# Synchronous CDMA

Synchronous CDMA exploits mathematical properties of orthogonality between vectors representing the data strings. For example, binary string "1011" is represented by the vector (1, 0, 1, 1). Vectors can be multiplied by taking their dot product, by summing the products of their respective components. If the dot product is zero, the two vectors are said to be orthogonal to each other. (Note: If u=(a,b) and v=(c,d), the dot product u.v = a*c + b*d) Some properties of the dot product help to understand how WCDMA works. If vectors a and b are orthogonal, then

Each user in synchronous CDMA uses an orthogonal codes to modulate their signal. An example of four mutually orthogonal digital signals is shown in the figure. Orthogonal codes have a cross-correlation equal to zero; in other words, they do not interfere with each other. In the case of IS-95 64 bit Walsh codes are used to encode the signal to separate different users. Since each of the 64 Walsh codes are orthogonal to one another, the signals are channelized into 64 orthogonal signals. The following example demonstrates how each users signal can be encoded and decoded.

Example

Start with a set of vectors that are mutually orthogonal. (Although mutual orthogonality is the only condition, these vectors are usually constructed for ease of decoding, for example columns or rows from Walsh matrices.) An example of orthogonal functions is shown in the picture on the left. These vectors will be assigned to individual users and are called the "code", "chipping code" or "chip code". In the interest of brevity, the rest of this example uses codes (v) with only 2 digits.

An example of four mutually orthogonal digital signals.

Each user is associated with a different code, say v. If the data to be transmitted is a digital zero, then the actual bits transmitted will be –v, and if the data to be transmitted is a digital one, then the actual bits transmitted will be v. For example, if v=(1,–1), and the data that the user wishes to transmit is (1, 0, 1, 1) this would correspond to (v, –v, v, v) which is then constructed in binary as ((1,–1),(–1,1),(1,–1),(1,–1)). For the purposes of this article, we call this constructed vector the transmitted vector.

Each sender has a different, unique vector v chosen from that set, but the construction method of the transmitted vector is identical.

Now, due to physical properties of interference, if two signals at a point are in phase, they add to give twice the amplitude of each signal, but if they are out of phase, they "subtract" and give a signal that is the difference of the amplitudes. Digitally, this behaviour can be modelled by the addition of the transmission vectors, component by component.

If sender0 has code (1,–1) and data (1,0,1,1), and sender1 has code (1,1) and data (0,0,1,1), and both senders transmit simultaneously, then this table describes the coding steps:

| Step | Encode sender0 | Encode sender1 |
|---|---|---|
| 0 | vector0=(1,–1), data0=(1,0,1,1)=(1,–1,1,1) | vector1=(1,1), data1=(0,0,1,1)=(–1,–1,1,1) |
| 1 | encode0=vector0.data0 | encode1=vector1.data1 |
| 2 | encode0=(1,–1).(1,–1,1,1) | encode1=(1,1).(–1,–1,1,1) |
| 3 | encode0=((1,–1),(–1,1),(1,–1),(1,–1)) | encode1=((–1,–1),(–1,–1),(1,1),(1,1)) |
| 4 | signal0=(1,–1,–1,1,1,–1,1,–1) | signal1=(–1,–1,–1,–1,1,1,1,1) |

Because signal0 and signal1 are transmitted at the same time into the air, they add to produce the raw signal:
(1,–1,–1,1,1,–1,1,–1) + (–1,–1,–1,–1,1,1,1,1) = (0,–2,–2,0,2,0,2,0)

This raw signal is called an interference pattern. The receiver then extracts an intelligible signal for any known sender by combining the sender's code with the interference pattern, the receiver combines it with the codes of the senders. The following table explains how this works and shows that the signals do not interfer with one another:

| Step | Decode sender0 | Decode sender1 |
|---|---|---|
| 0 | vector0=(1,–1), pattern=(0,–2,–2,0,2,0,2,0) | vector1=(1,1), pattern=(0,–2,–2,0,2,0,2,0) |
| 1 | decode0=pattern.vector0 | decode1=pattern.vector1 |
| 2 | decode0=((0,–2),(–2,0),(2,0),(2,0)).(1,–1) | decode1=((0,–2),(–2,0),(2,0),(2,0)).(1,1) |
| 3 | decode0=((0+2),(–2+0),(2+0),(2+0)) | decode1=((0–2),(–2+0),(2+0),(2+0)) |
| 4 | data0=(2,–2,2,2)=(1,0,1,1) | data1=(–2,–2,2,2)=(0,0,1,1) |

Further, after decoding, all values greater than 0 are interpreted as 1 while all values less than zero are interpreted as 0. For example, after decoding, data0 is (2,–2,2,2), but the receiver interprets this as (1,0,1,1).

We can also consider what would happen if a receiver tries to decode a signal when the user has not sent any information. Assume signal0=(1,-1,-1,1,1,-1,1,-1) is transmitted alone. The following table shows the decode at the receiver:

| Step | Decode sender0 | Decode sender1 |
|---|---|---|
| 0 | vector0=(1,–1), pattern=(1,-1,-1,1,1,-1,1,-1) | vector1=(1,1), pattern=(1,-1,-1,1,1,-1,1,-1) |
| 1 | decode0=pattern.vector0 | decode1=pattern.vector1 |
| 2 | decode0=((1,–1),(–1,1),(1,-1),(1,-1)).(1,–1) | decode1=((1,–1),(–1,1),(1,-1),(1,-1)).(1,1) |
| 3 | decode0=((1+1),(–1-1),(1+1),(1+1)) | decode1=((1–1),(–1+1),(1-1),(1-1)) |
| 4 | data0=(2,–2,2,2)=(1,0,1,1) | data1=(0,0,0,0) |

When the receiver attempts to decode the signal using sender1‘s code, the data is all zeros, therefore the cross correlation is equal to zero and it is clear that sender1 did not transmit any data.

Source : http://nprcet.org/e%20content/cse/ADC.pdf