complement representation. Hence, we obtain `1111 0010` as the two’s complement representation of decimal number −14.

6.3 Fixed- and Floating-Point Representations

Binary number to be processed in a digital system may have a fractional part. We distinguished the integer and fractional parts of such numbers by a dot in the previous section. This is not possible in a digital system. Instead, there are two methods to represent a binary number with integer and fractional parts. These are fixed-and floating-point representations.

6.3.1 Fixed-Point Representation

The number of bits assigned to the integer and fractional parts is fixed in this representation. Hence the name fixed-point representation. This method is easy to implement since the number of bits assigned to the integer and fractional parts is fixed.

We can show an unsigned fixed-point number (without a sign bit) as UQp.q. Here, U indicates the unsigned bit notation; pq represents the number, p being the integer and q being the fractional part. We provide some fixed-point representation formats in Table 6.3. Note that we are not limited by these formats in an FPGA implementation since the user is free to assign any number of bits to the integer and fractional parts. We will see such examples in Secs. 6.7 and 6.9.

Format	Minimum	Maximum	Resolution	# bits for p	# bits for q	# total bits
UQ16.	0	$2^{16}-1$	1	16	0	16
UQ.16	0	$1-2^{-16}$	2^{-16}	0	16	16
UQ16.16	0	$2^{16}-1$	2^{-16}	16	16	32

TABLE 6.3 Fixed-Point Unsigned Number Representation Formats

Let’s reconsider the decimal number 14.125. We know that binary representation of this number is `1110.001`. Assume that we would like to represent this number in UQ16. form. Therefore, there will be no fractional part. The number of bits to be assigned to the integer part will be 16. Hence, the resulting number in hexadecimal form will be `000E`. Zeros appended to the left of the number will not affect its value. They will only satisfy the fixed-point representation format. If the UQ16.16 fixed-point representation is used for the same number, then the integer part of 14.125 will be the same in hexadecimal form as `000E`. The fractional part will be in hexadecimal form as `0200`. Here, zeros are appended to the right of the number. Therefore, the value of the fractional part will not be affected. As a result, fixed-point representation of the number will be `000E0200`. As can be seen here, there is no separator between the integer and fractional parts of the number. Knowing that the number is in UQ16.16 form, we can easily extract the integer and fractional parts (since we know the number of bits assigned to each).

In a similar way, we can represent signed numbers. In this form, the MSB is reserved for the sign bit. Therefore, we use the sign bit representation here. We provide three signed bit formats for the fixed-point representation in Table 6.4. Similar to the unsigned bit representation, fixed-point number will be in the form Qp.q.

Format	Minimum	Maximum	Resolution	# bits for p	# bits for q	# total bits
Q15.	-2^{15}	$2^{15}-1$	1	15	0	16
Q.15	-1	$1-2^{-15}$	2^{-15}	0	15	16
Q15.16	-2^{15}	$2^{15}-2^{-16}$	2^{-16}	15	16	32

TABLE 6.4 Fixed-Point Signed Number Representation Formats

Let's consider the decimal number −14.125. Assume that we would like to represent this number in Q15. form. The resulting number in hexadecimal form will be `800E`. Here, the MSB is set to `1` as the sign bit to represent that the number is negative. If the Q15.16 fixed-point representation is used for the same number, then hexadecimal form of the number will be `800E0200`. Again, the MSB is kept as the sign bit in this representation.

6.3.2 Floating-Point Representation

Fixed-point representation is easy to implement and process. However, it has a major drawback. The number of bits assigned to integer and fractional parts is always fixed in this representation. This causes limitations both in the range of numbers to be represented and their resolution. Floating-point representation can be used to overcome these problems. As the name implies, the number of bits assigned to integer and fractional parts is not fixed in this representation. Instead, the assigned number of bits differ for each number depending on its significant digits. Therefore, a much wider range of values can be represented in this form.

In floating-point representation, a binary number with fractional part will be shown as $N = (-1)^S \times 2^E \times F$. Here, S stands for the sign bit, E represents the exponent value, and F stands for the fractional part. Then, floating-point number N is kept in memory as $X = SEF$.

To represent a floating-point number as $N = (-1)^S \times 2^E \times F$, the number should be normalized such that the integer part will have one digit. For ease of binary representation, the exponent will be biased by $2(^{e-1}) - 1$, where e is the number of bits to be used for E in the given format. Finally, certain number of bits will be assigned to S, E, and F depending on the standard format used for representation. The IEEE 754 standard is used by most digital systems in floating-point representation. This standard is summarized in Table 6.5.

Format	Exponent bias	# bits for S	# bits for E	# bits for F	# total bits
Half	15	1	5	10	16
Single	127	1	8	23	32
Double	1023	1	11	52	64
Quad	16383	1	15	112	128

TABLE 6.5 The IEEE 754 Standard for Floating-Point Representation

Let's take the decimal number 14.125 and represent it in floating-point form. We will follow the below itemized procedure for this purpose:

- Decide on the format: Let's pick the "half" format for this example.
- Represent the integer and fractional parts of the decimal number in binary form: The number becomes `1110.001`.
- Decide on the sign bit S: Since the number is positive, $(-1)^0 = 1$, `S=0`.
- Normalize the number such that the integer part will have one digit: The number becomes `1.110001`$\times 2^3$.
- Find the exponent value: For the half format, the exponent bias is 15. Therefore, the exponent will become $E = 15 + 3 = 18$ with bias. Or, in binary form `E=10010`.
- Find the fractional part: The fractional part (after normalization) was `110001`. Since 10 bits should be used to represent the fractional part of the number in half format, `F=1100010000`. Remember, since this is the fractional part, we append extra zeros to its right so that the value of the number is not affected.
- Construct $X = SEF$: Finally, `X = 0 10010 1100010000`. Or in hexadecimal form, `X=4B10`.

Next, let's represent the decimal number −14.125 in floating-point form. As in the previous example, let's use the half format. Then, the only change will be in the sign bit. As a result, the number will become `X = 1 10010 1100010000`. Or in hexadecimal form, `X=CB10`.

6.4 ASCII Code

We do not only process numbers in digital systems. For some applications, we may need to handle characters and symbols as well. We know that everything in a digital system is represented in binary form. Therefore, characters and symbols should also be represented as such. One way of representing characters and symbols in binary form is using the ASCII code. ASCII stands for the American Standard Code for Information Interchange. The ASCII code for characters and symbols are given in Table 6.6. In this table, LSB stands for least significant byte and MSB stands for most significant byte. To represent a specific character (or symbol), its corresponding code should be given.