Control & Status Registers

Copyright (c) 2010-2016 Young W. Lim.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice.

Based on

Chuck Benz ASIC and FPGA Design

csrGen - generate verilog RTL code
for processor memory maps in ASIC/FPGA designs

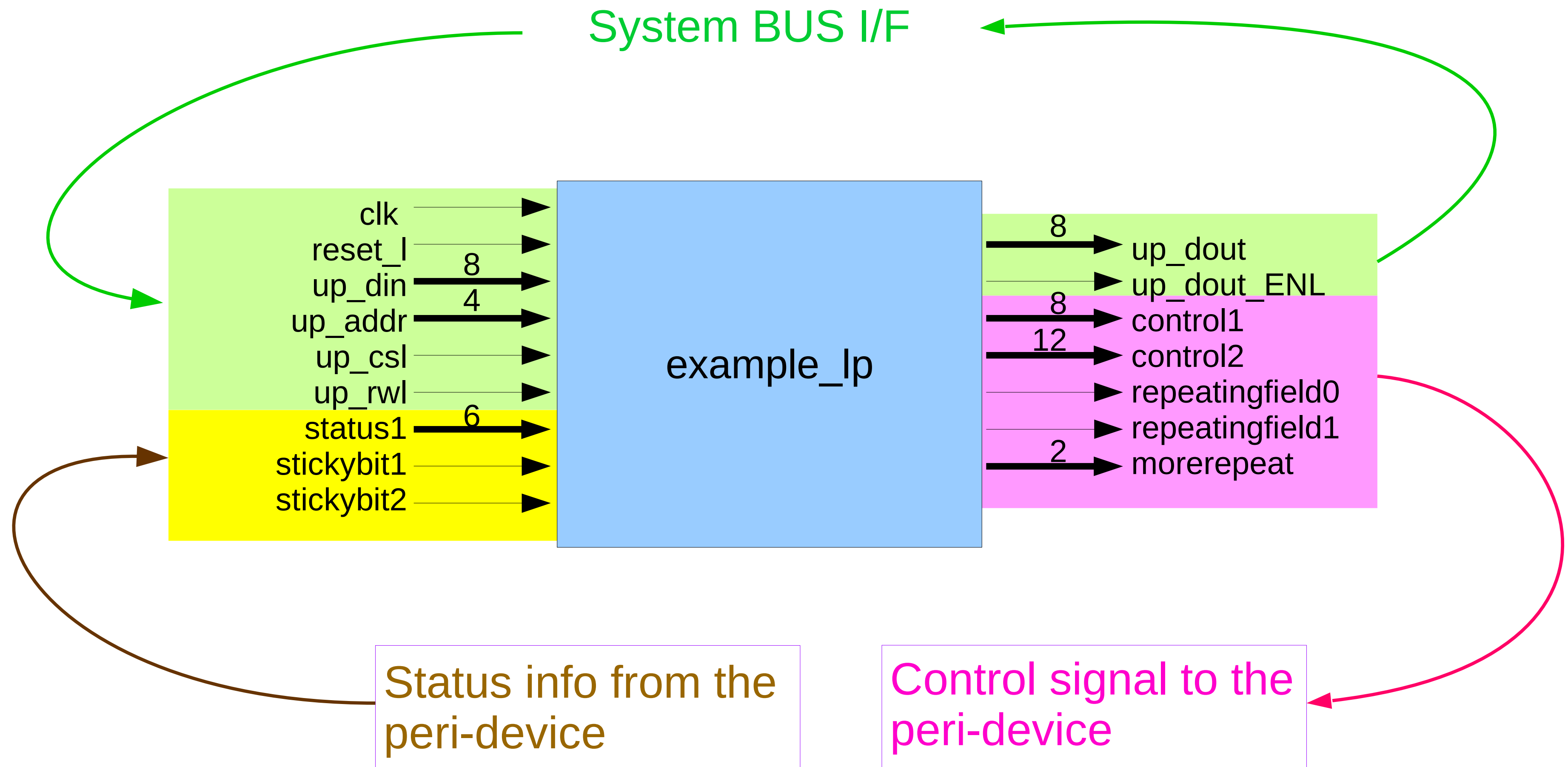
http://asics.chuckbenz.com/

Module Interface

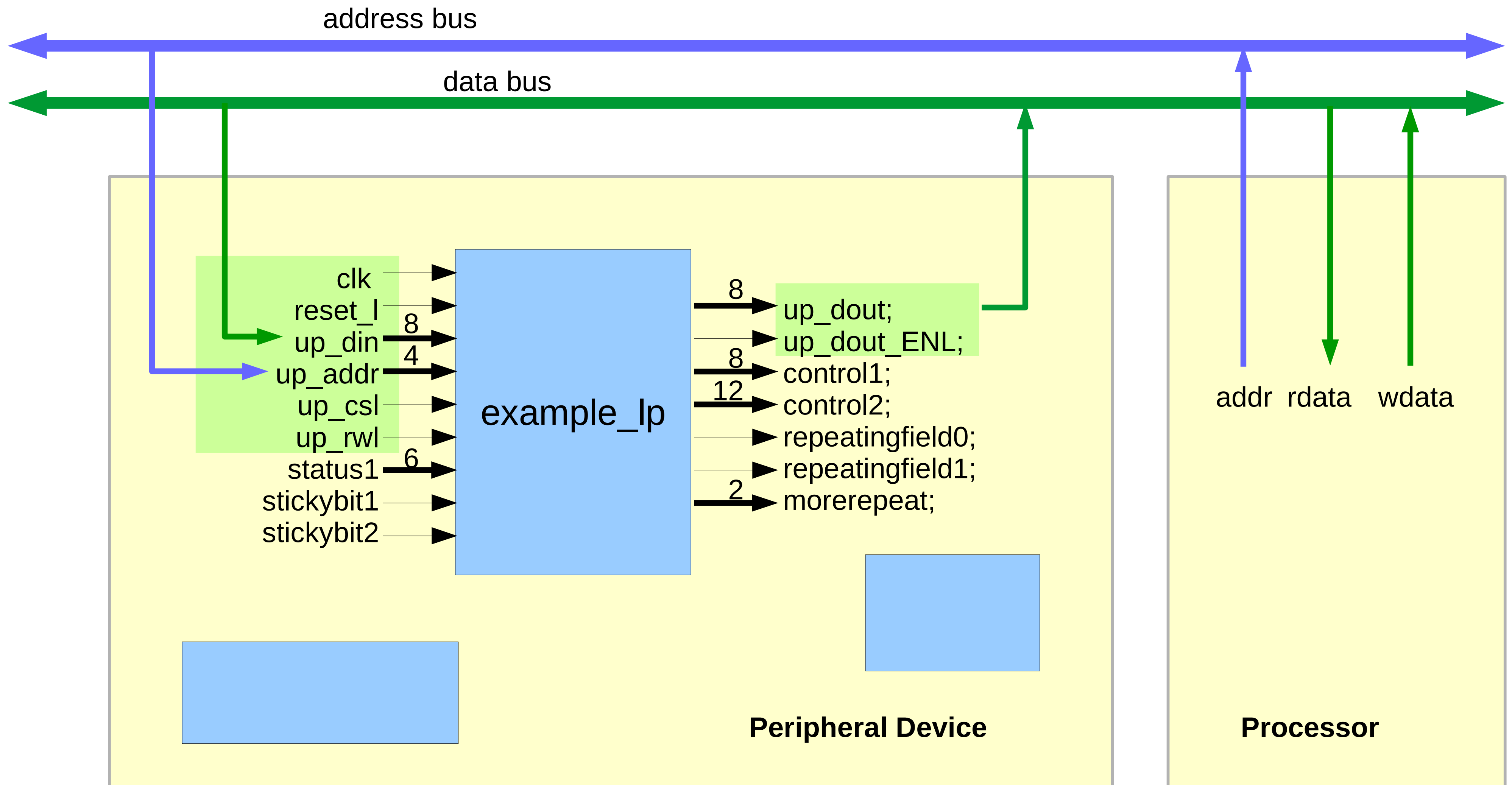
```
module example_lp (clk, reset_l
                   ,up_din
                   ,up_addr
                   ,up_csl
                   ,up_rwl
                   ,up_dout
                   ,up_dout_ENL
                   ,control1
                   ,control2
                   ,status1
                   ,stickybit1
                   ,stickybit2
                   ,repeatingfield0
                   ,repeatingfield1
                   ,morerepeat
               ) ;
```

```
input             clk, reset_l ;
input             up_din;
input    [3:0]    up_addr;
input             up_csl;
input             up_rwl;
input    [5:0]    status1;
input             stickybit1;
input             stickybit2;

output   [7:0]    up_dout;
output            up_dout_ENL;
output   [7:0]    control1;
output   [11:0]   control2;
output            repeatingfield0;
output            repeatingfield1;
output   [1:0]    morerepeat;

reg      [7:0]    up_dout;
reg               up_dout_ENL;
reg      [7:0]    control1;
reg      [11:0]   control2;
reg               repeatingfield0;
reg               repeatingfield1;
reg      [1:0]    morerepeat;
```

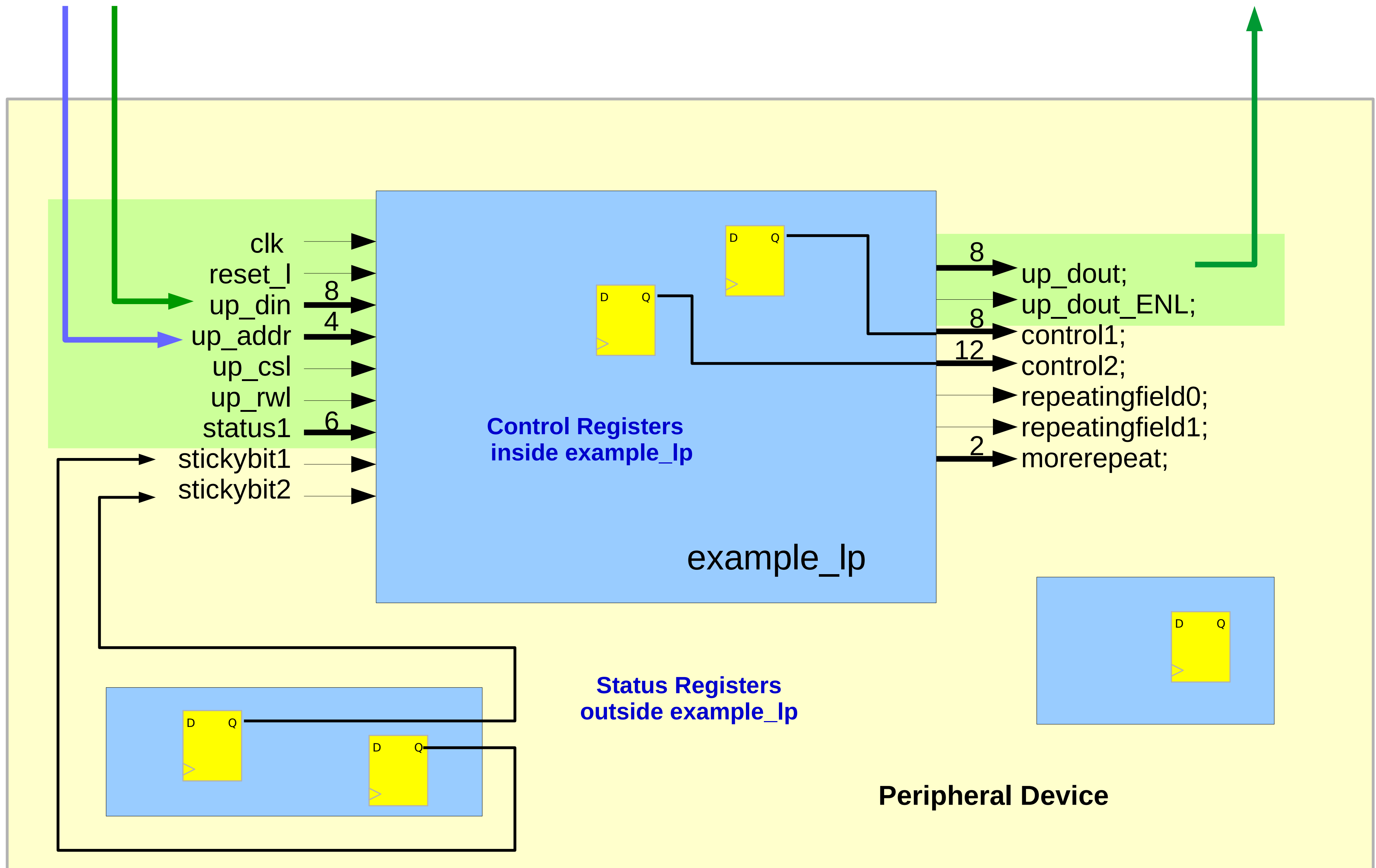
Module Interface



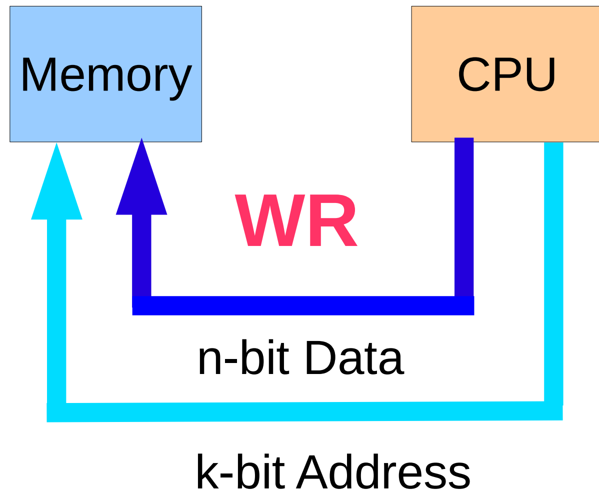
Module Interface

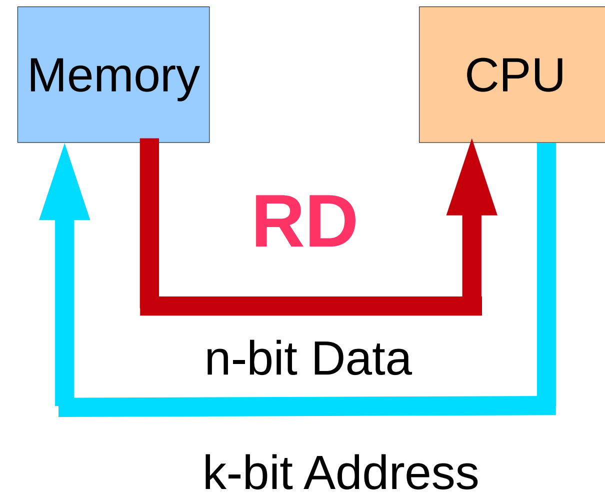


Control Registers and Status Registers

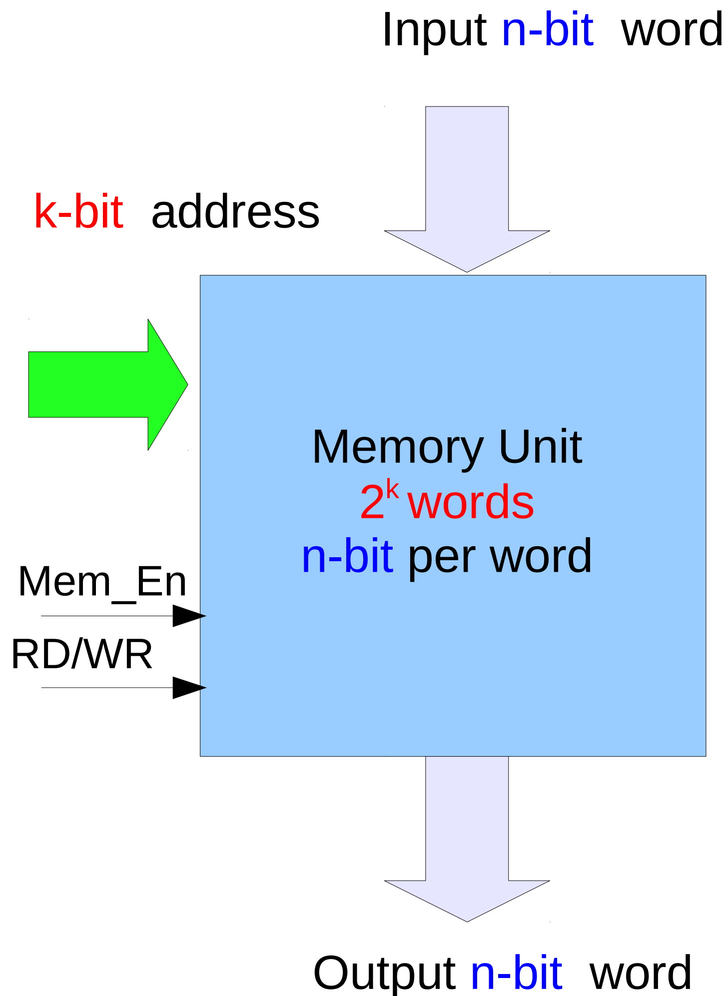


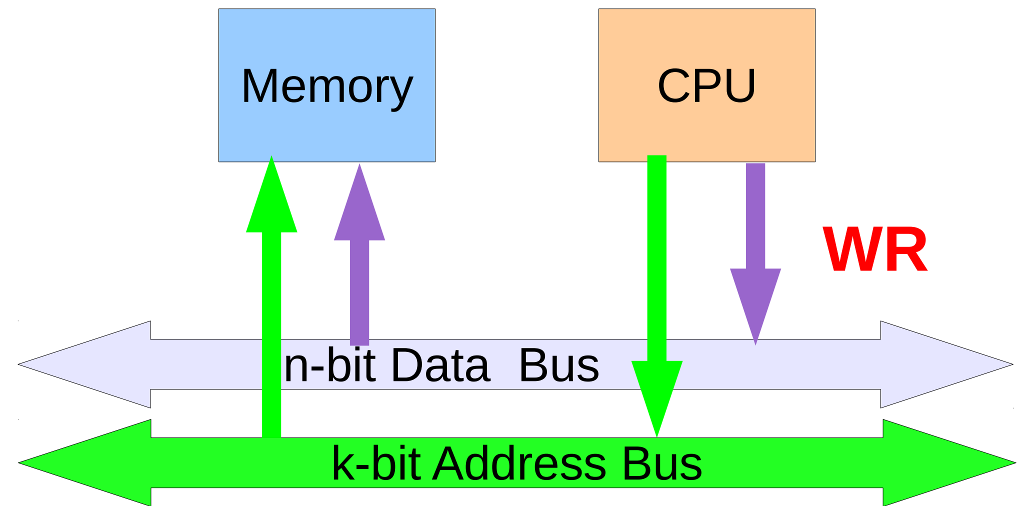
Memory Access Operations

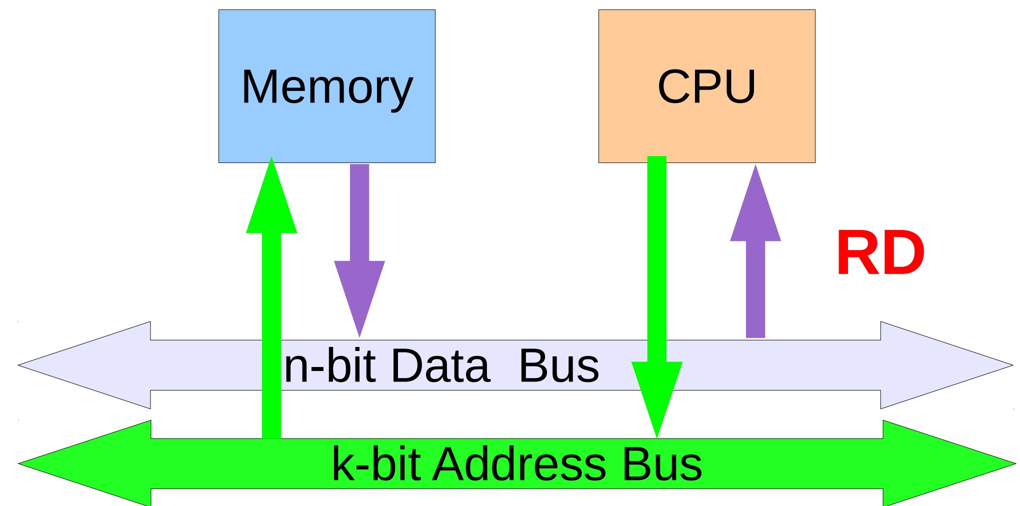




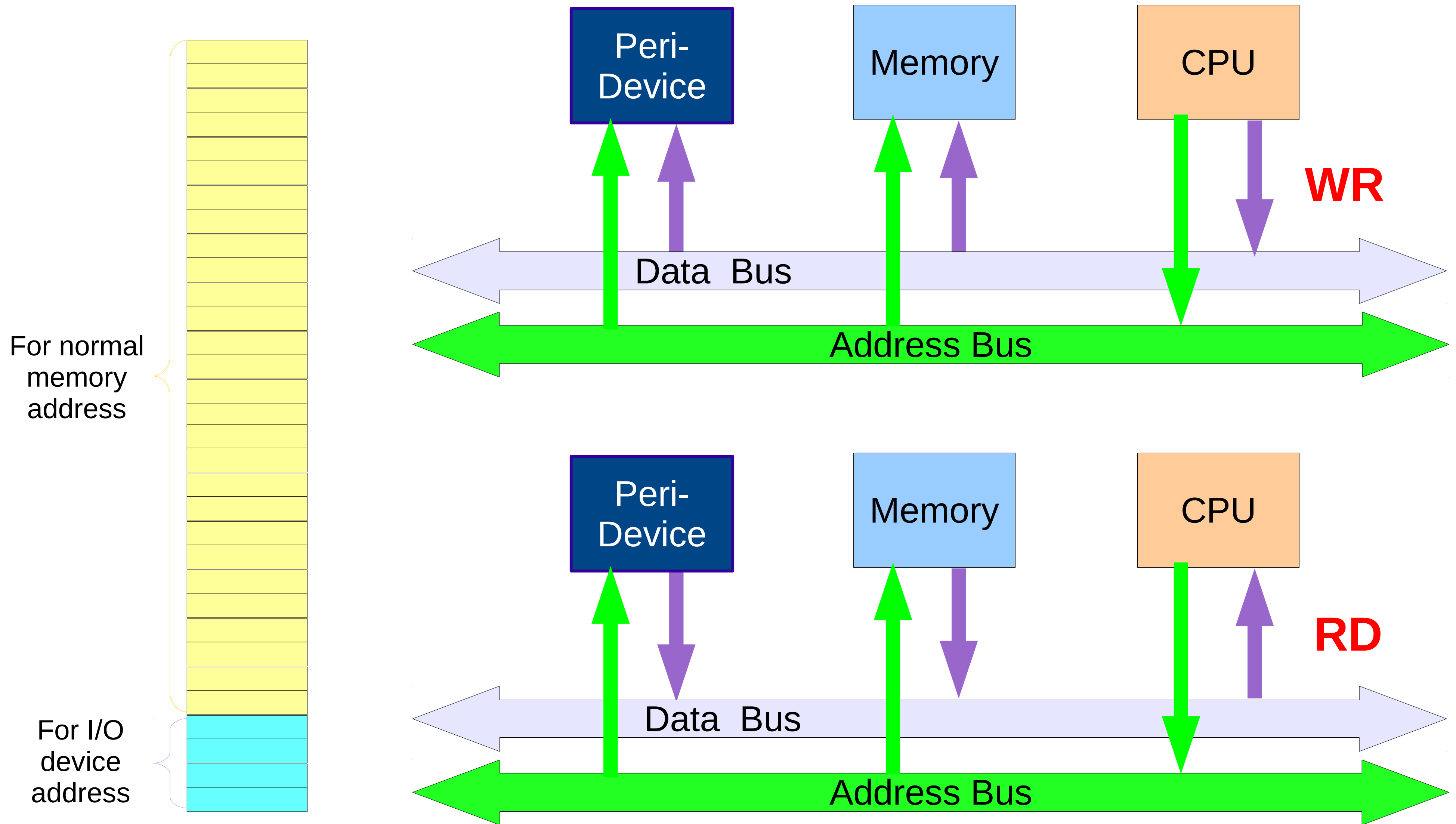
Memory RD & WR Operations



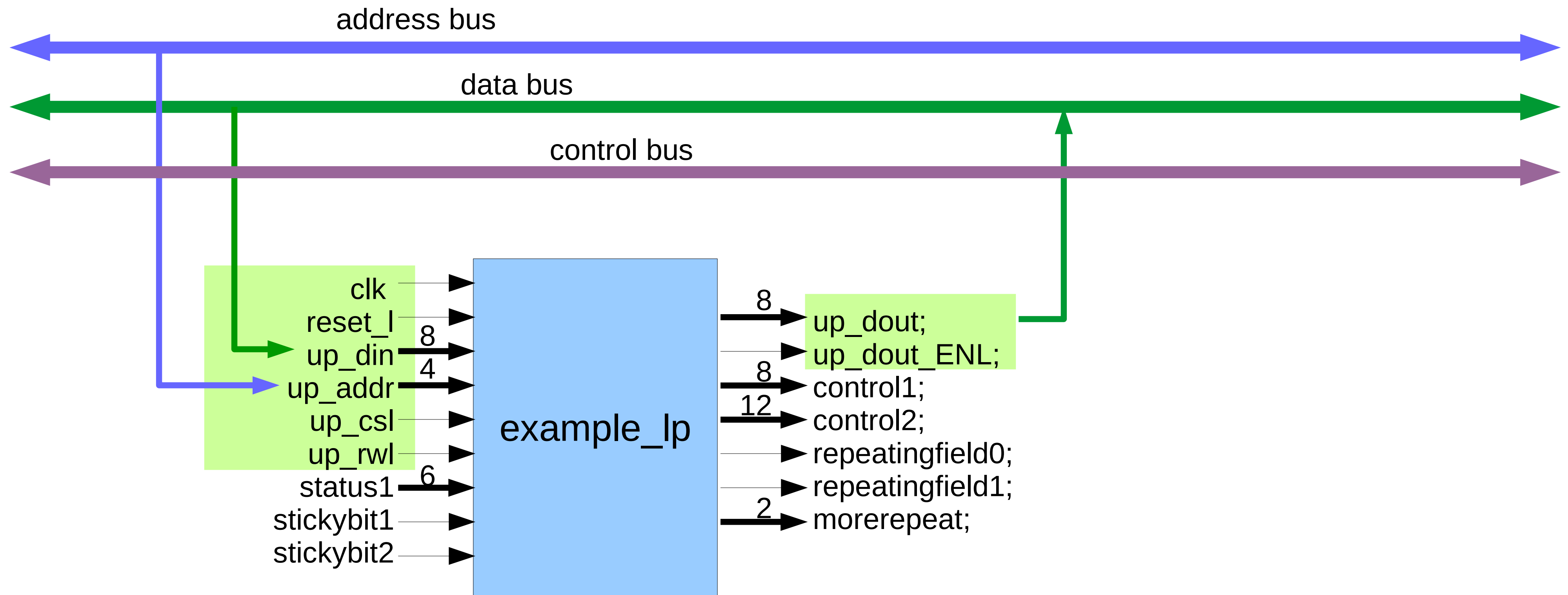




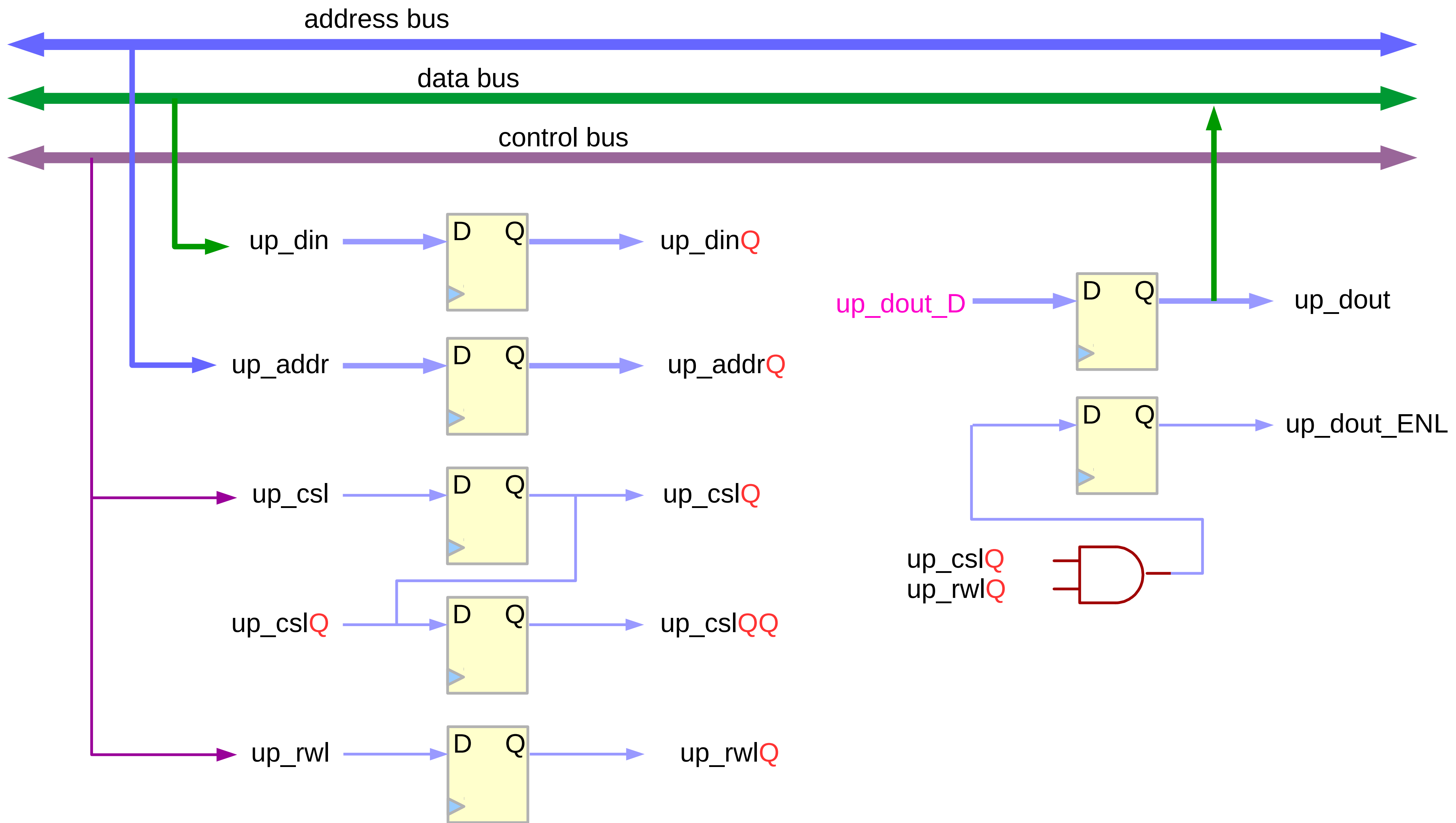
Memory-mapped IO Operations



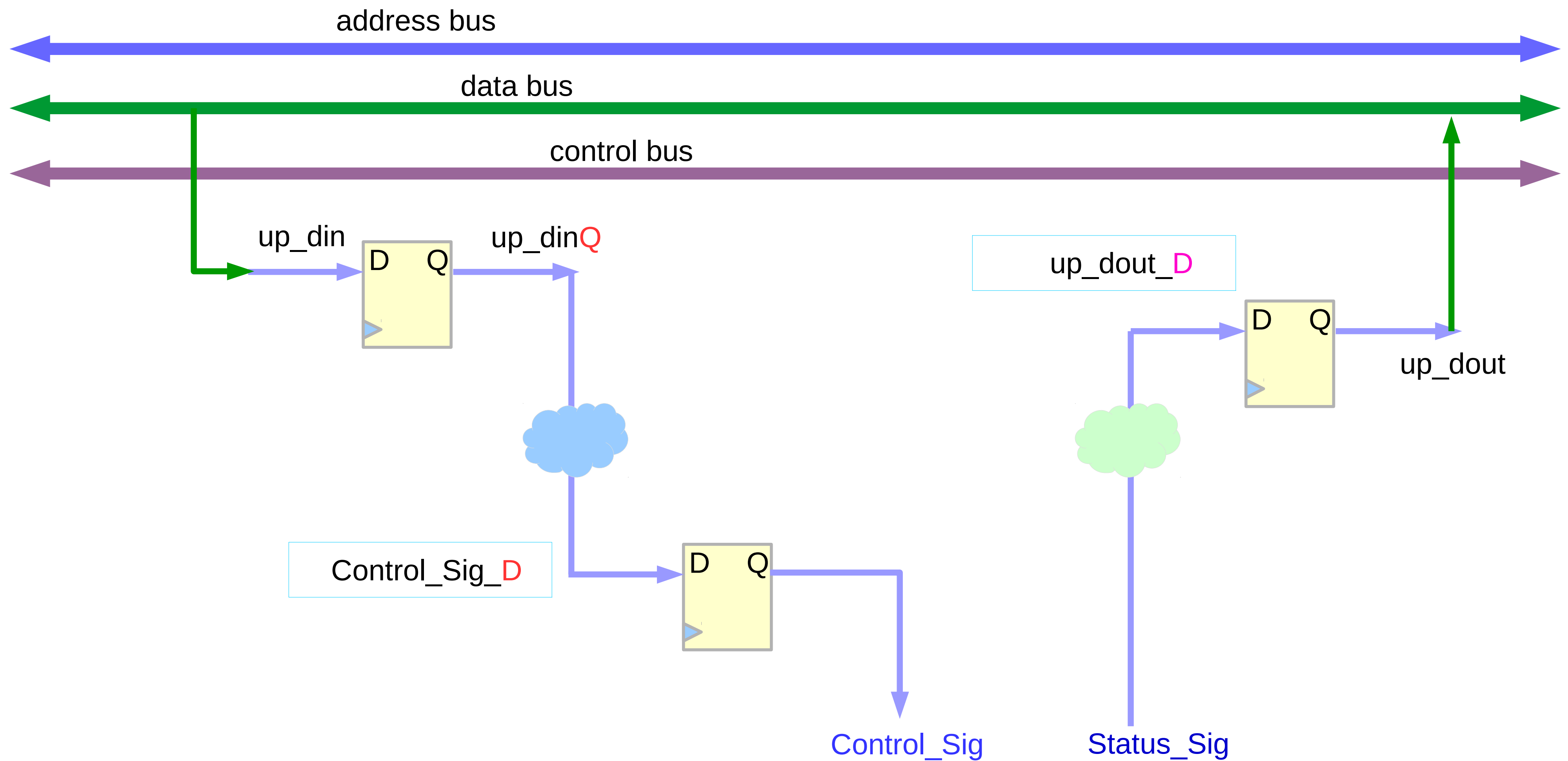
System Bus Interface



System Bus Registers



System Bus Data Registers



Combinational Logic Block

Sensitivity Lists

```
always @ (up_din
          or up_addr
          or up_csl
          or up_rwl
          or status1
          or stickybit1
          or stickybit2
          or up_dinQ
          or up_addrQ
          or up_cslQ
          or up_cslQQ
          or up_rwlQ
          or up_dout
          or up_dout_ENL
          or version
          or deviceID
          or control1
          or control2
          or stickybit1S
          or stickybit2S
          or repeatingfield0
          or repeatingfield1
          or morerepeat
         ) begin
```

default assignments

```
up_dout_D          = up_dout ;
control1_D         = control1 ;
control2_D         = control2 ;
stickybit1S_D      = stickybit1S | stickybit1 ;
stickybit2S_D      = stickybit2S | stickybit2 ;
repeatingfield0_D = repeatingfield0 ;
repeatingfield1_D = repeatingfield1 ;
morerepeat_D       = morerepeat ;
```

Control_Sig <= Control_Sig_D <= up_dinQ

up_dout <= up_dout_D <= Status_Sig

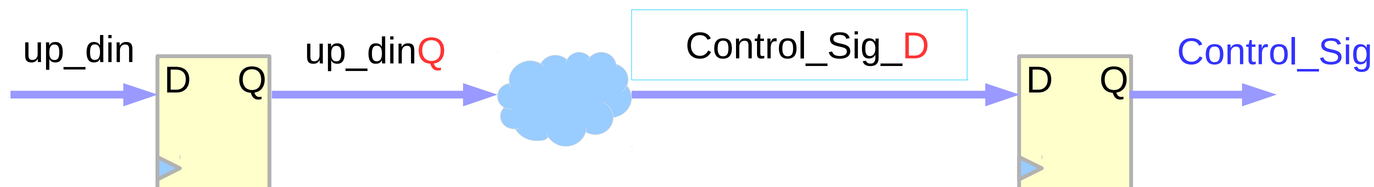
Reg Signal

```
reg      [3:0]    version ;
reg      [3:0]    deviceID ;
reg      [7:0]    up_dout_D ;
reg      [7:0]    control1_D;
reg      [11:0]   control2_D;
reg               stickybit1S, stickybit1S_D;
reg               stickybit2S, stickybit2S_D;
reg               repeatingfield0_D;
reg               repeatingfield1_D;
reg      [1:0]    morerepeat_D;
```

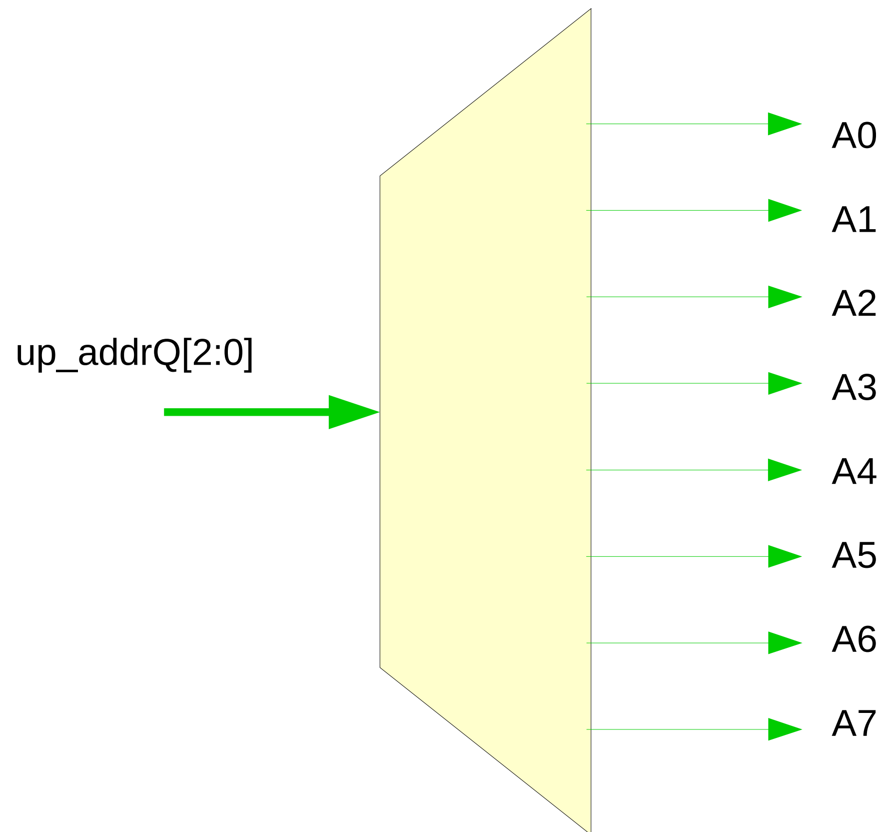
Control register input signals

Control_Sig_D <= up_dinQ

```
   // Compute FF input signals for WRITE
if (up_cslQQ & ! up_cslQ & ! up_rwlQ)
   case (up_addrQ)
     0: begin // 0x0
        end
     1: begin // 0x1
          control1_D = up_dinQ[7:0] ;
        end
     2: begin // 0x2
          control2_D[7:0] = up_dinQ[7:0] ;
        end
     3: begin // 0x3
          control2_D[11:8] = up_dinQ[3:0] ;
        end
     4: begin // 0x4
        end
     5: begin // 0x5
          stickybit1S_D = up_dinQ[7] ;
          stickybit2S_D = (stickybit2S_D & ~up_dinQ[6])
                          | stickybit2 ;
        end
     6: begin // 0x6
          repeatingfield0_D = up_dinQ[7] ;
          morerepeat_D[0] = up_dinQ[6] ;
        end
     7: begin // 0x7
          repeatingfield1_D = up_dinQ[7] ;
          morerepeat_D[1] = up_dinQ[6] ;
        end
   endcase
end // always begin ... end
```



Address Decoder



From up_dinQ to control register input signals

(up_cslQQ & !up_cslQ & !up_rwlQ) → IE

if (IE & A0**)**

if (IE & A1**)** up_dinQ[7:0] → control1_D[7:0]

if (IE & A2**)** up_dinQ[7:0] → control2_D[7:0]

if (IE & A3**)** up_dinQ[3:0] → control2_D[11:8]

if (IE & A4**)**

if (IE & A5**)** up_dinQ[7] → stickybit1S_D

if (IE & A5**)** ((stickybit2S_D & ~up_dinQ[6]) | stickybit2) → stickybit2S_D

if (IE & A6**)** up_dinQ[7] → repeatingfield0_D

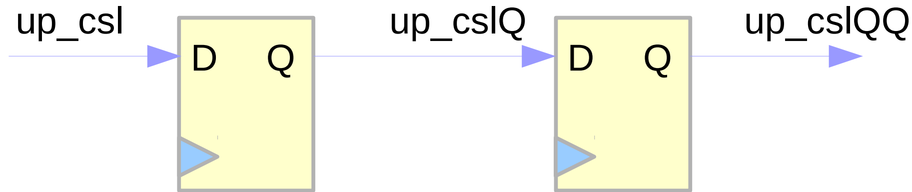
if (IE & A6**)** up_dinQ[6] → morerepeat_D[0]

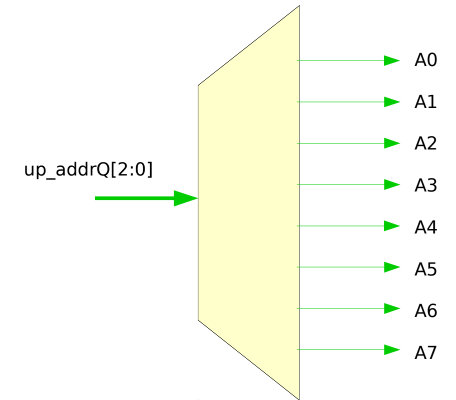
if (IE & A7**)** up_dinQ[7] → repeatingfield1_D

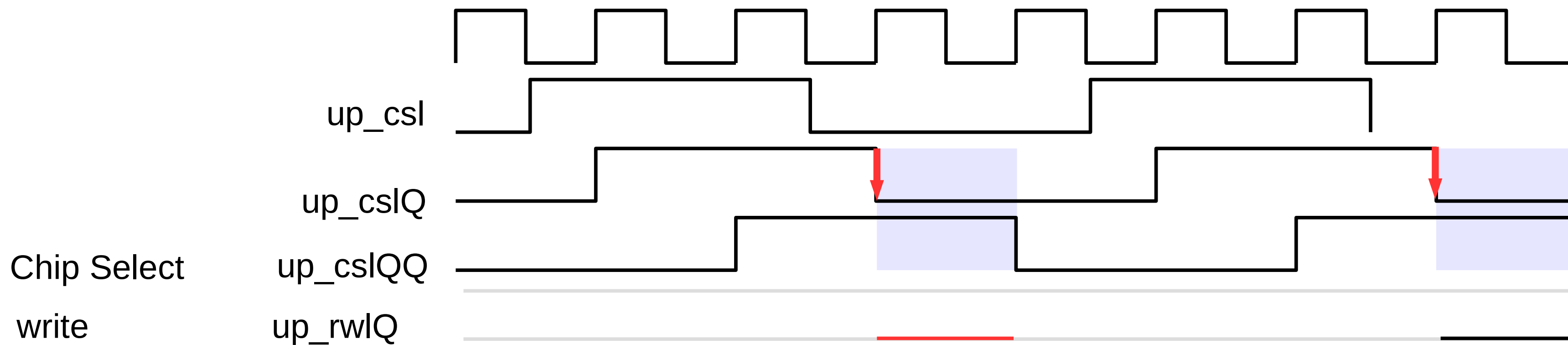
if (IE & A7**)** up_dinQ[6] → morerepeat_D[1]

Control Register Input Condition

(up_cslQQ & !up_cslQ & !up_rwlQ) → IE







Control Register Timing Diagram

up_csl

up_cslQ

Chip Select up_cslQQ

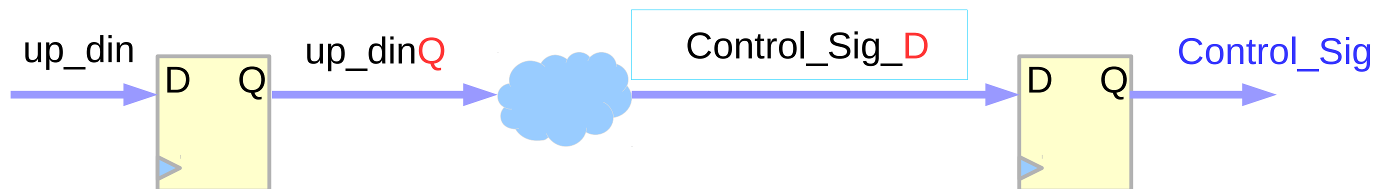
write up_rwlQ

up_addr

up_din

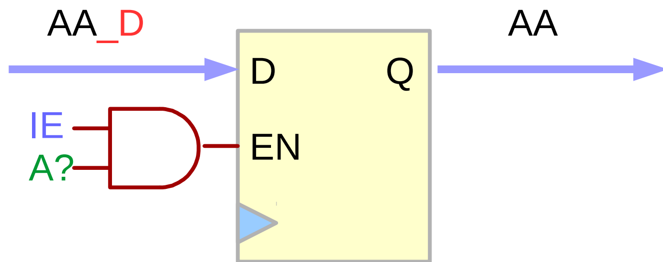
up_dinQ

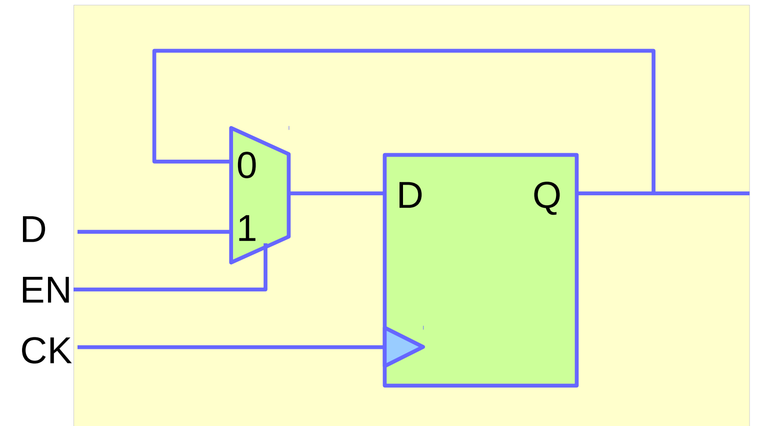
Control_Sig



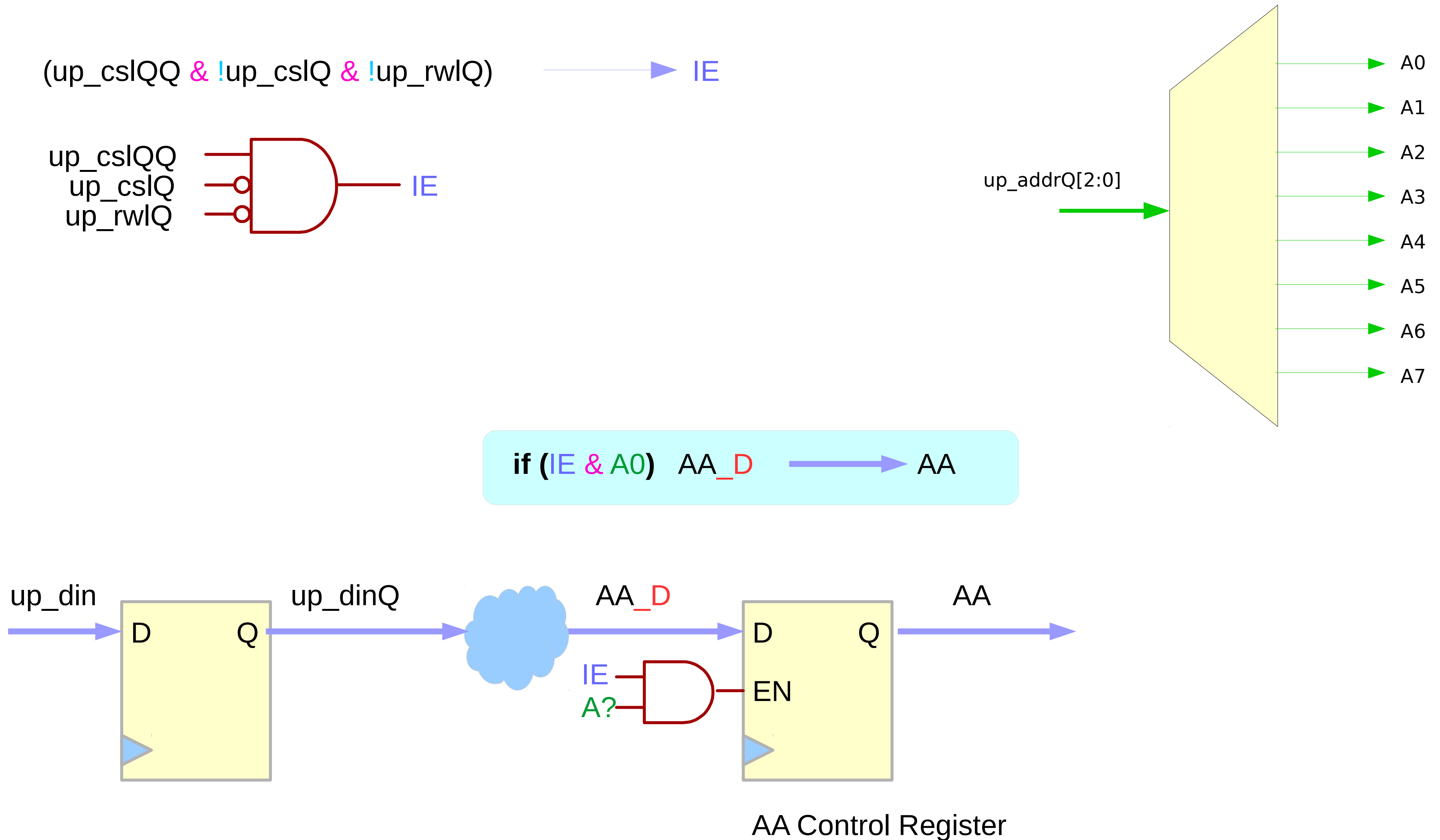
D FlipFlop with Enable

if (IE & A0**)** AA_D → AA

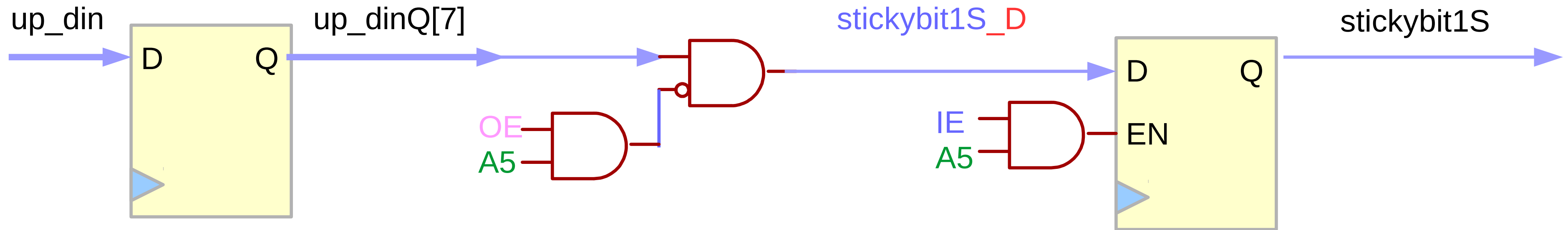


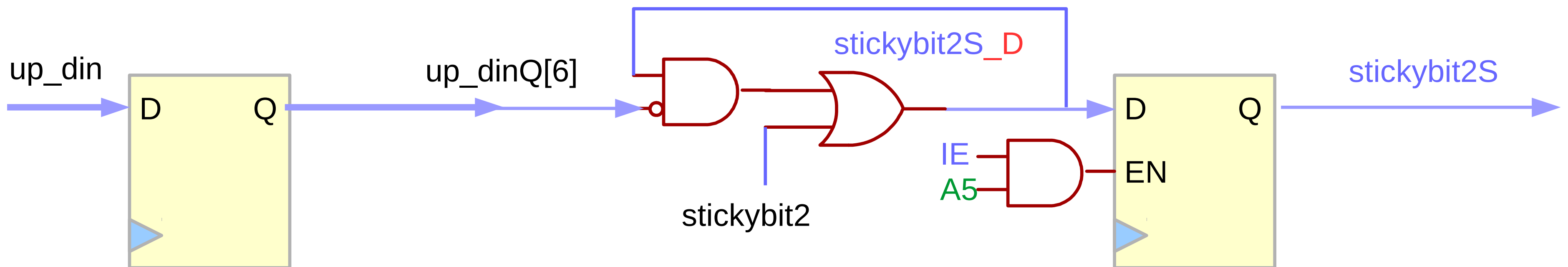


Control Register Input from DinQ

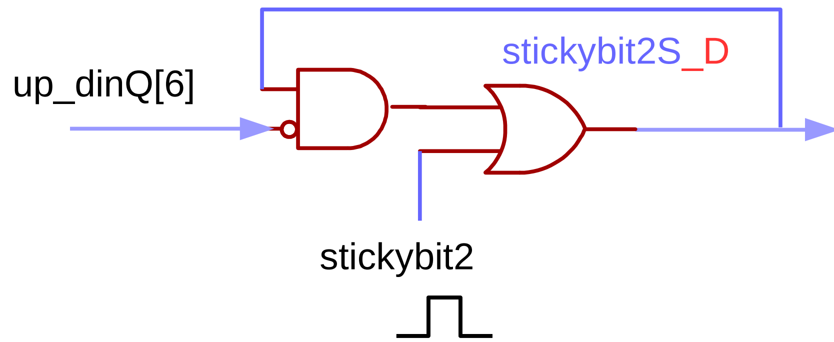


Stickybit Registers

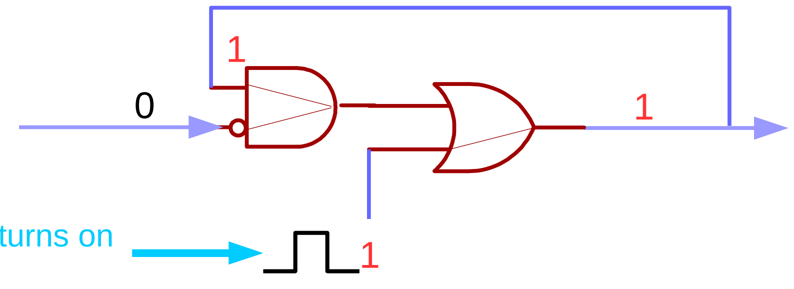




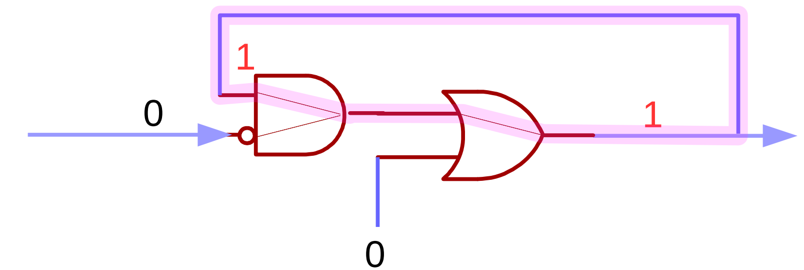
Stickybit Registers – Combinational Feedback Loop



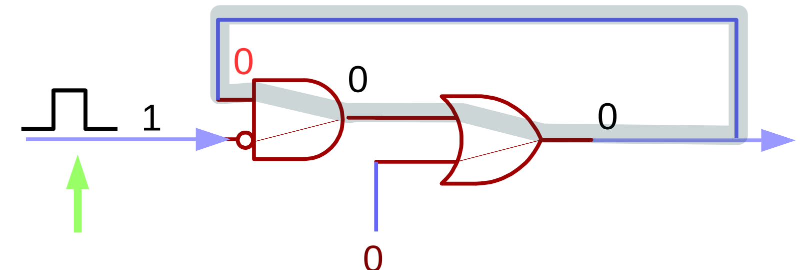
High stickybit2 from peri device turns on the feedback loop
This H can only be turned off by H up_din[6]



H pulse turns on the loop



Stable Loop
Stuck-At-1



set_false_path STA command

H pulse turns off the loop

Stickybit Registers – related code segments

```
always @ (up_din
        or stickybit1
        or stickybit2
        or stickybit1S
        or stickybit2S
       ) begin

 stickybit1S_D    = stickybit1S | stickybit1 ;
 stickybit2S_D    = stickybit2S | stickybit2 ;

 // Compute FF input signals for READ
 up_dout_D = 0 ;
 if (up_cslQQ & ! up_cslQ & up_rwlQ)
    case (up_addrQ)
      5: begin // 0x5
           up_dout_D[7]  = stickybit1S ;
           stickybit1S_D = 0 ;
           up_dout_D[6]  = stickybit2S ;
         end
    endcase

    // Compute FF input signals for WRITE
 if (up_cslQQ & ! up_cslQ & ! up_rwlQ)
    case (up_addrQ)
      5: begin // 0x5
           stickybit1S_D = up_dinQ[7] ;
           stickybit2S_D = (stickybit2S_D & ~up_dinQ[6])
                          | stickybit2 ;
         end
      endcase
 end // always begin ... end
```

```
module example_lp (
                       ,stickybit1
                       ,stickybit2
                    ) ;

input          stickybit1;
input          stickybit2;

reg            stickybit1S, stickybit1S_D;
reg            stickybit2S, stickybit2S_D;
```

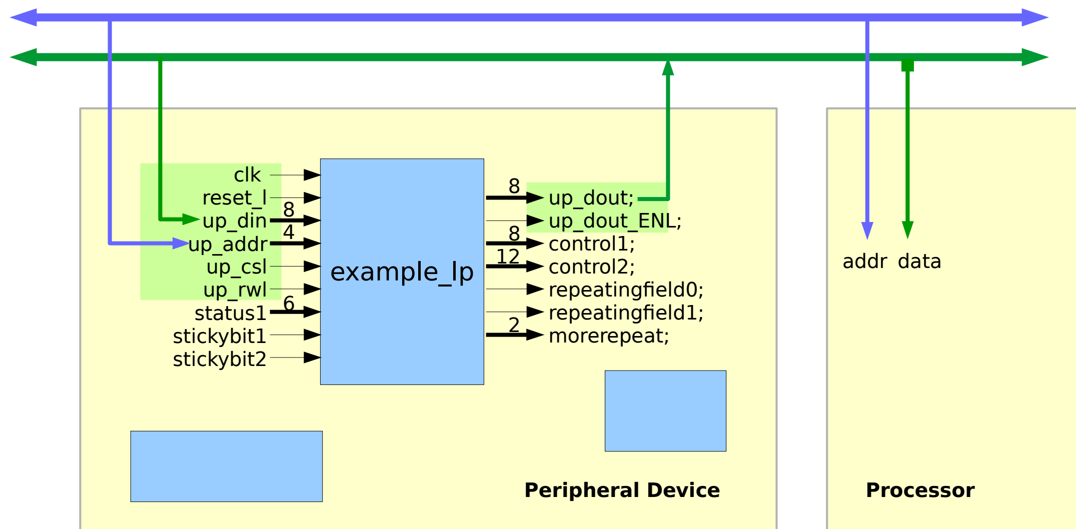
```
always @ (posedge clk or negedge reset_l)
  if ( ! reset_l) begin
     stickybit1S     <= 0 ;
     stickybit2S     <= 0 ;
  end
  else begin
     stickybit1S     <= stickybit1S_D ;
     stickybit2S     <= stickybit2S_D ;
  end
```

RepeatingField Registers – related code segments

```
always @ (or repeatingfield0 or repeatingfield1 or morerepeat ) begin
  repeatingfield0_D = repeatingfield0 ;
  repeatingfield1_D = repeatingfield1 ;
  morerepeat_D      = morerepeat  ;

  // Compute FF input signals for READ
  up_dout_D = 0 ;
  if (up_cslQQ & ! up_cslQ & up_rwlQ)
     case (up_addrQ)
       6: begin // 0x6
            up_dout_D[7] = repeatingfield0 ;
            up_dout_D[6] = morerepeat[0] ;
          end
       7: begin // 0x7
            up_dout_D[7] = repeatingfield1 ;
            up_dout_D[6] = morerepeat[1] ;
          end
     endcase

  // Compute FF input signals for WRITE
  if (up_cslQQ & ! up_cslQ & ! up_rwlQ)
     case (up_addrQ)
       6: begin // 0x6
            repeatingfield0_D = up_dinQ[7] ;
            morerepeat_D[0] = up_dinQ[6] ;
          end
       7: begin // 0x7
            repeatingfield1_D = up_dinQ[7] ;
            morerepeat_D[1] = up_dinQ[6] ;
          end
       endcase
  end // always begin ... end
```

```
module example_lp (
                 ,repeatingfield0
                 ,repeatingfield1
                 ,morerepeat
             ) ;

output           repeatingfield0;
output           repeatingfield1;
output   [1:0]   morerepeat;

reg              repeatingfield0;
reg              repeatingfield1;
reg      [1:0]   morerepeat;

reg              repeatingfield0_D;
reg              repeatingfield1_D;
reg      [1:0]   morerepeat_D;

  always @ (posedge clk or negedge reset_l)
    if ( ! reset_l) begin
      repeatingfield0 <= 0 ;
      repeatingfield1 <= 0 ;
      morerepeat      <= 0 ;
    end
    else begin
      repeatingfield0 <= repeatingfield0_D ;
      repeatingfield1 <= repeatingfield1_D ;
      morerepeat      <= morerepeat_D ;
    end
```

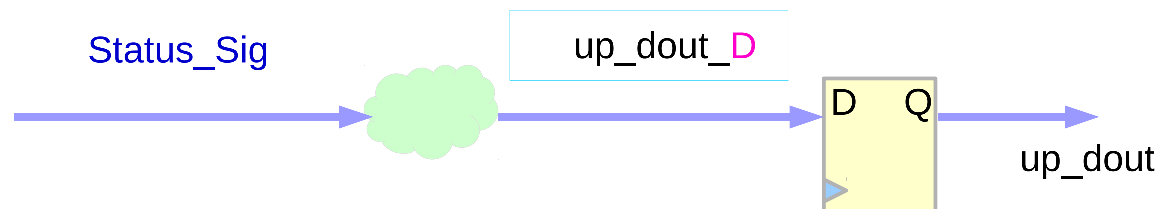
Write Registers



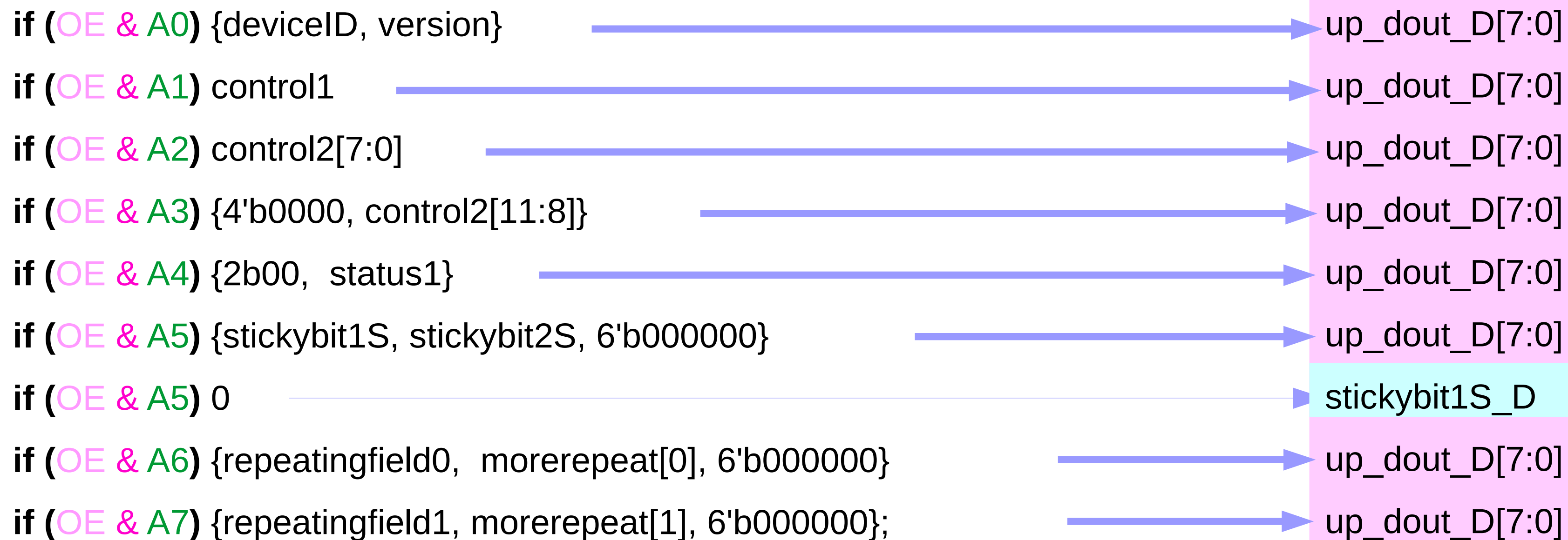
Output data register input signals

up_dout_D <= Status_Sig

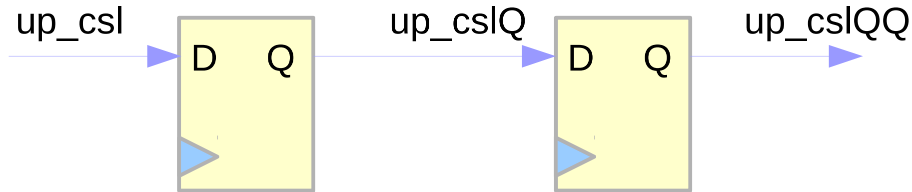
```
// Compute FF input signals for READ
up_dout_D = 0 ;
if (up_cslQQ & ! up_cslQ & up_rwlQ)
   case (up_addrQ)
     0: begin // 0x0
          up_dout_D[3:0] = version ;
          up_dout_D[7:4] = deviceID ;
        end
     1: begin // 0x1
          up_dout_D[7:0] = control1 ;
        end
     2: begin // 0x2
          up_dout_D[7:0] = control2[7:0] ;
        end
     3: begin // 0x3
          up_dout_D[3:0] = control2[11:8] ;
        end
     4: begin // 0x4
          up_dout_D[5:0] = status1 ;
        end
     5: begin // 0x5
          up_dout_D[7]  = stickybit1S ;
          stickybit1S_D = 0 ;
          up_dout_D[6]  = stickybit2S ;
        end
     6: begin // 0x6
          up_dout_D[7] = repeatingfield0 ;
          up_dout_D[6] = morerepeat[0] ;
        end
     7: begin // 0x7
          up_dout_D[7] = repeatingfield1 ;
          up_dout_D[6] = morerepeat[1] ;
        end
   endcase
```

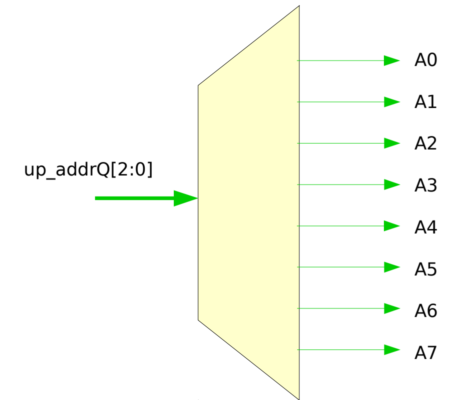


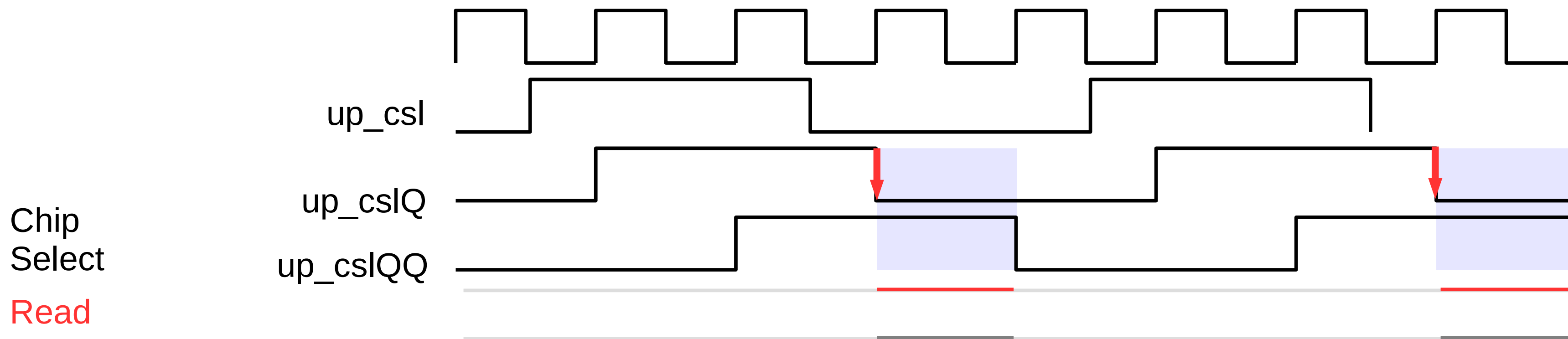
Input data to up_dout register



Status Register Output Condition

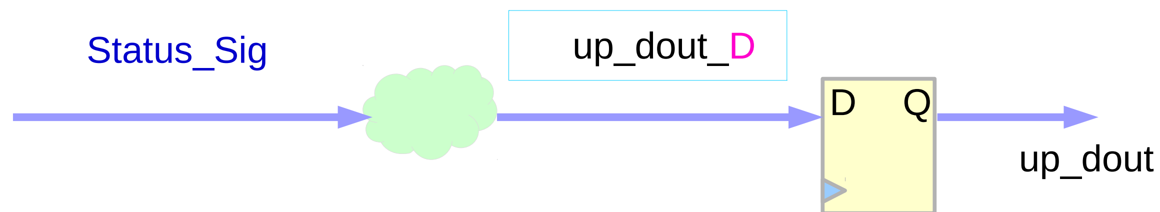






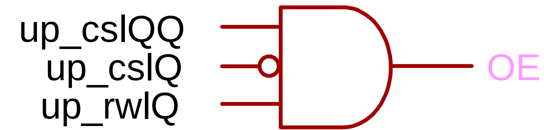
Output Data Register Timing Diagram

up_csl
up_cslQ
Chip Select up_cslQQ
write up_rwlQ
up_addr
up_din
Status_Sig
up_dout
up_dout_ENL



From Status Signals to Dout

(up_cslQQ & !up_cslQ & !up_rwlQ) → OE



up_addrQ[2:0]

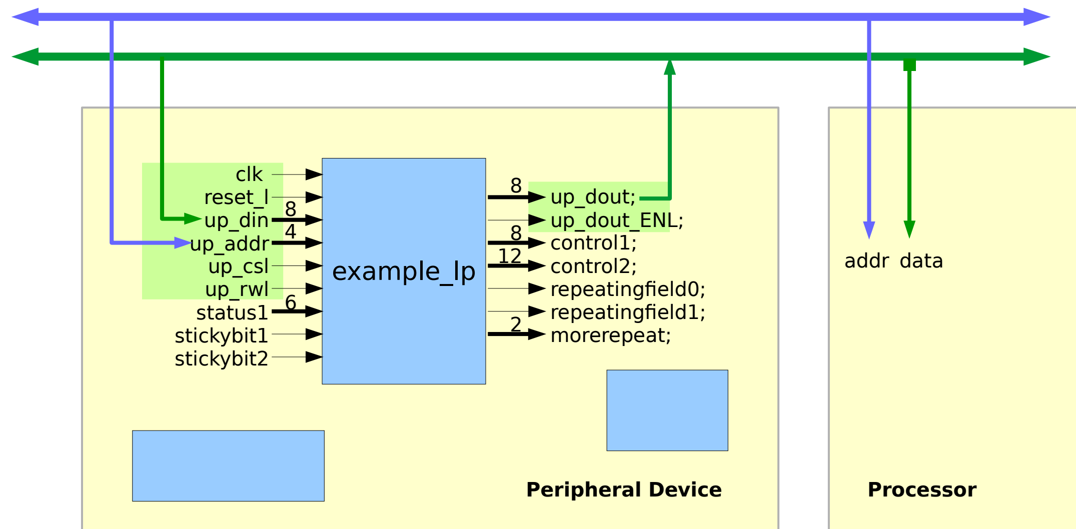
{deviceID, version}
control1
control2[7:0]
{4'b0, control2[11:8]}
{2b0, status1}
{stickybit1S, stickybit2S, 6'b0}
0
{rptfd0, mrpt[0], 6'b0}
{rptfd1, mrpt[1], 6'b0};

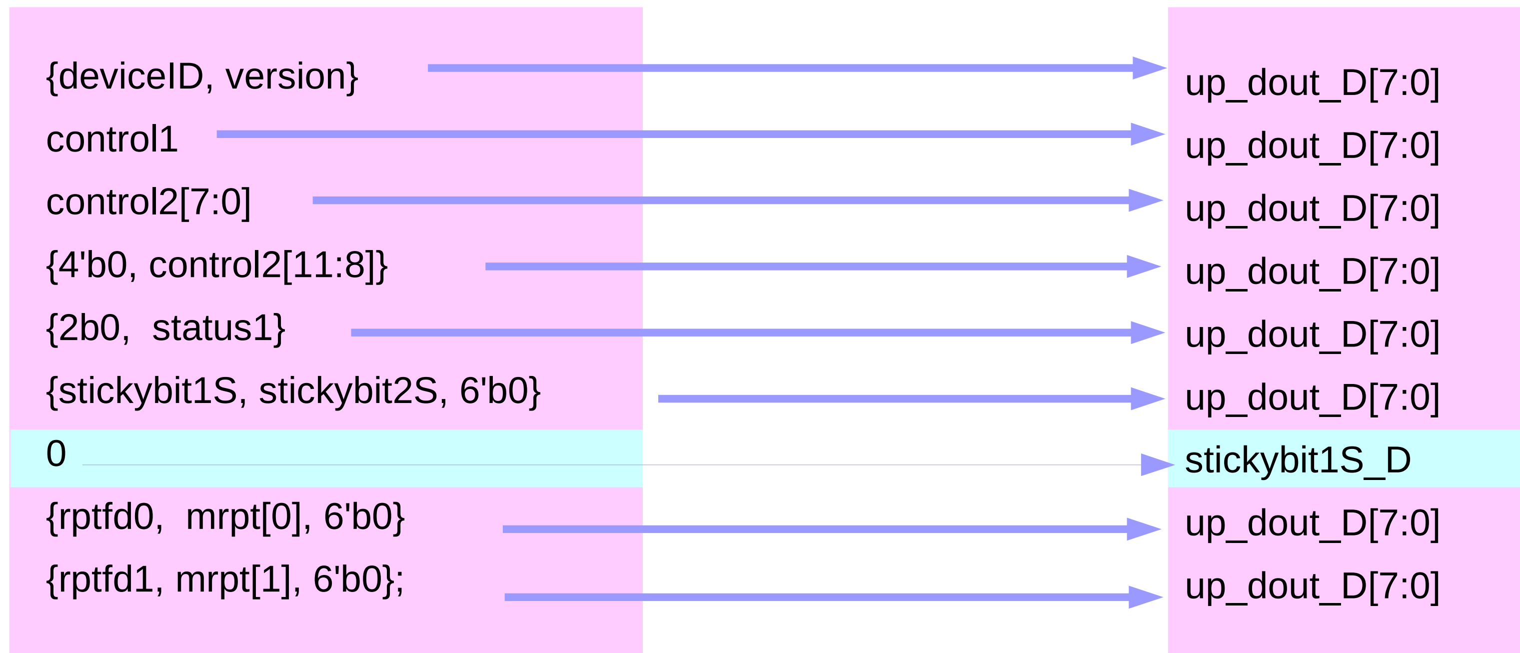
up_dout_D[7:0]

D Q
OE EN

up_dout

Read Status Registers



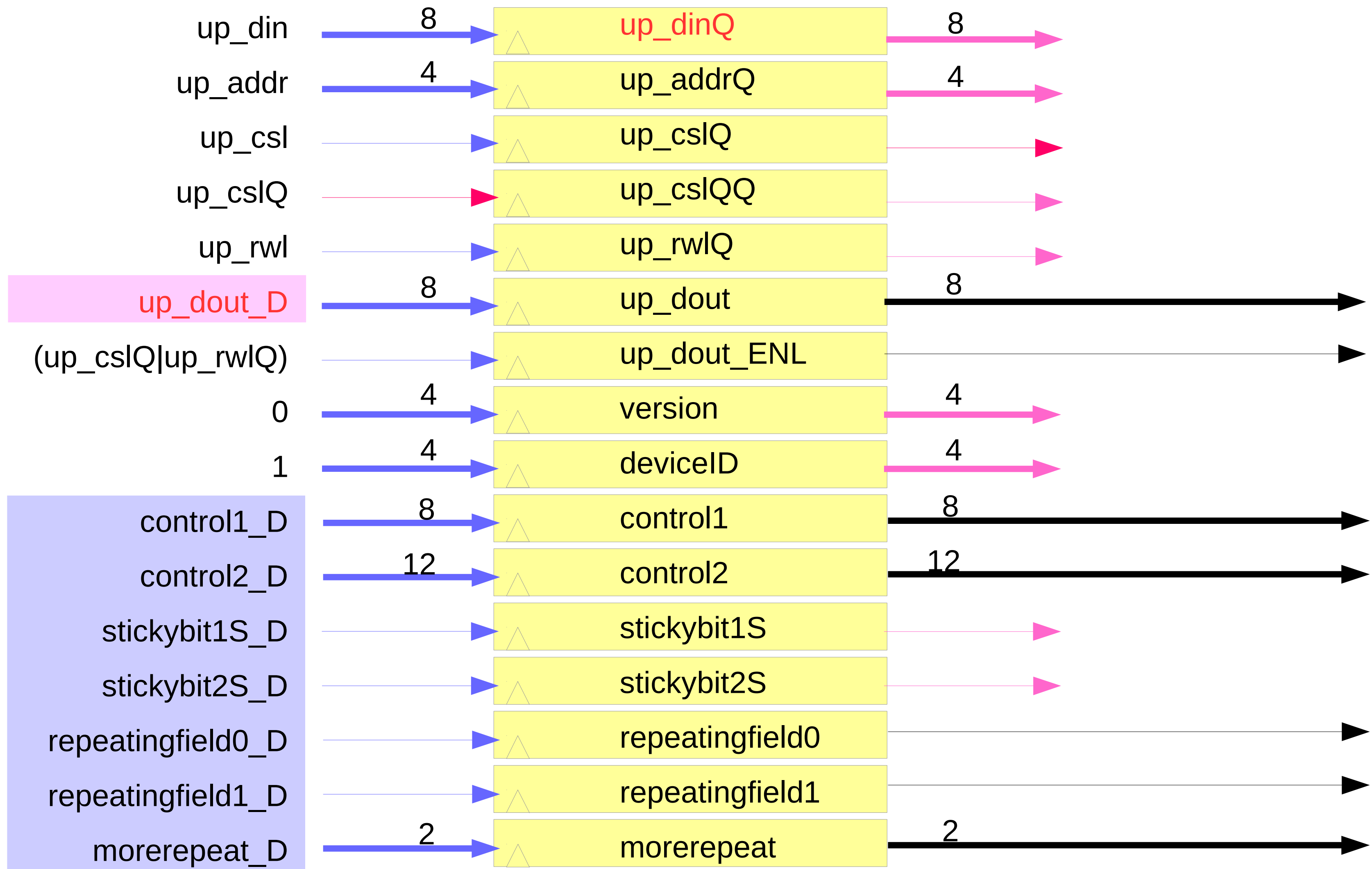


FF Inference

```
always @ (posedge clk or negedge reset_l)
  if ( ! reset_l) begin
    up_dinQ           <= 0 ;
    up_addrQ          <= 0 ;
    up_cslQ           <= 0 ;
    up_cslQQ          <= 0 ;
    up_rwlQ           <= 0 ;
    up_dout           <= 0 ;
    up_dout_ENL       <= 1 ;
    version           <= 15 ;
    deviceID          <= 14 ;
    control1          <= 0 ;
    control2          <= 0 ;
    stickybit1S       <= 0 ;
    stickybit2S       <= 0 ;
    repeatingfield0 <= 0 ;
    repeatingfield1 <= 0 ;
    morerepeat        <= 0 ;
  end
```

```
  else begin
    up_dinQ           <= up_din ;
    up_addrQ          <= up_addr ;
    up_cslQ           <= up_csl ;
    up_cslQQ          <= up_cslQ ;
    up_rwlQ           <= up_rwl ;
    up_dout           <= up_dout_D ;
    up_dout_ENL       <= (up_cslQ|up_rwlQ) ;
    version           <= 0 ;
    deviceID          <= 1 ;
    control1          <= control1_D ;
    control2          <= control2_D ;
    stickybit1S       <= stickybit1S_D ;
    stickybit2S       <= stickybit2S_D ;
    repeatingfield0 <= repeatingfield0_D ;
    repeatingfield1 <= repeatingfield1_D ;
    morerepeat        <= morerepeat_D ;
  end

endmodule
```

Registers



Module Skeleton (1)

```
module chip_up_ifc (/*AUTOARG*/) ;
input clock, initl ;
input   [7:0]     version;
input             someerror;
input             read;
input             write;
input   [3:0]     address;
input   [7:0]     up_datain;
output  [7:0]     field1;
output  [3:0]     field2;
output  [7:0]     up_dataout;
reg     [7:0]     field1, field1_D;
reg     [3:0]     field2, field2_D;
reg               someerrorS, someerrorS_D;
reg     [7:0]     up_dataout, up_dataout_D ;

always @ (/*AUTOSENSE*/) begin
  field1_D = field1 ;
  field2_D = field2 ;
  someerrorS_D = someerrorS | someerror ;
  up_dataout_D = up_dataout ;

if (write) case (address)
0: begin
  field1_D = up_datain[7:0] ;
end
1: begin
end
2: begin
  field2_D = up_datain[3:0] ;
  someerrorS_D = (someerrorS_D & ~up_datain[6]) | someerror ;
end
endcase
```

Module Skeleton (2)

```
if (read) case (address)
0: begin
  up_dataout_D[7:0] = field1 ;
end
1: begin
  up_dataout_D[7:0] = version ;
end
2: begin
  up_dataout_D[3:0] = field2 ;
  up_dataout_D[6] = someerrorS ;
end
endcase
end

always @ (posedge clock or negedge initl)
  if ( ! initl) begin
    field1 <= 0 ;
    field2 <= 0 ;
    someerror <= 0 ;
    up_dataout <= 0 ;
  end
  else begin
    field1 <= field1_D ;
    field2 <= field2_D ;
    someerrorS <= someerrorS_D ;
    up_dataout <= up_dataout_D ;
  end
endmodule
```

CSR Documentation

Address: 0x0000 - a simple version register			
3:0	RO	0	da version
7:4	RO	0	deviceID

Address: 0x0001			
7:0	R/W	0	control1

Address: 0x0002			
7:0	R/W	0	control2

Address: 0x0003			
3:0	R/W	0	control2

Address: 0x0004			
5:0	RO	0	status1

Address: 0x0005			
7	COR	0	stickybit1
6	RW1C	0	stickybit2

Addresses: 0x0006 through 0x0007			
7	R/W	0	repeatingfield***N***
6	R/W	0	morerepeat***[N]***

Properties of CSRs (1)

This section lists the properties that may be set on individual register fields. Most can be applied in combination with others. These keywords can be in any order on the same line with the register field definition. Upper vs. lower case is not significant.

- Intern: internal – indicates that the field is not an output or input of this verilog module.
- RO: read only – the field is only present in the read logic, and is an input to the module (unless 'Intern' property is also declared).
- COR: clear on read – the field is reset to 0 when the address is read.
- W1C: write-1-to-clear – the field is reset to 0 when the address is written and the corresponding bit(s) is/are 1, but is not altered if the bit(s) is/are 0. (csrGen implements this on a per-bit basis for multi-bit fields).
- ST: sticky – the field is sticky. The name in the definition is taken as an input to the module (unless 'Intern'), and a flop is created for the sticky memory. The sticky value will remain 1 if the nominal value is ever 1, if even only for one clock cycle. In the example above, someerror becomes an input, someerrorS is the sticky flop, and logic is also present for the 'W1C' property. This is customarily a single bit field.
- SOR: set on read – the field is set to all 1's when the address is read.
- DOR: decrement on read – the field is decremented by 1 when the address is read.

csrGen User's Manual: Automated CSRs for ASICs/FPGAs 9

Properties of CSRs (2)

- DORS: decrement on read, saturating – if not zero, the field is decremented by 1 when the address is read.
- IOR: increment on read – the field is incremented by 1 when the address is read.
- IORS: increment on read, saturating – if not all 1's, the field is incremented by 1 when the address is read.
- W1S: write-1-to-set – the field is set to 1's when the address is written and the corresponding bit(s) is/are 1, but is not altered if the bit(s) is/are 0. (csrGen implements this on a per-bit basis for multi-bit fields).
- WO: write only – the field is only present in the write logic.
- ST0: sticky low – like sticky, but the sticky value stays 0 if the nominal value is ever 0.
- Incr: incrementer – the field is an incrementing counter. The name in the definition is taken as a one bit input to the module (unless 'Intern'), and a counter with the field width is defined with '_cntr' appended to the name.
- IncrS: incrementer, saturating – same as Incr, but stops incrementing at all 1's.
- Decr: decrementer – similar to Incr, but decrementing.
- DecrS: decrementer, saturating – same as Decr, but stops decrementing at all 0's.

Properties of CSRs (3)

- SUB/SUBM: subset and subset msb – if several fields/addresses are to be catenated to form a larger field, the same name is used for each subset and is defined as a field with the SUB property followed by a range definition. The subset field with the most significant bit of the larger field must use the SUBM property. For example:

```
%A 0
31:0 bigfield SUB 31:0
%A 1
15:0 bigfield SUBM 47:32
```

- Shadow: any field that is not to be constructed by csrGen, but is implemented by user added logic can still be defined in the template for the purpose of documentation and generating firmware or verification definition files. The shadow property prevents csrGen from generating logic for the register.
- Pulse: the field asserts for one cycle when written as 1, and always reads as 0.
- PulseA: assert until acknowledged – when written as 1, the field asserts until an acknowledgement is received. The name used for the acknowledgment is the field name with _ack appended. The acknowledgment results in the deassertion at the next clock edge.
- any numeric value is taken as a reset value for the flops in the register.
- buss: for `%AREPEAT` only, single bit fields only, specifies that the name is taken as a vector with a width matching the repeat count, and each address corresponds to one bit of the vector.

References

[1] http://wiki.osdev.org/ARM_RaspberryPi_Tutorial_C
[2] http://blog.bobuhiro11.net/2014/01-13-baremetal.html
[3] http://www.valvers.com/open-software/raspberry-pi/
[4] https://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/downloads.html