Devil: an IDL for Hardware Programming

Fabrice Mérillon, Laurent Réveillère,
Charles Consel, Renaud Marlet,
Gilles Muller

Compose Group
http://www.irisa.fr/compose
Irisa/Labri, Rennes/Bordeaux (France)

Interfacing Devices ...

The Devil is
in the details

Looking into the "Details"

(busmouse.h)

```
#define MSE_DATA_PORT            0x23c
#define MSE_SIGNATURE_PORT       0x23d
#define MSE_CONTROL_PORT         0x23e
...
#define MSE_READ_X_LOW           0x80
#define MSE_READ_X_HIGH          0xa0
```

(busmouse.c)

```
outb(MSE_READ_Y_LOW, MSE_CONTROL_PORT );
dy = (inb(MSE_DATA_PORT) & 0xf);
outb(MSE_READ_Y_HIGH, MSE_CONTROL_PORT);
buttons = inb(MSE_DATA_PORT);
dy |= (buttons & 0xf) << 4;
buttons = ((buttons >> 5) & 0x07);
```

Assessment of Existing (Linux) Drivers

- Assembly-level programming style
- Macros
  - unreadable code factorized, not suppressed
  - no single programming style
  - no typing / consistency checking

The hardware interface code is:

⤷hard to read and maintain

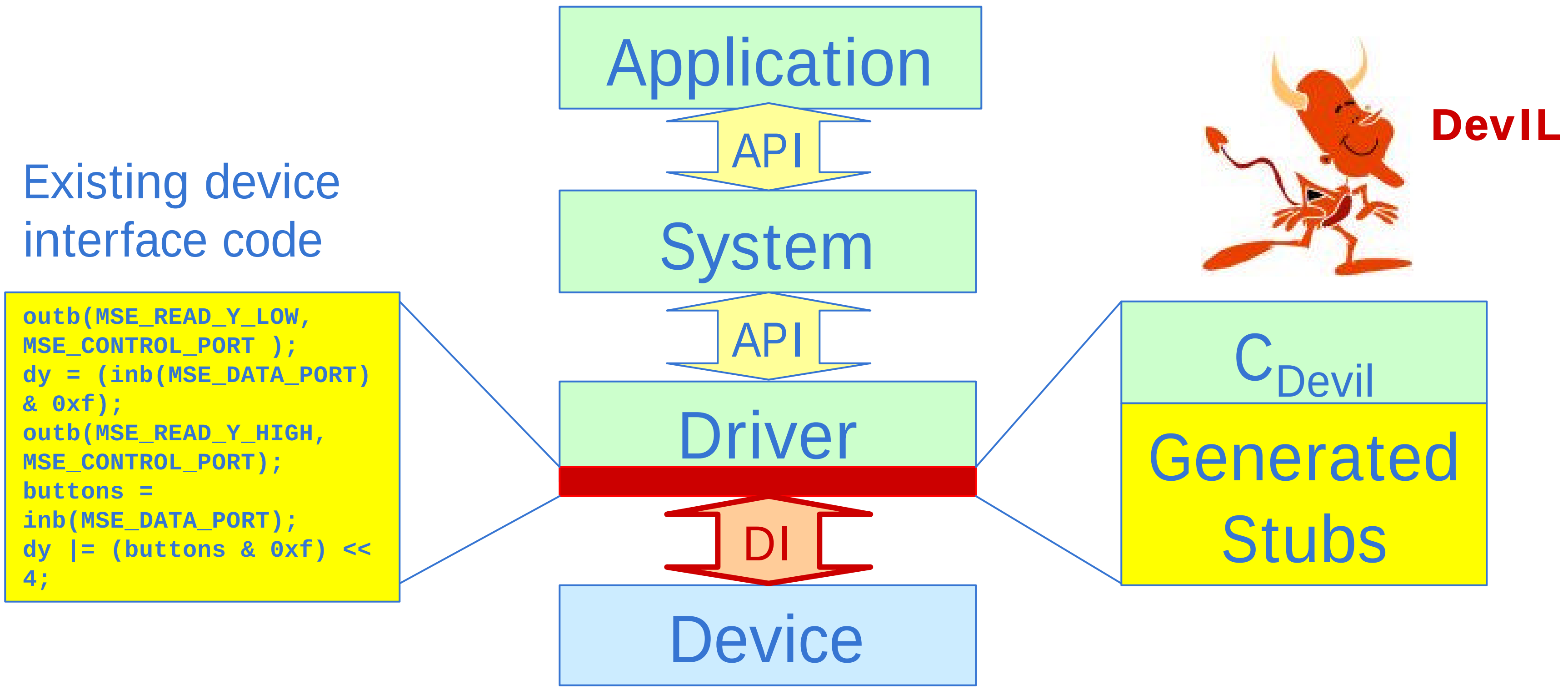
⤷tedious and error prone

Lack of Support for the Driver Programmer

- Paper documentation
  - » natural language
  - » manufacturer-specific terminology
  - » inconsistencies, ambiguities, omissions, typos
- Mixed levels of abstractions
  - » communication mechanisms, data layout, semantics
- Inherent complexity of devices

↪Laborious testing until expected functionality is obtained

Devil: a DEVice Interface Language

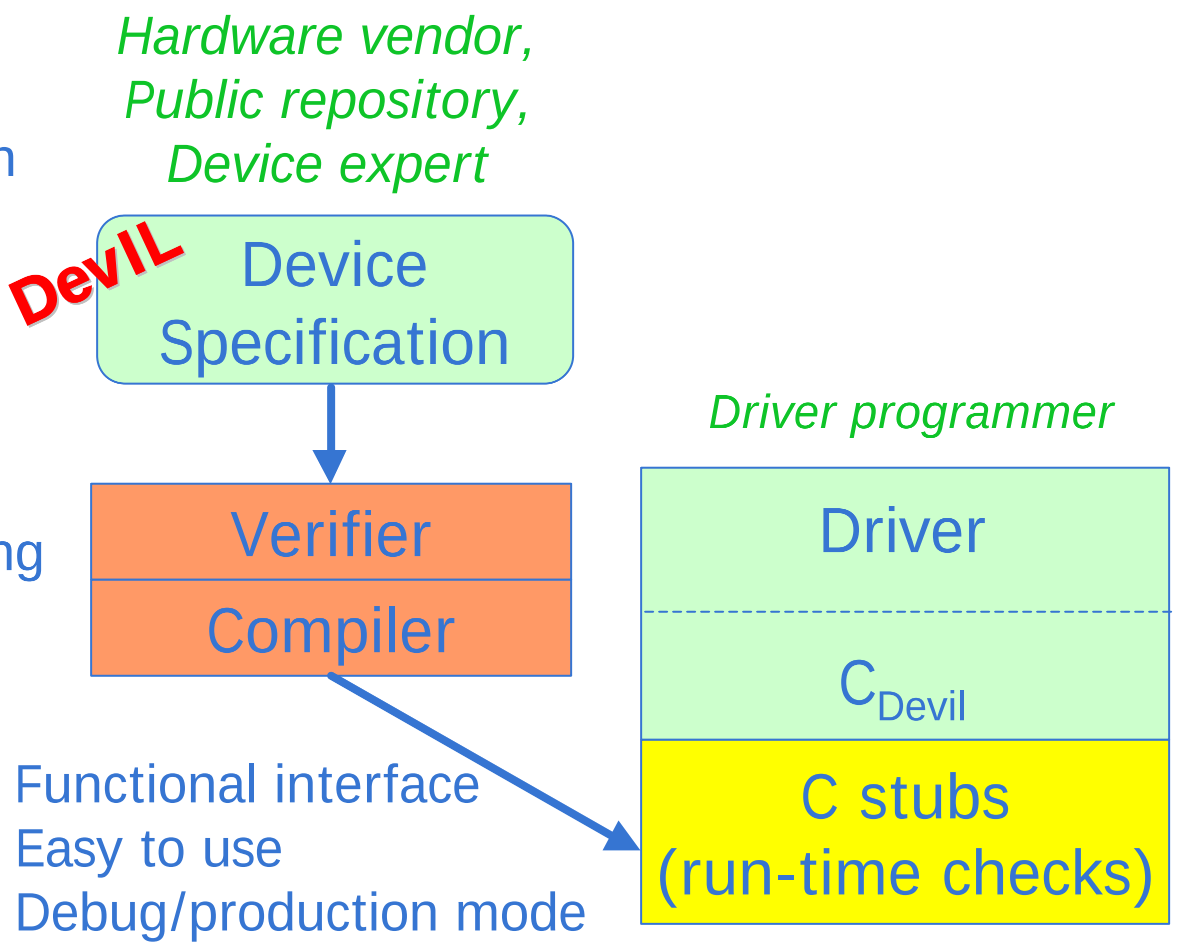


```
outb(MSE_READ_Y_LOW,
MSE_CONTROL_PORT );
dy = (inb(MSE_DATA_PORT)
& 0xf);
outb(MSE_READ_Y_HIGH,
MSE_CONTROL_PORT);
buttons =
inb(MSE_DATA_PORT);
dy |= (buttons & 0xf) <<
4;
```

Our Vision

- High-level description of device interface
- Easy to write
- Strongly typed

- Consistency checking
- Code generation



- Functional interface
- Easy to use
- Debug/production mode

Devil: Key Concepts

- Ports
  - communication point, address range
- Registers
  - repository of data, granule of exchange
- Variables: programmer interface
  - collection of register fragments
  - semantic values:
    - » bounded integers
    - » enumerated types

Programmer Support: Verifying Critical Properties

- Consistency of Devil specifications
  - – no omission
  - – no double definition
  - – no overlapping
  - – type/size of variables
- C_{Devil} interface usage in debug mode
  - – compile-time (type checking)
  - – run-time (assertion checking)

The Logitech Busmouse: functional interface

- dx (delta X)
  - relative horizontal movement
- dy (delta Y)
  - relative vertical movement
- buttons
  - button state

Read only variables !

Device Interface Code: C_{Devil} vs. existing drivers

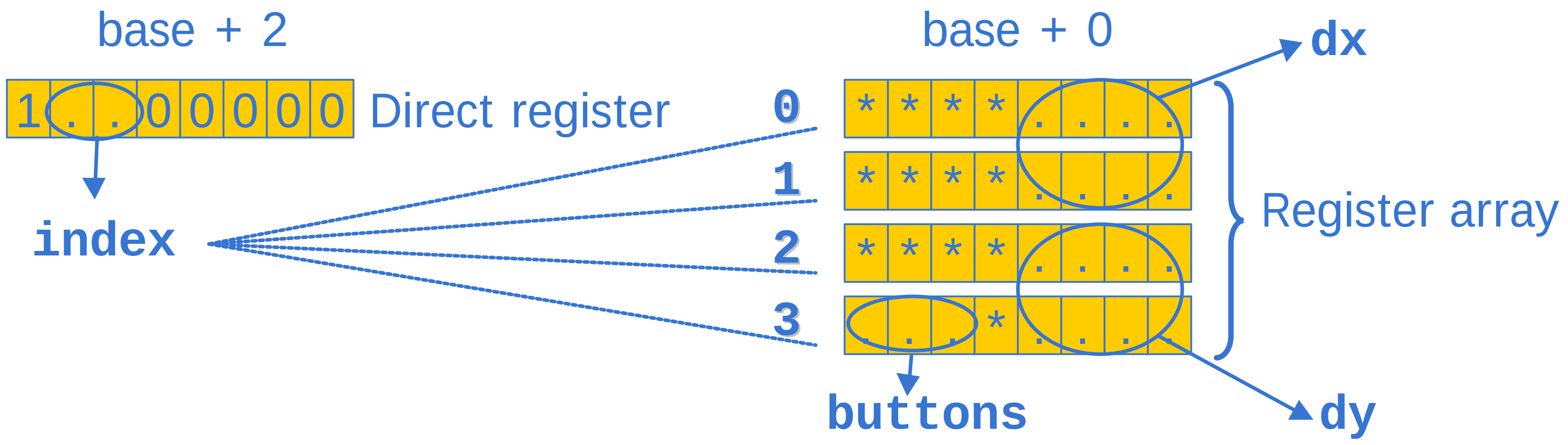
```
dx = get_dx();
dy = get_dy();
buttons = get_buttons();
```

C_{Devil}

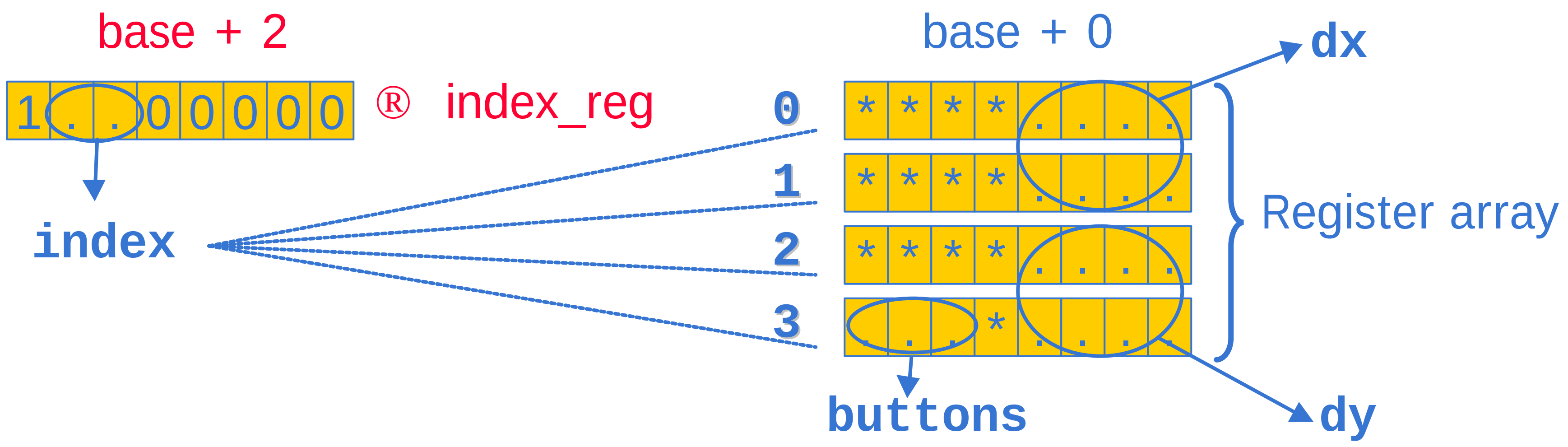
```
outb(MSE_READ_X_LOW, MSE_CONTROL_PORT);
dx = (inb(MSE_DATA_PORT) & 0xf);
outb(MSE_READ_X_HIGH, MSE_CONTROL_PORT);
dx |= (inb(MSE_DATA_PORT) & 0xf) << 4;
outb(MSE_READ_Y_LOW, MSE_CONTROL_PORT );
dy = (inb(MSE_DATA_PORT) & 0xf);
outb(MSE_READ_Y_HIGH, MSE_CONTROL_PORT);
buttons = inb(MSE_DATA_PORT);
dy |= (buttons & 0xf) << 4;
buttons = ((buttons >> 5) & 0x07);
```

Existing code

The Logitech Busmouse: detailed description

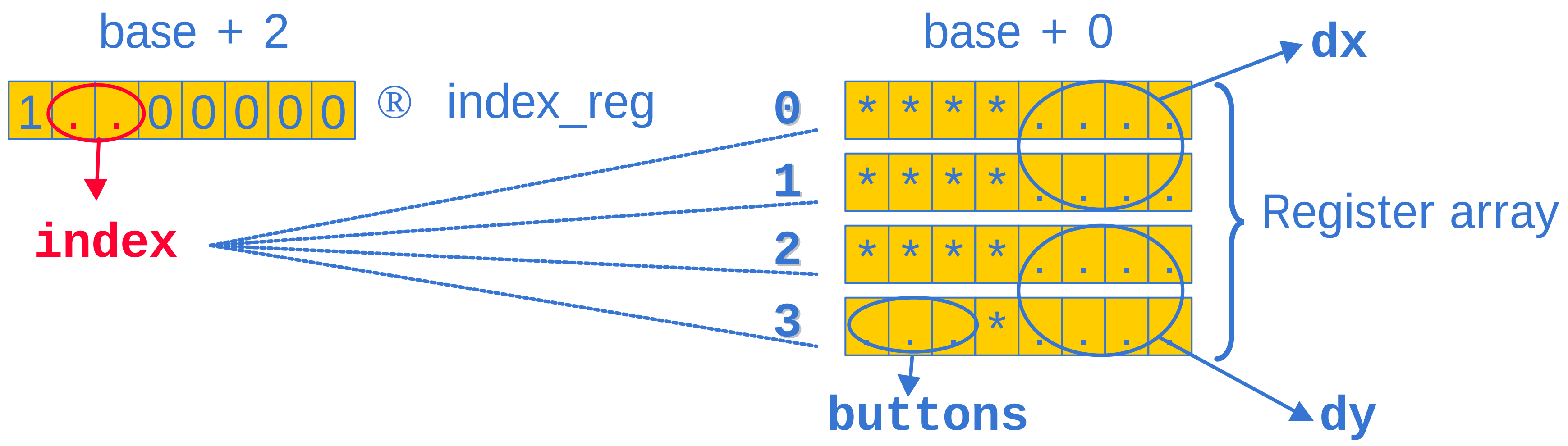


Direct registers



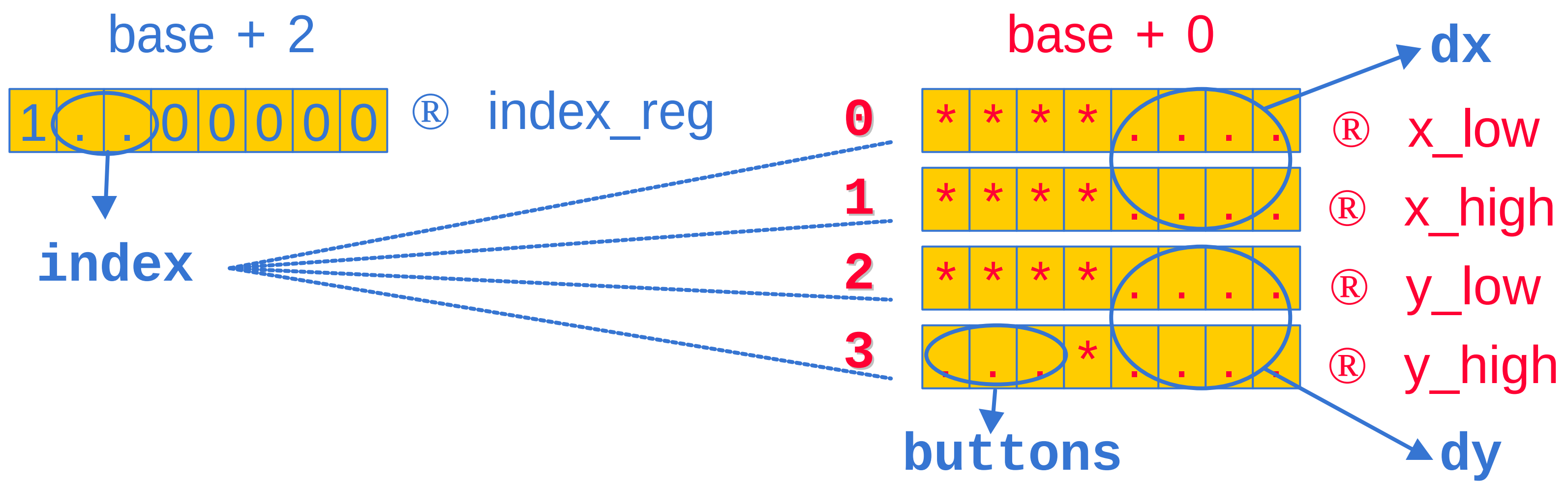
```
...
register index_reg = write base@2,
                     mask '1..00000': bit[8];
```

Private device variables



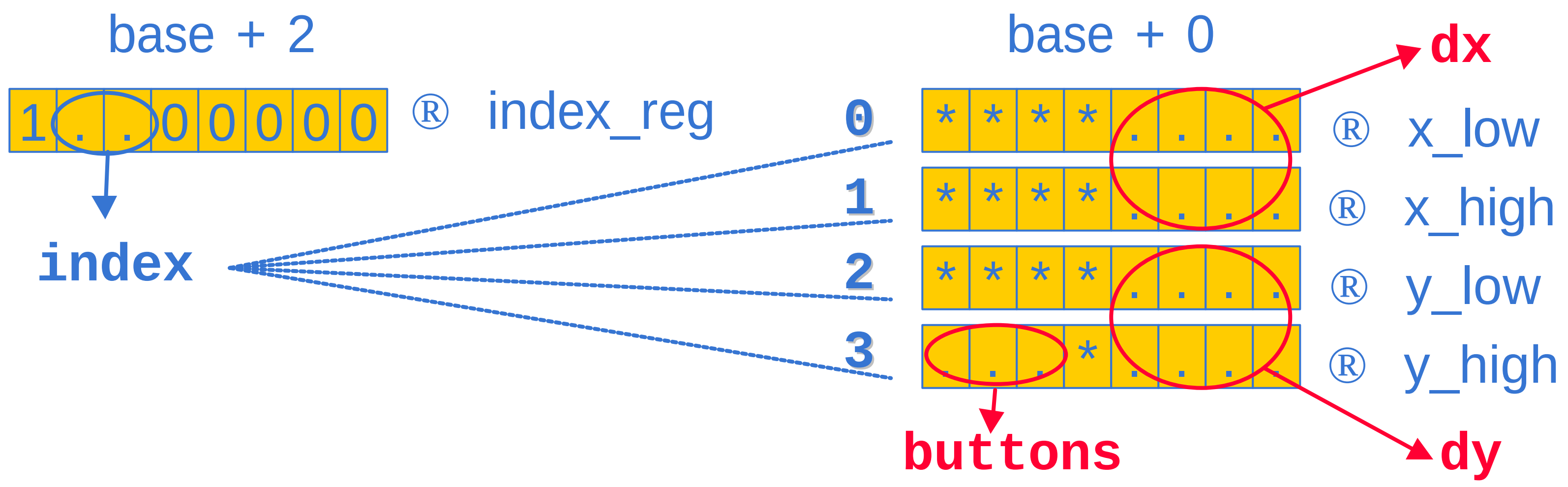
```
...
register index_reg = write base@2, mask '1..00000' : bit[8];
private variable index = index_reg[6..5] : int(2);
```

Indexed registers



```
...
register index_reg = write base@2, mask '1..00000' : bit[8];
private variable index = index_reg[6..5] : int(2);
...
register x_low  = read base@0, pre {index = 0},
                  mask '****....' : bit[8];
register x_high = read base@0, pre {index = 1},
                  mask '****....' : bit[8];
...
```

Interface variables



```
...
register index_reg = write base@2, mask '1..00000' : bit[8];
private variable index = index_reg[6..5] : int(2);
...
register x_low  = read base@0, pre {index = 0}, mask '****....' : bit[8];
register x_high = read base@0, pre {index = 1}, mask '****....' : bit[8];
...
variable dx = x_high[3..0] # x_low[3..0],
                  volatile : signed int(8);
variable dy = y_high[3..0] # y_low[3..0],
                  volatile : signed int(8);
```

What About Performance ?

- Sharing registers between variables may induce performance penalty
  - additional I/O w.r.t. hand-crafted drivers
  - command parameters
- Re-engineering of performance critical drivers
  - IDE disk driver
  - Permedia2 X11 driver

IDE Linux Driver (intel 82371SB)

- Characteristics:
  - many initializations
  - DMA/PIO-loop for transfer
- DMA mode

C: 14 I/Os	Devil: 20 I/Os	14.25 Mb/s

- PIO-32bits mode, 16 sectors/interrupt

C (rep loop):	8.17 Mb/s
Devil (C loop):	7.36 Mb/s (90% of C)
Devil (rep loop):	8.17 Mb/s

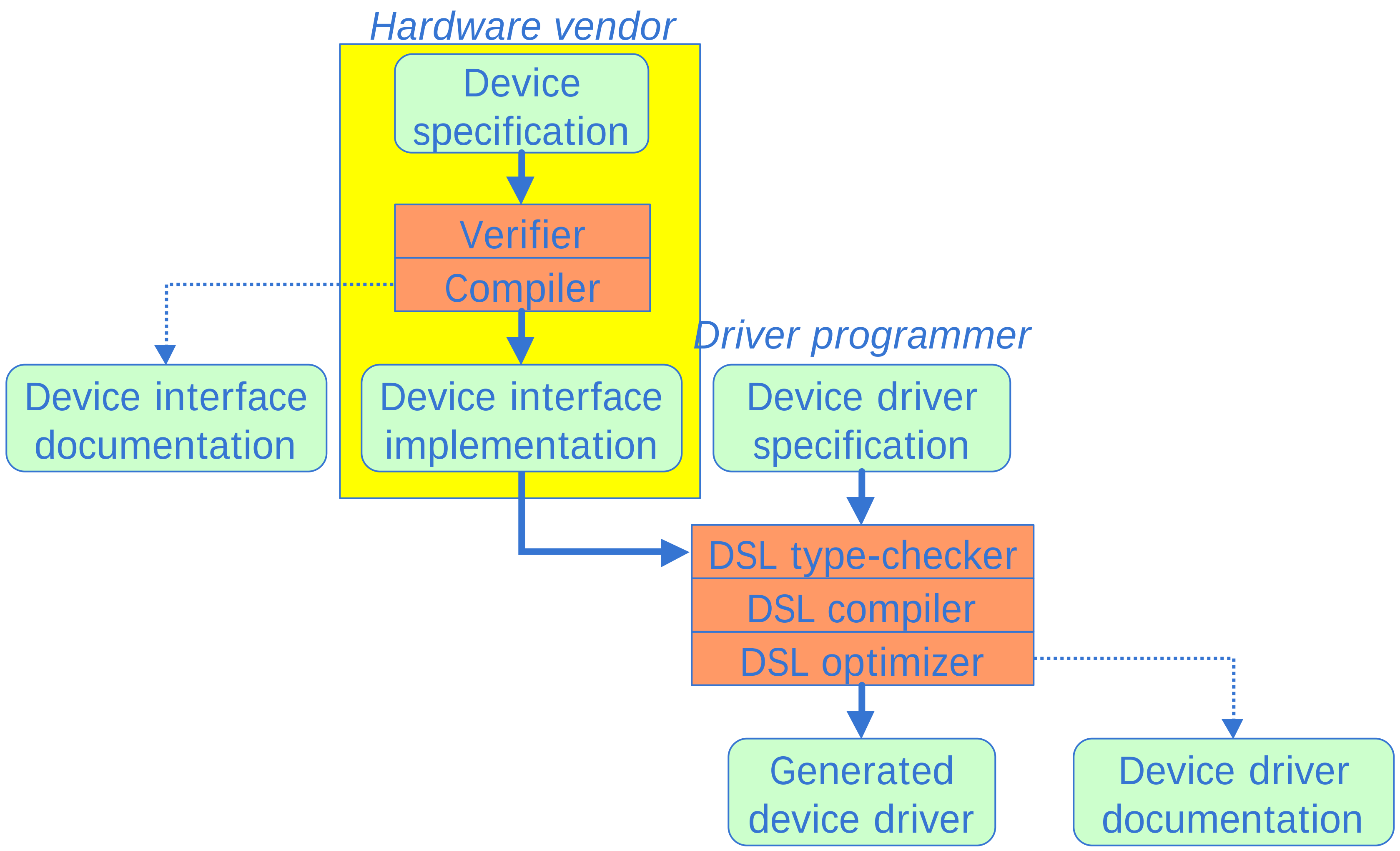
Permedia2 X11 driver

- Characteristics:
  - registers mapped in memory
  - buffered write (on-chip FIFO)
  - same number of I/Os
- Screen copy operations (24 bpp)
  - 100% performance of C
- Rectangle operations (24 bpp)
  - 97%-100% performance of C
  - difference due to stub code for small size operation

Benefits of Devil

- Expressivity
  - » advanced Devil constructs (see paper and manual)
  - » DMA, sound, interrupt, Ethernet controllers
- Guaranteed safety
  - » Mutation-based experiment (typo simulation)
  - » 5 times less prone to errors than C code
- Negligible performance overhead
- Improved productivity
  - » reuse of specifications
  - » tools and verifications

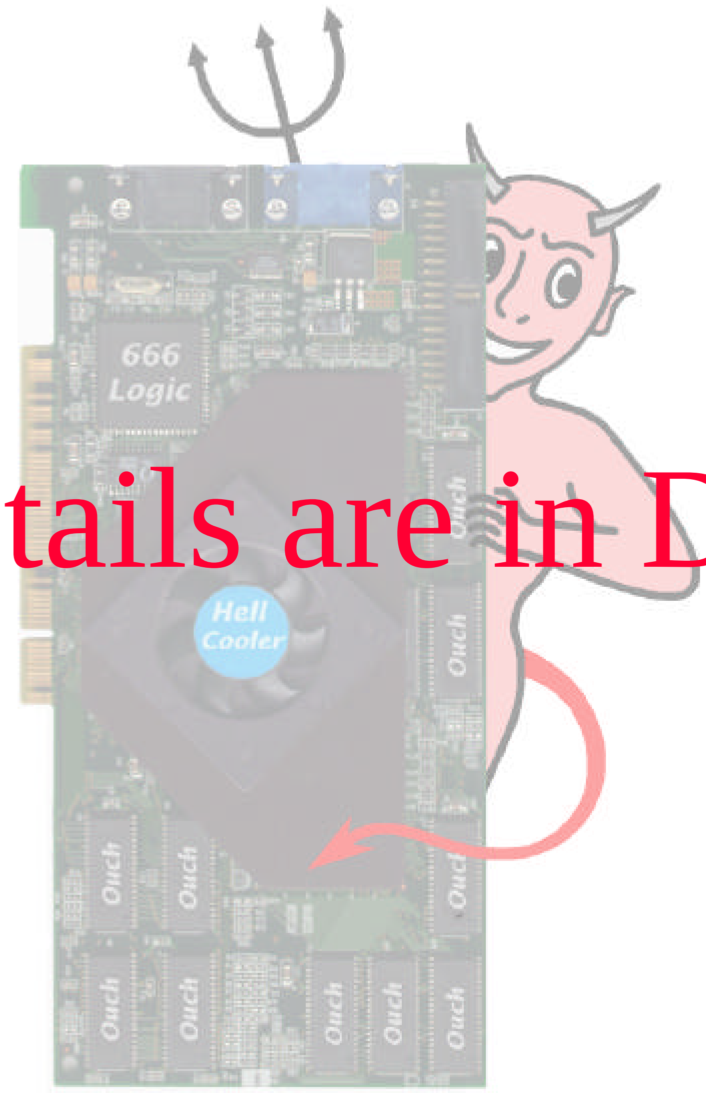
Our Vision (On-going work)



Conclusion

- Step toward the development of robust drivers
- Compiler/checker available
- No performance penalty
- Expressive enough to allow the specification of various devices

Instance of our vision: “DSL for Operating System Design”



The details are in Devil

Questions ?

- Specifications/compiler/manual available :

  www.irisa.fr/compose/devil

- Public CVS repository of specs

  Please contribute …