# Register diagrams with field descriptions

Matthew Lovell
lovell@indra.com

2007/03/08

**Abstract**

The `register` package is designed for typesetting the programmable elements in digital hardware, i.e., registers. Such registers typically have many fields and can be quite wide; they are thus a challenge to typeset in a consistent manner. This package attempts to address such issues. It is similar in some aspects to the `bitfield` package by Reuben Thomas and the `bytefield` package by Scott Pakin.

## List of Registers

## 1 Introduction

My group at work designs the memory and I/O controllers for HP's servers and workstations. Historically, our chip documentation was done with FrameMaker or, more recently, Microsoft Word. While these approaches have various disadvantages, one of the most egregious was register documentation.

Our recent chips have had 64-bit wide control-status registers (CSR) or, more simply, registers. Throw in the fact that many of these registers have a large number of single-bit fields, and you get a typesetting challenge. The typical solution was to describe such registers using a table, typing field names vertically if space became a problem. For a complicated register, these tables became quite complex and filled a large portion of a page.

When we decided to evaluate LaTeX for the latest round of documentation, we had three goals in mind with respect to registers:

1. Create a method of documenting registers which was consistent and easy to read, regardless of the number of fields within a register.

2. Automate the creation of lists of registers, both in order of appearance within the text and in memory order.
3. Enable the automatic extraction of documented register reset values in order to verify register functionality in the chip itself.

The `register` package is my attempt at meeting all three goals. It has been in use by my group since April 1999; it may be not be pretty to all eyes, but it certainly has proven itself stable.

In order to promote LaTeX within our group at the time, we adopted LyX. The `register` package thus attempts to work well within that environment. All LyX-specific code, however, is controlled via package options.

### 1.1 Feedback

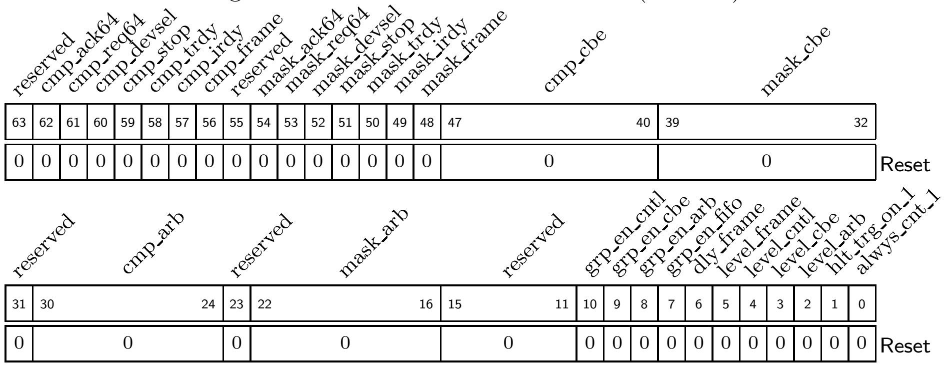
As `register` is my first LaTeX package, I would welcome feedback regarding problems, new features, or more proper means of implementing existing features. For example, is there an "official" method for determining the current font's maximum height and depth (see Section 5.4.1 for details)? I would also appreciate knowing whether anyone ever actually uses it.

It might also be worthwhile to investigate combining `register` with `bitfield` or `bytefield`.

## 2 Examples

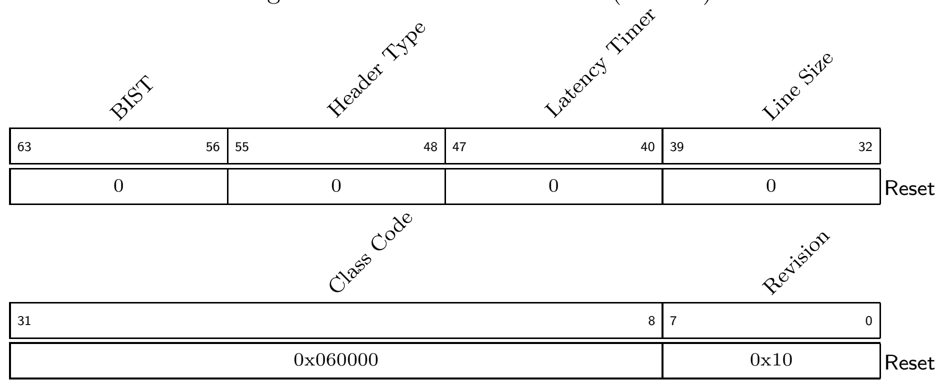
Register 2.1 depicts a register with many single-bit fields. Registers such as this stress any scheme of typesetting, since the field names are so much longer than the space available for them within the diagram. By typesetting the field names at an angle, `register` manages to be consistent regardless of the number or width of fields. This rotation, however, means that documents which use `register` must be produced using PostScript.

Register 2.1: Diagnostic Control (0x1F28)



Register 2.2 is an example of a register one could derive from the PCI specification. It is a miscellaneous register in a PCI card which controls device independent functions. Note the field descriptions which, in this case, are actually part of the float.

Register 2.2: Function Class (0x0008)



| | |
|---|---|
| **BIST** | (Read only) Always returns 0. |
| **Header Type** | (Read only) Always returns 0. |
| **Latency Timer** | PCI Latency Timer value (PCI 2.2 spec, Section 6.2.4). |
| **Line Size** | PCI Cache Line Size (PCI 2.2 spec, Section 6.2.4). Valid values are listed below; any other value will be treated as indicating 64 byte cache lines. **0010_0000** 128 bytes; **0001_0000** 64 bytes |
| **Class Code** | (Read only) PCI Class Code (PCI 2.2 spec, Section 6.2.1). Chip identifies itself as a Host bridge. |
| **Revision** | (Read only) Chip revision number. Bits 4–7 provide the major revision number, and bits 0–3 provide the minor revision number. |

A final example is shown in Figure 1. This example shows that the register drawing macros don't have to be used solely within a `register` float.

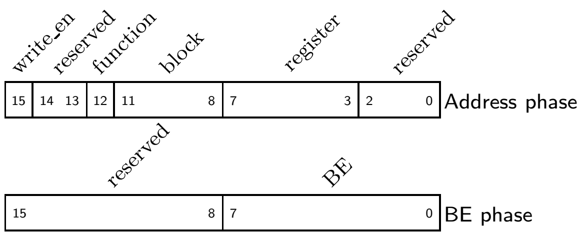
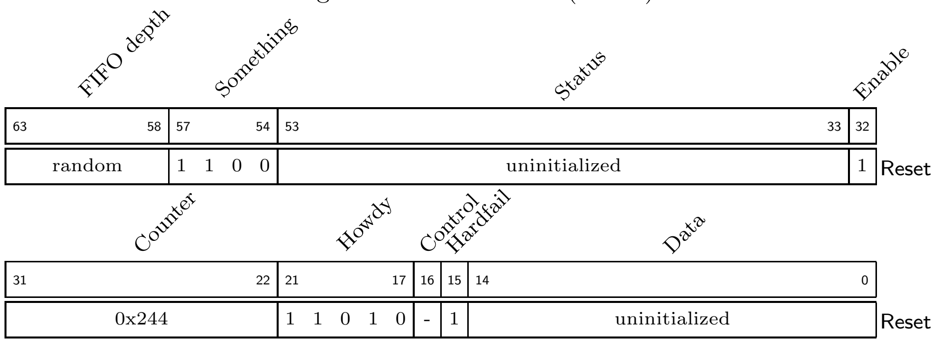


Figure 1: Address and BE phases for register access

## 3 User Interface

The user interface for register diagrams could be regarded as overly verbose. The main idea behind the parameters was to allow LaTeX *and* accompanying Perl scripts to decipher register fields and reset values. The simplest way to show the interface is via another example. Below is a 64-bit register.

Register 3.1: Example (0x250)



It was created with the following commands:

```
\begin{register}{H}{Example}{0x250}% name=example
  \label{example}%
  \regfield{FIFO depth}{6}{58}{{random}}%
  \regfield{Something}{4}{54}{1100}%
  \regfield{Status}{21}{33}{{uninitialized}}%
  \regfield{Enable}{1}{32}{1}%
  \reglabel{Reset}\regnewline%
  \regfield{Counter}{10}{22}{{0x244}}% READ_ONLY
  \regfield{Howdy}{5}{17}{1_1010}%
  \regfield{Control}{1}{16}{-}%
  \regfield{Hardfail}{1}{15}{1}%
  \regfield{Data}{15}{0}{{uninitialized}}%
  \reglabel{Reset}%\regnewline%
 \end{register}
```

`register` `\listofregisters` The register environment begins with `\begin{register}`. The three parameters to the environment are the float specification (`h`, `H`, `t`, `b`, or `p`); the register name; and the register's offset in memory space. These parameters become the register caption as well as the entry in the list of registers, which is typeset using `\listofregisters`.

`\begin{register}{`⟨*placement*⟩`}{`⟨*register name*⟩`}{`⟨*register offset*⟩`}`

`\regfield` Each register field is then typeset using `\regfield`. The fields must be specified[1] starting with the most-significant bit. The `\regfield` macro takes four parameters: field name, field length (in bits), starting bit number, and reset value. The reset value, unless you pass it as a single token (see example), will be spread apart to fill up the width of the field. The reset value, since it could be just a very long binary number, can be interrupted with underscores to improve readability.

`\regfield{`⟨*name*⟩`}{`⟨*length*⟩`}{`⟨*start bit*⟩`}{`⟨*reset value*⟩`}`

`\regBitWidth` `\regnewline` `\reglabel` The macro `\regBitWidth` specifies the maximum number of bits to be typeset on a single line. Once a line has been completed, a new line can be inserted via `\regnewline`. Before starting a new line, however, an appropriate label for the reset value should be typed. This label is generated via `\reglabel`, which takes the string to typeset as a single parameter. The package, via the macro `\regResetName`, assumes that `Reset` will be used in order to set some lengths appropriately; it really only matters for very full lines. In a future version, it may be possible to make `\regResetName` be the default parameter to `\reglabel`.

`\reglabel{`⟨*reset label*⟩`}`

As a final example, consider Register 3.2, which has field descriptions as part of the float. This diagram was created with the following code:

```
\begin{register}{htbp}{Configuration}{0x2848}% name=CONFIG
  \label{Configuration}%
  \regfield{soft reset perf}{1}{63}{0}% STATUS
  \regfield{reserved}{30}{33}{0}%
  \regfield{Test mode}{1}{32}{0}%
  \reglabel{Reset}\regnewline%
  \regfield{reserved}{13}{19}{0}%
  \regfield{\parbox[b]{0pt}{Request Depth}}{7}{12}{1}%
  \regfield{reserved}{3}{9}{0}%
  \regfield{line\_2x\_L}{1}{8}{?}% STATUS
  \regfield{reserved}{1}{7}{0}%
  \regfield{ill\_cmd\_enable}{1}{6}{0}%
  \regfield{LPCE}{1}{5}{0}%
  \regfield{DVI disable}{1}{4}{0}%
  \regfield{SBA enable}{1}{3}{0}%
  \regfield{reserved}{2}{1}{0}%
```

[1]Provided you're a little-endian person!

```
   \regfield{line\_2x\_enable}{1}{0}{0}% READ_ONLY
   \reglabel{Reset}\regnewline%
   \begin{regdesc}\begin{reglist}[Request~Depth]
    \item [line\_2x\_enable]Setting this bit enables the chip
       to utilize a second connected data line.
    \item [SBA~enable]Setting this bit activates the sideband-address port.
       The SBA port is only useable in a double-line configuration.
    \item [DVI~disable]Setting this bit \emph{turns off} DVI extraction for
       DMA requests.
    \item [LPCE]Line Parity Check Enable.
    \item [ill\_cmd\_enable]\TR{3}Illegal Command enable.
       This bit is new for TR3.
    \item [line\_2x\_L](Read only) Indicates whether this chip is connected
       to a second data line. When this bit is 0, a second line
       is available.
    \item [Request~Depth]Controls number of outstanding DMA requests.
    \item [Test~mode]Activates data line test mode.
    \item [soft~reset~perf]\TR{4}Indicates that a soft reset has been
       performed.  This bit is new for TR4.
 \end{reglist}\end{regdesc}\end{register}
```

`regdesc` `reglist` `lyxlist` This example uses the `regdesc` environment, which can either be part of a register float or used standalone. The purpose of `regdesc` is really just to turn on centering and, when the `LyX` package option is specified, redefine the `lyxlist` environment. If one is using LyX, then register field descriptions can be entered using its List type. If one is typing LaTeX directly, just use the `reglist` list definition.

The `regdesc` and `reglist` environments both take optional arguments. For `regdesc`, the optional argument specifies how wide the register description body should be. The default value is 90% of `\textwidth`. For `reglist`, the optional argument is the longest field name, which allows the list to be spaced appropriately (like examples given in *The LaTeX Companion*).

`\regfieldb` Finally, if a register with no reset values is desired, one can use the `\regfieldb` macro. It takes the same first three arguments as `\regfield`. Figure 1 was produced using `\regfieldb`.

## 3.1 TR Flags

`\TR` The last example also shows the TR flags. These marginal notes can be used to denote features that are specific to a particular release of a chip. These macros were created since LaTeX's `\marginpar` is itself a float; as such it cannot be nested inside other floats. The `\TR` macros thus implement something very similar to the solution for Problem 14.28 in *The TeXbook*.

TR flags are turned on via the `TRflags` package option. If the option is not specified, then the `\TR` macro still exists but is empty. The TR flags are placed inside an `\fbox` if the `TRboxed` package option is specified.

Register 3.2: CONFIGURATION (0x2848)

| soft reset perf | reserved | Test mode | |
|---|---|---|---|
| 63 | 62 … 33 | 32 | |
| 0 | 0 | 0 | Reset |

| reserved | Request Depth | reserved | line_2x_L | reserved | ill_cmd_enable | PCE | DVI disable | SBA enable | reserved | line_2x_enable | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 … 19 | 18 … 12 | 11 … 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 … 1 | 0 | |
| 0 | 1 | 0 | ? | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Reset |

| | | |
|---|---|---|
| **line_2x_enable** | Setting this bit enables this chip to utilize a second connected data line. | |
| **SBA enable** | Setting this bit activates the sideband-address port. The SBA port is only useable in a double-line configuration. | |
| **DVI disable** | Setting this bit *turns off* DVI extraction for DMA requests. | |
| **LPCE** | Line Parity Check Enable. | |
| **ill_cmd_enable** | Illegal OCMD enable. This bit is new for TR3. | $\boxed{\text{TR3}}$ |
| **line_2x_L** | (Read only) Indicates whether this chip is connected to a second data line. When this bit is 0, a second line is available. | |
| **Request Depth** | Controls number of outstanding DMA requests. | |
| **Test mode** | Activates data line test mode. | |
| **soft reset perf** | Indicates that a soft reset has been performed. This bit is new for TR4. | $\boxed{\text{TR4}}$ |

For two-sided documents, the TR flags are placed in the outer margins. The placement decision is made by placing labels for each TR flag and then determining even/odd page numbering in a subsequent LaTeX run.

## 4 Helper Scripts

A Perl module and a perl script are included as part of the `register` package. The module augments the package by providing the ability to parse the register macro information into Perl. This information consists of the parameters used within the macros themselves, as well as the comments which follow. If you examine the last register example, you will notice that some fields specify `READ\_ONLY` or `STATUS` in a comment. The "directives" can be used to indicate that particular fields are unaffected by writes or unaffected by reset, respectively.

The script, `reg_list.pl`, utilizes the Perl module to produce lists of registers in offset-order or to generate reset value and mask information for each register. The later data can be used in simulation to ensure that registers behave as documented.

## 5 Implementation

### 5.1 Identification & Options

The `register` document class can only be used with LaTeX $2_\varepsilon$, so we make sure that an appropriate message is displayed when another TeX format is used.

```
⟨*package⟩\NeedsTeXFormat{LaTeX2e}
```

Announce the name and load required packages:

```
\ProvidesPackage{register} [2007/03/08 v1.5 Register macros with
hyperref/LyX support]

\RequirePackage{ifthen}[1997/11/02]
\RequirePackage{graphicx}[1997/06/09]
\RequirePackage{float}[2001/07/25]
\RequirePackage{calc}[1998/06/07]
```

Declare the package options and some booleans. There are options indicating whether `register` is being used with LyX or `hyperref`. The marginal TR flags can also be turned on or off with an option.

```
\DeclareOption{LyX}{\setboolean{RegisterLyX}{true}}
\DeclareOption{hyperref}{\setboolean{RegisterHyperref}{true}}
\DeclareOption{TRflags}{\setboolean{RegisterTRFlags}{true}}
\DeclareOption{TRboxed}{\setboolean{RegisterTRBoxed}{true}}

\DeclareOption*{%  Emit a warning for other options
  \PackageWarning{register}{Unknown option '\CurrentOption'}%
}

\newboolean{RegisterLyX}
\newboolean{RegisterHyperref}
\newboolean{RegisterTRFlags}
\newboolean{RegisterTRBoxed}

\setboolean{RegisterLyX}{false}
\setboolean{RegisterHyperref}{false}
\setboolean{RegisterTRFlags}{false}
\setboolean{RegisterTRBoxed}{false}

\ProcessOptions\relax  % Process package options
```

Also, provide a flag indicating when one is inside a register context. Some users may wish to define macros, for example, which make use of `\marginpar`. Those cannot be used inside of a floating register. So, this flag is provided to provide something to test against.

```
\newboolean{RegisterContext}
\setboolean{RegisterContext}{false}
```

## 5.2 Float Declaration & Lengths

Declare the new register float type and the lengths which the macros will use. The register diagrams tend to look best with the caption at the top of the float. If `\chapter` is defined in the current document class, then the register diagrams will count by chapter, otherwise they will count by section.

```
\floatstyle{plaintop} \@ifundefined{chapter}
{\newfloat{Regfloat}{tbp}{rdf}[section]}
{\newfloat{Regfloat}{tbp}{rdf}[chapter]}
\floatname{Regfloat}{Register}
```

## 5.3 Style Parameters

`\regWidth`
`\regBitWidth`

The `\regWidth` length controls how physically wide the register diagrams are, while `\regBitWidth` specifies how the maximum number of bits per line in the diagram. The default value for `\regWidth` is 95% of the width of the main text body.

`\regResetName`
`\regBitSize`
`\regBitFamily`
`\regResetSize`
`\regLabelSize`
`\regLabelFamily`
`\regDescFamily`

The string with which to label reset values is `\regResetName`. The `\regBitSize` command controls the font size used for bit values, `\regResetSize` controls the font size used for reset values, and `\regBitFamily` controls which font family is used to denote bit positions. The font size for field names and for the `\regResetName` is controlled by `\regLabelSize`. Finally, `\regDescFamily` controls the appearance of field names within register descriptions.

```
\newlength{\regWidth}
\newlength{\regFieldLen}
\newlength{\regLabelAdjust}
\newlength{\regResetHeight}
\newlength{\regResetDepth}
\newlength{\regResetDrop}
\newlength{\regDescSkip}
\setlength{\regWidth}{0.95\textwidth}
\newcommand{\regBitWidth}{32}
\newcommand{\regResetName}{Reset}
\newcommand{\regBitSize}{\tiny}
\newcommand{\regBitFamily}{\sffamily}
\newcommand{\regResetSize}{\scriptsize}
\newcommand{\regLabelSize}{\footnotesize}
\newcommand{\regLabelFamily}{\rmfamily}
\newcommand{\regDescFamily}{\bf}
```

The lengths below control the vertical spacing between a register diagrams and the register field descriptions. This distance must change when the `\regdesc` is part of the `\Regfloat` or standalone.

```
\newlength{\regdescsep}
\newlength{\oldregdescsep}
\setlength{\regdescsep}{-\medskipamount}

```

```
\newsavebox{\Label}
\newsavebox{\RotatedLabel}
\newcounter{upperbit}
\newcounter{lowerbit}
```

## 5.4 Register macros

`\reglist` Define a new environment for register field descriptions, beneath a register diagram. These descriptions can indicate what a particular field is used for, whether it is read-only, etc. The field descriptions themselves are in a `\reglist` list environment, unless LyX is being used.

```
\newenvironment{reglist}[1][M]
  {\begin{list}{}
    {\settowidth{\labelwidth}{\regDescFamily #1}
     \addtolength{\labelwidth}{\labelsep}
     \setlength{\leftmargin}{\labelwidth}
     \addtolength{\leftmargin}{\labelsep}
     \addtolength{\leftmargin}{0.5\regDescSkip}
     \addtolength{\rightmargin}{0.5\regDescSkip}
     \setlength{\topsep}{0pt}
     \setlength{\itemsep}{0pt}
     \setlength{\parsep}{0.5\baselineskip}
     \renewcommand{\makelabel}[1]{\regDescFamily ##1 \hfill}}}
  {\end{list}}
```

`\regdesc` Next is the `\regdesc` environment. If LyX is being used, the environment temporarily redefines the `lyxlist` environment, which is what LyX provides as the default list structure. Otherwise, the user is left to use `\reglist`. The redefinition should look very similar to that for `\reglist`.

In all cases, `\regdesc` starts by attempting to place a small amount of vertical space between it and the presumed register diagram immediately above it.

```
\newenvironment{regdesc}[1][0.90\textwidth]%
% --------------------------------------------
{%
 \setlength{\regDescSkip}{\textwidth - #1}%
 \vspace{\regdescsep}%
\ifthenelse{\boolean{RegisterLyX}}{%
  \renewenvironment{lyxlist}[1]
    {\begin{list}{}
      {\settowidth{\labelwidth}{\regDescFamily ##1}
       \addtolength{\labelwidth}{\labelsep}
       \setlength{\leftmargin}{\labelwidth}
       \addtolength{\leftmargin}{\labelsep}
       \addtolength{\leftmargin}{0.5\regDescSkip}
       \addtolength{\rightmargin}{0.5\regDescSkip}
       \setlength{\topsep}{0pt}
       %\setlength{\partopsep}{0pt}
       \renewcommand{\makelabel}[1]{\regDescFamily ####1 \hfill}}}
    {\end{list}}}{}%
 % endif
 % set spacing appropriately
 \leftskip 0.5\regDescSkip%
 \rightskip 0.5\regDescSkip%
 \parfillskip=\z@ plus 1fil%
 \parskip=0.5\baselineskip \advance\parskip by 0pt plus 2pt%
}% end begin{regdesc}
% ------------------------------------------
{\vskip\baselineskip}
```

`\regnewline` This command allows a register diagram to be broken into multiple lines

```
\newcommand{\regnewline}{\\*}
```

`\register` This environment declares the floating environment in which a register diagram and, optionally, its field descriptions can be typeset. It takes as arguments the float type (h, t, p, H); the register name; and the register offset/address.

```
\newenvironment{register}[3]
{\begin{Regfloat}[#1]%
  \setlength{\leftskip}{0pt}%
  \setlength{\oldregdescsep}{\regdescsep}%
  \setlength{\regdescsep}{0.5\baselineskip}%
  \setlength{\partopsep}{0pt}%
  \setlength{\topsep}{0pt}%
  \setboolean{RegisterContext}{true}%
  \ifthenelse{\equal{#3}{}}%
  {\caption[#2]{\textsc{#2}}}% else
  {\caption[#2]{\textsc{#2} ({#3})}}%
  \centering}
{% restore lengths
  \leftskip\z@%
  \rightskip\z@%
  \parfillskip=\z@ plus 1fil%
  \setlength{\regdescsep}{\oldregdescsep}%
  \setboolean{RegisterContext}{false}%
  \end{Regfloat}}
```

`\regUnderScore` `\regFiller` `\regSpread` The next definitions provide a method of spreading out an argument, i.e., placing `\hfill`'s between each character. They can be used to space the reset value of a register field appropriately. Previous versions of these macros used TEX's conditionals incorrectly.

If you are using the `underscore` package, you should *repeat* the definition of `\regUnderScore`, after loading `underscore` in order to observe the changes made by that package. The spreading macros below strive to strip out underscores.

```
\def\regUnderScore{_}%
\def\regFiller#1{\def\regInner{#1}%
\ifx\regInner\regUnderScore%
\else%
\ifnum\count0>0%
\hfill#1%
\else#1\fi%
\fi%
\advance\count0 by 1%
}
\def\regSpread#1{\count0=0{}\regSpreadaux#1\empty}
\def\regSpreadaux#1#2\empty{\def\aux{#1}%\show#1%
\ifx\aux\empty%
\else%
\def\aux{#2}%
\regFiller{#1}%
\ifx\aux\empty%
\else%
\regSpreadaux#2\empty%
\fi\fi}
```

### 5.4.1 Register fields

Define some internal utility macros which get called repeatedly. These macros figure out some of the dimensions of the current font (is there a better way to do this?), construct and rotate label boxes, and typeset bit positions.

```
\newcommand{\setRegLengths}{%
  % Compute basic width of a single bit
  \settowidth{\regFieldLen}{\regLabelSize \regResetName}%
  \setlength{\regFieldLen}{\regWidth - \regFieldLen}%
  \setlength{\regFieldLen}{\regFieldLen / \regBitWidth}%
  % Figure out height and depth of reset field in current font
  % Is there a more ‘‘official’’ method to do this?
  \settodepth{\regResetDepth}{\regResetSize ()jgpq}%
  \settoheight{\regResetHeight}{\regResetSize ()ABCjkl}%
  \addtolength{\regResetHeight}{\regResetDepth}%
  % Compute how far to drop the reset fields down.  The value at
  % the end is effectively the separation between the bit position
  % box and the reset value box.
  \setlength{\regResetDrop}{\regResetHeight + 2\fboxsep - 2\fboxrule + 3pt}%
}
```

These macros assembly, rotate, and typset the register field name. It is the use of `\rotatebox` below which makes `register` require PostScript processing.

```
\newcommand{\regMakeFieldName}[1]{%
  % Create box to hold label
  \savebox{\Label}{\regLabelSize\regLabelFamily #1}%
}
\newcommand{\regRotateFieldName}{%
  \savebox{\RotatedLabel}{\rotatebox[origin=lB]{45}{\usebox{\Label}}}%
  \makebox[0pt][l]{\raisebox{\regResetHeight + \fboxsep + \depth + 1pt}%
        {\hspace{\regLabelAdjust}\usebox{\RotatedLabel}}}%
}
```

The `\typesetRegBits` macro constructs the frame for a particular register field and sets the starting and ending bit positions within that frame.

```
\newcommand{\typesetRegBits}[1]{%
  \ifthenelse{#1 > 1}%
    {\framebox[\regFieldLen][c]%
        {\regBitSize \rule[-1\regResetDepth]{0pt}{\regResetHeight}%
           \regBitFamily\arabic{upperbit} \hfill \arabic{lowerbit}}}%
    {\framebox[\regFieldLen][c]%
        {\regBitSize \rule[-1\regResetDepth]{0pt}{\regResetHeight}%
           \regBitFamily\arabic{lowerbit}}}%
}
```

This macro constructs a frame, below the bit position frame, displaying the reset (or some other) value for a register field.

```
\newcommand{\typesetRegReset}[1]{%
 % Typeset reset value in a framebox
  \makebox[0pt][l]{\raisebox{-1\regResetDrop}{\framebox[\regFieldLen][c]%
        % Place an invisible rule to control the box
        % surrounding the reset field
        {\regResetSize \rule[-1\regResetDepth]{0pt}{\regResetHeight}\regSpread{#1}}}}%
}
```

`\regfield` The macros below are the user interface to the register drawing routines. The first of these, `\regfield` takes four arguments: the field name, field length (in bits), the starting bit number, and the field's reset value.

```
\newcommand{\regfield}[4]{%
  % Compute overall field length
  \setRegLengths%
  \setlength{\regFieldLen}{#2\regFieldLen + \fboxrule}%
  % Figure out bit positions
  \setcounter{lowerbit}{#3}%
  \setcounter{upperbit}{#3 + #2 - 1}%
  \regMakeFieldName{#1}%
  % Figure out how far over to place label, accounting for height
  \setlength{\regLabelAdjust}{0.5\regFieldLen - 0.707107\ht\Label}%
  % Now, rotate and type the label
  \regRotateFieldName%
  \typesetRegReset{#4}%
  % Typeset bit positions in a framebox
  \typesetRegBits{#2}%
  \hspace{-1\fboxrule}%
}
```

`\regfieldb` The `\regfieldb` macro can be used to construct a register diagram without reset values. It's parameters are thus the same as for `\regfield` with the omission of reset value.

```
\newcommand{\regfieldb}[3]{%
```

```
  % Compute overall field length
  \setRegLengths%
  \setlength{\regFieldLen}{#2\regFieldLen + \fboxrule}%
  % Figure out bit positions
  \setcounter{lowerbit}{#3}%
  \setcounter{upperbit}{#3 + #2 - 1}%
  % Create box to hold label
  \regMakeFieldName{#1}%
  % Figure out how far over to place label, accounting for height
  \setlength{\regLabelAdjust}{0.5\regFieldLen - 0.707107\ht\Label}%
  % Now, rotate and typeset the label
  \regRotateFieldName%
  % Typeset bit positions
  \typesetRegBits{#2}%
  \hspace{-1\fboxrule}%
}
```

`\regbits` The `\regbits` macro typesets a register field without showing bit positions. As such, it only takes three parameters: field name, field bit length, and field value.

```
\newcommand{\regbits}[3]{%
  % Compute overall field length
  \setRegLengths%
  \setlength{\regFieldLen}{#2\regFieldLen + \fboxrule}%
  % Figure out bit positions
  \setcounter{lowerbit}{#3}%
  \setcounter{upperbit}{#3 + #2 - 1}%
  % Create box to hold label
  \regMakeFieldName{#1}%
  % Figure out how far over to place label, accounting for height
  \setlength{\regLabelAdjust}{0.5\regFieldLen - 0.707107\ht\Label}%
  % Now, rotate and typeset the label
  \regRotateFieldName%
  % Typeset field value
  \framebox[\regFieldLen][c]%
        {\tiny\regSpread{#3}}%
  \hspace{-1\fboxrule}%
}
```

`\regspace` `\reglabel` `\reglabelb` Finally, define some additional utility macros. An invisible box with the same width as a specified number of bits is typeset by `\regspace`. It takes as its sole parameter the desired bitwidth. The `\reglabel` macro typesets the label for the reset field in register diagrams. Finally, `\reglabelb` performs the same function, but without any drop.

```
\newcommand{\regspace}[1]{%
  % Compute overall field length
  \setRegLengths%
  \setlength{\regFieldLen}{#1\regFieldLen + \fboxrule}%
  \makebox[\regFieldLen]{}%
}

\newcommand{\reglabel}[1]{%
  \settowidth{\regFieldLen}{\regLabelSize \regResetName}%
  \setlength{\regResetDrop}{\regResetDrop + 0.5\fboxsep}%
  $\,$\raisebox{-1\regResetDrop}{\makebox[\regFieldLen][l]%
    {\regLabelSize\regBitFamily #1}}%
}

\newcommand{\reglabelb}[1]{%
  \settowidth{\regFieldLen}{\regLabelSize \regResetName}%
  $\,$\raisebox{-0.5\fboxsep}{\makebox[\regFieldLen][l]%
    {\regLabelSize\regBitFamily #1}}%
}
```

### 5.4.2 Hyperref Interface

This code helps with the hyperlinks to register floats when producing linked PDF.

```
\ifthenelse{\boolean{RegisterHyperref}}{%
  % Define a counter for the hyperref package.  Otherwise,
  % the hyperlinks to registers don't work correctly
  \@namedef{theHRegfloat}{\theRegfloat}

  % Define a bookmark level for Regfloats (for hyperref package)
  \def\toclevel@Regfloat{0}
}{}
```

### 5.4.3 List of Registers

The code below redefines what the `float` package specifies in order to provide a hook for changing the format of the ToC line for registers.

```
\newcommand{\listofregisters}{%
  \@ifundefined{ext@Regfloat}{\float@error{Regfloat}}{%
    \@ifundefined{chapter}{\def\@tempa{\section*}}%
      {\def\@tempa{\chapter*}}%
    \@tempa{List of Registers\@mkboth{\uppercase{List of Registers}}%
       {\uppercase{List of Registers}}}%
    \addcontentsline{toc}{chapter}{List of Registers}%
    \@starttoc{\@nameuse{ext@Regfloat}}}}
\newcommand\l@Regfloat{\@dottedtocline{1}{1.5em}{2.3em}}
```

The `\chapter` command, if present, is also redefined in order to get identical chapter-related spacing in the list of registers as appears in the LoF and LoT.

```
\@ifundefined{chapter}{}
{% Adjust chapter definition slightly for Regfloats
\def\@chapter[#1]#2{\ifnum \c@secnumdepth >\m@ne
  \if@mainmatter
    \refstepcounter{chapter}%
    \typeout{\@chapapp\space\thechapter.}%
    \addcontentsline{toc}{chapter}%
    {\protect\numberline{\thechapter}#1}%
  \else
    \addcontentsline{toc}{chapter}{#1}%
  \fi
  \else
    \addcontentsline{toc}{chapter}{#1}%
  \fi
  \chaptermark{#1}%
    \addtocontents{lof}{\protect\addvspace{10\p@}}%
    \addtocontents{lot}{\protect\addvspace{10\p@}}%
    \addtocontents{rdf}{\protect\addvspace{10\p@}}% --- Add space ---
  \if@twocolumn
    \@topnewpage[\@makechapterhead{#2}]%
  \else
    \@makechapterhead{#2}%
    \@afterheading
  \fi}
}
```

## 5.5 TR flag macros

Define a mechanism for including TR flags in register descriptions, as well as possibly in the main body of the text. One cannot just use `\marginpar`'s within a register description, since they are typically part of a register float. LaTeX's `\marginpar` command is *also* considered a float, so they cannot be nested.

So, we resort to using some lower level LaTeX. This certainly isn't the most elegant way to get the flags, but I think it will work for the majority of cases.

The `\TR{x}` command is intended to be placed at the start of a register field description, so that the alignment is obvious on the page. To also get decent alignment in main body text, it is necessary to have a second definition. So, we'll see these commands get redefined inside regdesc, below.

The TR flags also decide which page they are on by placing labels and testing the resulting pagenumber. This enables the flags to be placed in the outer margin in two-sided documents.

`\TRfamily` `\TRwidth` The font family used for the TR flags can be controlled via `\TRfamily`. The width available for TR flags is controlled by `\TRwidth`, which by default is set to `\marginparwidth`. Setting the width may be important if you are using the `TRboxed` package option.

```
% Define a pageref which will work with \isodd
\newcommand\@GetTRSecondParam{}
\long\def\@GetTRSecondParam#1#2#3\@nil{#2}
\newcommand*{\GetTRPageRef}[1]{%
  \expandafter\expandafter\expandafter\@GetTRSecondParam
    \csname r@#1\endcsname
    0% dummy, if the reference is undefined
    \@nil
}

\newcommand{\TRfamily}{\sffamily}
\newcounter{T@peReleaseTag}
\newlength{\TRwidth}
\setlength{\TRwidth}{\marginparwidth}
\newlength{\T@peReleaseDepth}

% Internal command to form the actual TR margin note.
\newcommand{\TRwriteout}[1]{%
  \makebox[\TRwidth][c]{%
    \raisebox{\T@peReleaseDepth}{%
     \ifthenelse{\boolean{RegisterTRBoxed}}%
       {\fbox{\TRfamily #1}}%
       {\TRfamily #1}}}%
}

% Internal command to typeset a TR flag in the right margin
\newcommand{\TRrightlabel}[1]{%
  % Place a strut in order to set line depth
  \mbox{}\strut\settodepth{\T@peReleaseDepth}{\strut}%
   \vadjust{\hspace{\textwidth}\hspace{\marginparsep}%
   \smash{\rlap{\TRwriteout{#1}}}}}

% Internal command to typeset a TR flag in the left margin
\ifthenelse{\boolean{@twoside}}{
  % Two-sided document
  \newcommand{\TRleftlabel}[1]{%
    % Place a strut in order to set line depth
    \mbox{}\strut\settodepth{\T@peReleaseDepth}{\strut}%
    \vadjust{\smash{\llap{\TRwriteout{#1}}\kern\marginparsep}}}
}{
  % Otherwise, the command is the same as rightlabel
  \newcommand{\TRleftlabel}[1]{%
    \TRrightlabel{#1}}
}
```

`\TR` Finally, the user-level command is simply `\TR`.

```
\ifthenelse{\boolean{RegisterTRFlags}}{
\newcommand{\TR}[1]{%
  \stepcounter{T@peReleaseTag}%
  \label{TapeReleaseTag-\theT@peReleaseTag}%
  \ifthenelse{\isodd{\GetTRPageRef{TapeReleaseTag-\theT@peReleaseTag}}}%
   {\TRrightlabel{TR\hspace{1pt}#1}}%
   {\TRleftlabel{TR\hspace{1pt}#1}}%
}}
{\newcommand{\TR}[1]{}}
```