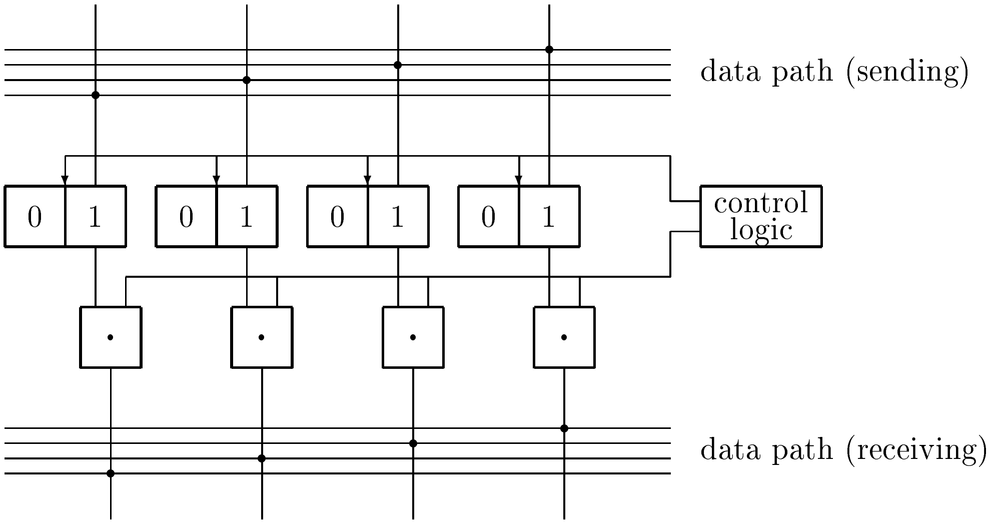
9. Registers and functional elements

In this chapter we will take a brief look at a few different circuits that are available. These are registers and bus drivers, comparators, de- and encoders, as well as (de)multiplexers. Such circuits are quite common in microprocessor systems and available in MSI (medium scale integration) and LSI (large scale integration) chips. We cannot deal with all different forms, because this is just an introduction which should allow to understand the terminology and functionality. The interested reader should take a look at some data books, for example the *Texas Instruments* Data Book.

9.1 Registers and busses

As remarked in Chapter 7, the D flip-flop is one of the simplest FF types, and mainly used to store some bits temporarily. As an example, we can have a few parallel input lines and the same number of output lines. These may be the data or the address busses of a microprocessor, of a complex CPU (central processing unit) printed circuit board, or all the connections of all boards in a big *mainframe* computer.

One of the simplest forms to store data temporarily is taking a few D flip-flops in parallel, perhaps with a few ANDs if we want to control not only the input but also the output:

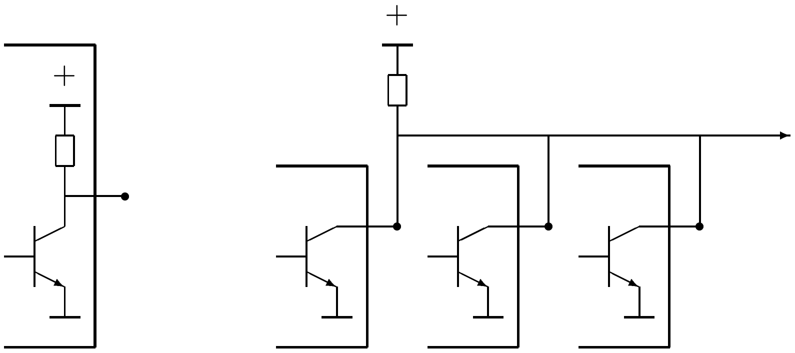


The sending data path is also called **input bus** or simply inbus, the receiving data path is then called **output bus** or outbus. The signals from the control logic are normally called **input strobe** and **output strobe**. The **register** shown above has 4 bits, but this could be 8, 16, 32, or in the case of modern RISC[1] machines 64 bits.

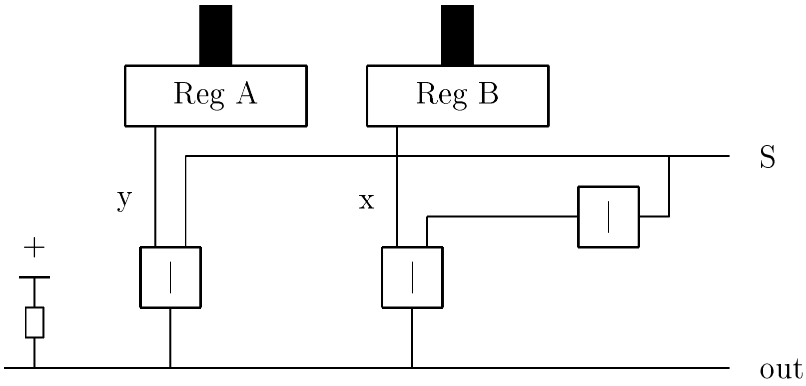
If we want to copy e.g. one byte from one input bus to two output busses, we can take two registers in parallel and control the data flow by means of two signals: an input strobe for each register. However, if we want to copy data from two input busses to one central output bus, we need other solutions. One such solution is based on *open collector outputs.* Any gate has a transistor at its output and this is normally already connected to a resistor and e.g. +5V inside the chip (below left). If the collector is not connected to a resistor

[1]RISC = reduced instruction set computer

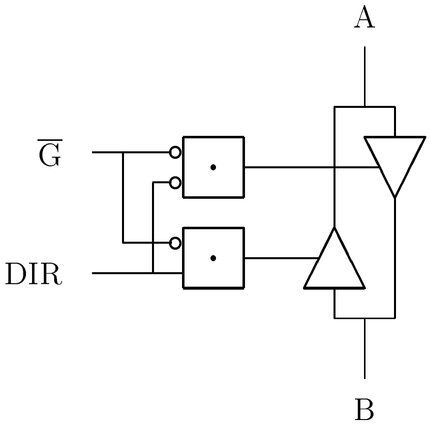
inside the chip, i.e. the collector is "open," we need to take a resistor outside the chip, in which case we may connect multiple open collector outputs to a single line (below right).

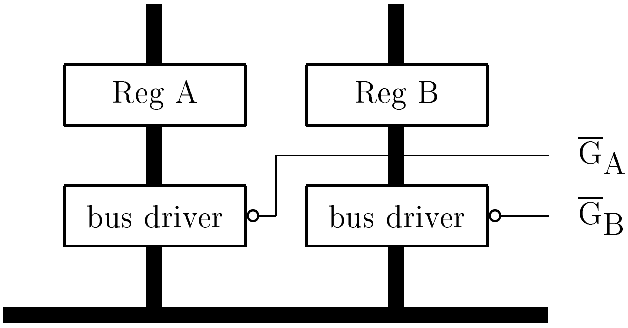


The external resistor is called *pull-up* resistor because it "pulls" the outputs to e.g. +5V. If one or more transistors is conducting, the output line will go to 0. For this reason we speak of a *wire-AND* connection, for example to select one register to a common output bus (we show only one bit of the registers and the output bus):



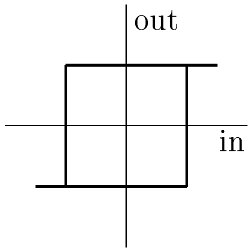
In this example there are two input busses and two registers of N bits. If the control signal S (from select) is 0, the output of the left NAND will be 1, but the output of the right NAND and therefore the output line, will be $\overline{1 \cdot x} = \bar{x}$. In the case that S=1 we will have $\bar{y}$. In this example we used NAND's, but we could have used AND's as well. Bus buffering is so common in computer systems that special circuits are available. Even bi-directional *bus transceivers* exist which allow to send data from bus A to B or from B to A (figure to the right). Such a transceiver has two control inputs: DIR for defining the direction and the active-low $\overline{\mathrm{G}}$ which is an *enable* input. If $\overline{\mathrm{G}} = 1$ the outputs have such a high impedance that they virtually do not exist (3-state outputs; 0, 1, or high impedance). Such 3-state transceivers allow to connect different registers or busses together, see the next figure.





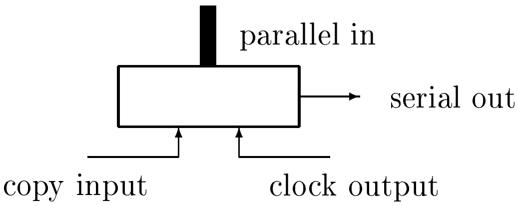
In this case there are no bi-directional bus transceivers but one-directional bus drivers. Care must be taken that the two enable signals $\overline{G}_A$ and $\overline{G}_B$ are not both 0 at the same time.

If the connections are long, which means that noise from neighbouring connections can be induced, transceivers can be selected with hysteresis. This hysteresis means that the input signal must exceed a certain voltage difference before the output will change, see figure to the right. Note that this figure is an idealisation, the hysteresis function does not need to be an ideal square.

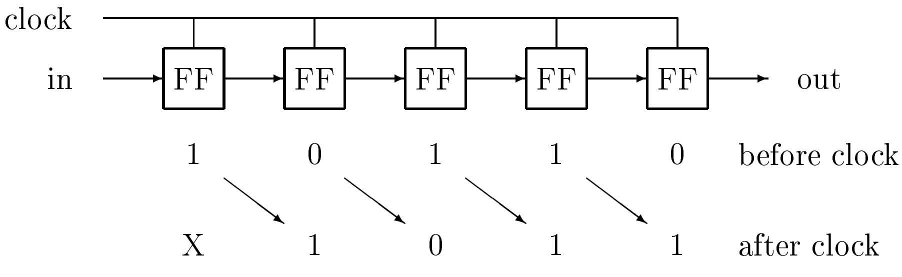


Shift registers

Sometimes data is passed in a serial way instead of in parallel. Examples are a terminal link type “RS232” and the MIDI interface for connecting synthesizers (or synthesizers with computers). In such cases the data need to be converted from parallel to serial, which can be done by means of **shift registers**.

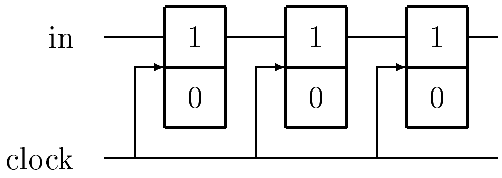


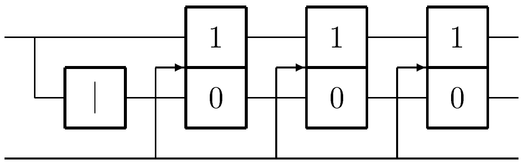
A shift register contains a number of flip-flops which can pass their information to their left or right neighbour. Passing = shifting. There exist different types: serial in, serial out; parallel in, serial out; and serial in, parallel out. An example, serial in serial out is shown below:



• The *rightmost* bit is lost after shifting to the right, unless it is stored somewhere else or transmitted.
• The leftmost bit can be filled in three ways: (a) always a 0, (b) always a 1, or (c) with other information. An example of the last option is to make a feedback loop from output to input, in which case we will get a *cyclic* shift.

Shift registers are also used for hardware multiplication and division, and the shift function is therefore often incorporated into an “accumulator” or ALU (arithmetic logical unit) of a CPU (central processing unit). Shift registers can be realised by means of cascades of D (left) or JK/SR (right) flip-flops:





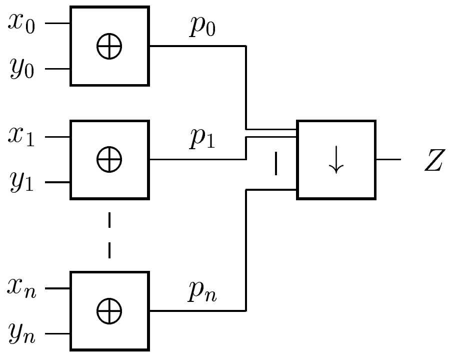
Apart from simple gates like ANDs, ORs, counters, etc, there are many MSI (medium scale integration) circuits that perform specific tasks. These are often used in computer circuits: comparators, encoders, decoders, multiplexers, demultiplexers.

9.2 Comparator

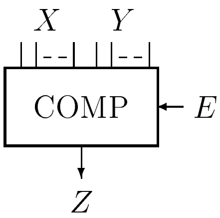
Purpose: compare two binary numbers X and Y.

- For many applications only one output is necessary, i.e. $Z = 1$ if $X = Y$ and $Z = 0$ if $X \neq Y$.

  Example



$$p_0 = x_0 \oplus y_0 = \begin{cases} 1 & \text{if } x_0 \neq y_0 \\ 0 & \text{if } x_0 = y_0 \end{cases}$$

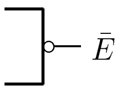


  From the above scheme, we can observe that

$$Z = \overline{p_0 + p_1 + \cdots + p_n} = \overline{p_0} \cdot \overline{p_1} \cdots \overline{p_n}$$

  Often there is an enable (E) input:

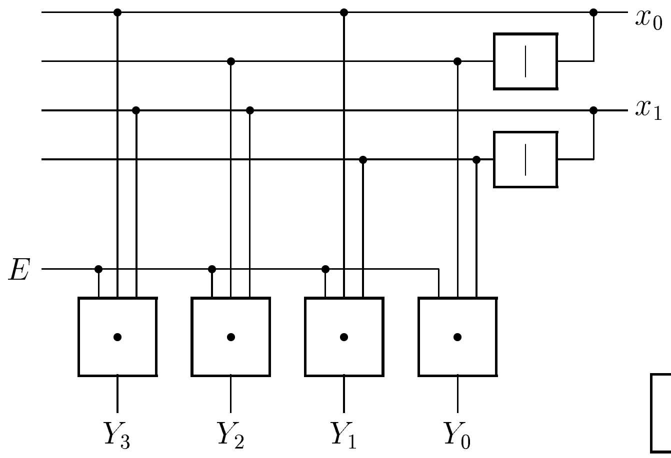
$$Z = \text{invalid if } E = 0, \text{ valid if } E = 1$$

  Another option is to use $\bar{E}$ (valid if $\bar{E} = 0$), indicated by 

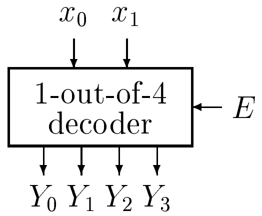
- If additional information is required, i.e. $X < Y$ and/or $X > Y$, relatively complex circuits with many gates are required, especially if the number of bits is large. It is then simpler to subtract X and Y and to examine the sign bit.

9.3 Decoder

We have seen examples of decoders on page 31, i.e. BCD and 7-segment. A normal decoder (*binary*) is not based on exploiting don't cares in the function table.
Purpose: a binary 1-out-of-2^n decoder converts a code to one bit line, i.e. exactly 1 output bit may be one.



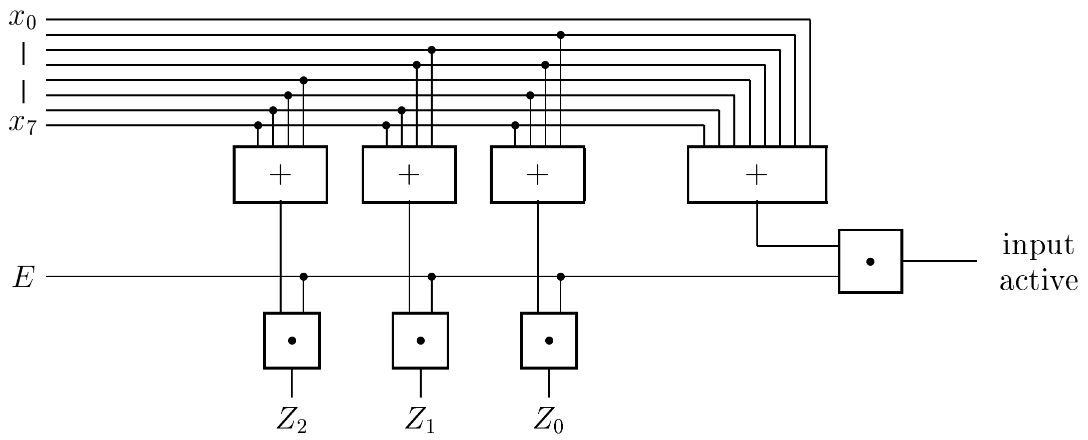
in		out			
x_1	x_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

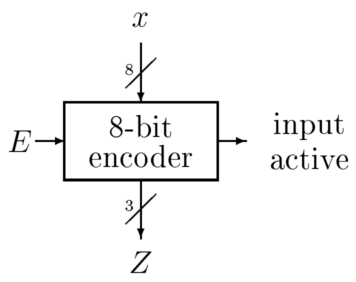


9.4 Encoder

An encoder does exactly the reverse, namely it generates the binary code of an active input line. In general, 2^k inputs give k outputs. An extra output line may indicate whether there is one input line active (to detect the case that *no* input line is active). It may also be necessary to detect the case that multiple lines are active and an invalid code is generated.
A solution to the latter is a so-called *priority encoder*: if more than one input line is active, the output code corresponds to the highest-numbered active input line.

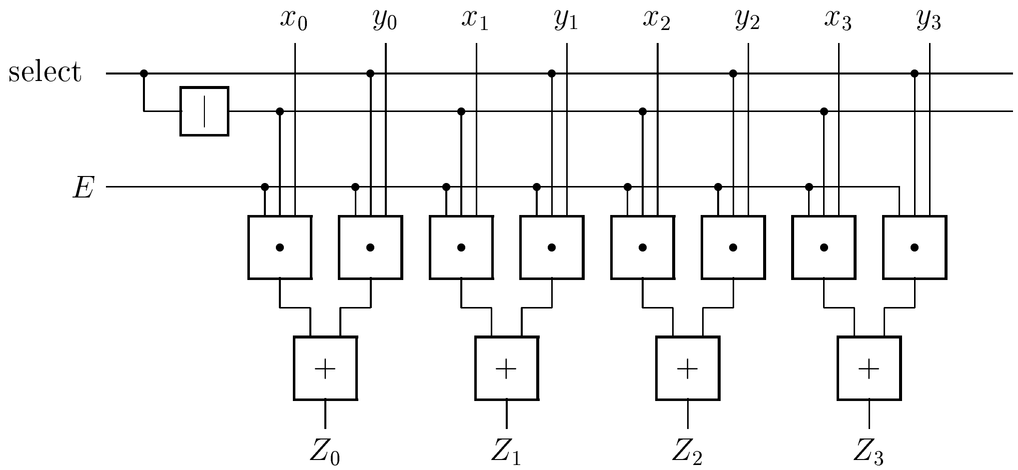
x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0	Z_2	Z_1	Z_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

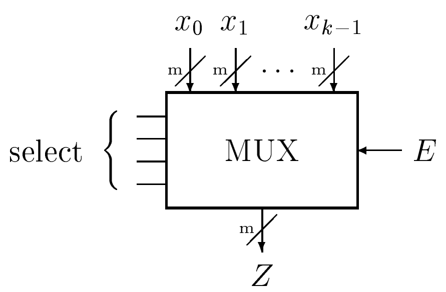




9.5 Multiplexer

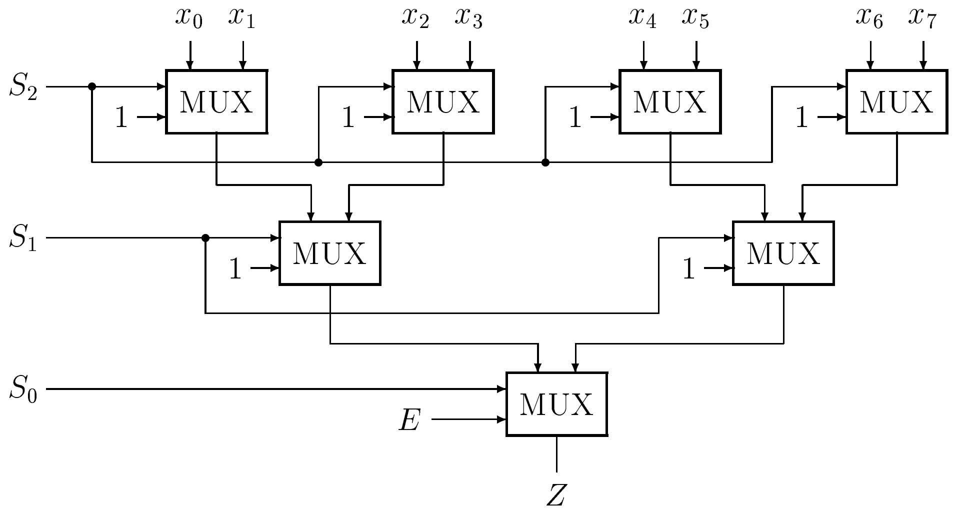
Purpose: to route data from one of several sources to a common destination.
The source is selected by applying appropriate selection signals. If the maximum number of data sources is k and each source has m bits, the multiplexer is called a k-input m-bit multiplexer. It is convinient to choose $k = 2^p$. It is convinient to make $k = 2^p$ such that the source is encoded by p bits.
A multiplexer can be used to generate any combinational function.





k-input m-bit multiplexer

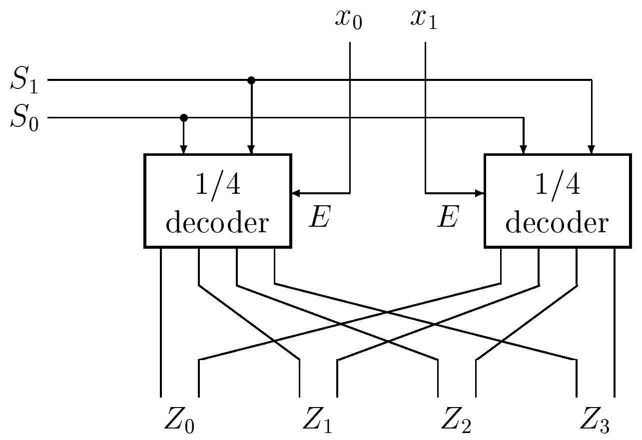
In the above symbol the input x_0 corresponds to the two signals x_0 and y_0, as shown in the circuit, for the case that $m = 2$. Multiplexers can be applied in a tree-like structure, e.g. we can construct an 8-input 1-bit multiplexer from seven 2-input 1-bit multiplexers:



In the above figure, the S_i are the *select* signals. In this circuit only the last multiplexers have been connected to the enable input; all others are always enabled (1, i.e. connected with a resistor to +5V). This has the advantage that the circuit which generates the enable signal needs to drive only one input.

9.6 Demultiplexer

Purpose: to route data from a common source to one of 2^n destinations. This is exactly the reverse function of a multiplexer.
This can be done by employing *decoders*: the enable (control) input of each decoder is a 1-bit source, the destination is the selection address applied to the decoder:



4-output
2-bit
demultiplexer

A k-output m-bit multiplexer can be constructed from m $1/k$ decoders. In general, $k = 2^p$.