opcode: buf24!  ( 24b -- 16b )
2 NONE USR Opcode: buf32!  ( 32b -- 16b )

ENCODE NONE ALU Opcode: tag  ( sample previous -- sample code )

: encode ( val ch# -- )
   samples + ld >r tag
   carry IF  buf8!  ELSE  ovl IF  buf32!  THEN  buf16!  THEN
   r> !
;
```

The core can be clocked at 1/4 speed, saving 20 mW.

The code is also more understandable and therefore, reliable beyond doubt©.

# uCore Scalability

```
CONSTANT data_width         : NATURAL := 32;
CONSTANT data_addr_width    : NATURAL := 21;
CONSTANT dcache_addr_width  : NATURAL :=  0;

CONSTANT prog_addr_width    : NATURAL := 19;
CONSTANT pcache_addr_width  : NATURAL :=  0;
CONSTANT prog_ram_width     : NATURAL := 16;

CONSTANT ds_addr_width      : NATURAL :=  6;
CONSTANT rs_addr_width      : NATURAL :=  8;

CONSTANT tasks_addr_width   : NATURAL :=  3;

CONSTANT interrupts         : NATURAL :=  2;
```

# uCore100 Prototyping Board

XC2S200 FPGA

256k x 8 program memory

256k x 32 data memory/return stack

User USB port

USB configuration port (to be programmed!)

Umbilical Interface

- Centronics
- RS232

Lots of uncommitted I/O pins

# uCore Licensing

uCore Exploratory License

- Equivalent terms to FreeBSD

My Business Interest

- tayloring instructions
- certifying compatibility with the original model

# Next Steps

Building a Development Community

GCC backend

Simulator for Windows/Linux

USB Programming Interface (8051$^{\text{no fun}}$)

www.microcore.org