DETAILS

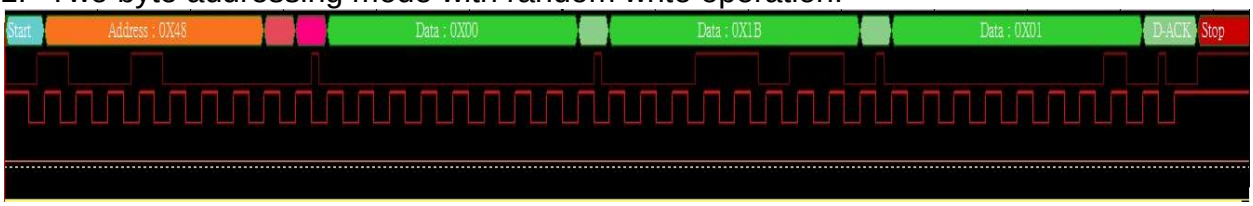
FX3 has I2C master component build inside, and the SDK provides I2C read/write functions to communicate with I2C slave device. We can use the SDK APIs to generate I2C waveforms for communication with devices which use one byte and two bytes addressing mode.

The basic APIs are CyU3PI2cTransmitBytes and CyU3PI2cReceiveBytes functions. This two APIs use a data structure, CyU3PI2cPreamble_t, to control the output waveform. CyU3PI2cPreamble_t is composed of a eight bytes buffer, one byte data and two bytes mask information. The eight bytes buffer represents the slave address and memory address we want to send by I2C waveforms. In random address write operation, we have to fill up buffer[0] with the slave address for writing, buffer[1] and buffer[2] need to be filled up with memory address according the addressing mode of slave(if in two bytes addressing mode, buffer[1] is for higher byte of memory address and buffer[2] is for lower byte). And we have to indicate the real data length of the buffer in the one byte data, then we can use the APIs to write a data buffer into the I2C device.

In random read operation, because locating memory address is needed, we have to use the same data as write, and fill up another field in the eight bytes buffer with the slave address for read. And most import, we have to take care of the mask information in CyU3PI2cPreamble_t structure. In random address read mode, after sending the slave address and memory address(one byte or two bytes) for write, we have to restart a I2C communication by raise a start condition which can be controlled by set the corresponding bit of the mask field. For example, in two byte mode, we can set the mask field as 0x0004. It means after sending slave address for write and two bytes memory address, there will be a start signal, then FX3 will send the slave address for read and the slave will output the data.

The following section will show the typical I2C waveforms and the sample code which produce these waveform.

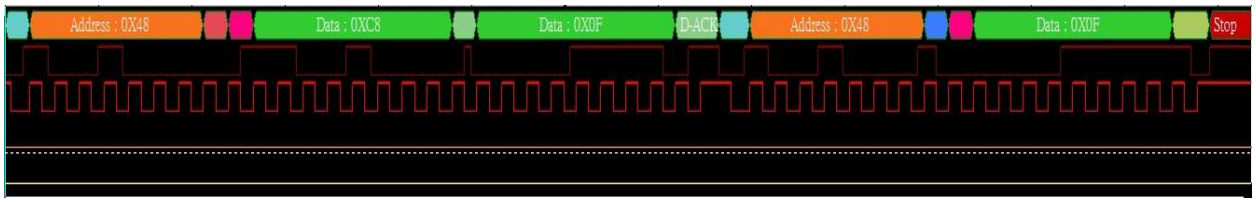
1. Two byte addressing mode with random write operation:



Sample code:

```
preamble.buffer[0] = SlaveAddr; /* Slave address: Write operation */
preamble.buffer[1] = (uint8_t)((MenAddr & 0Xff00) >> 8);
preamble.buffer[2] = (uint8_t)( MenAddr & 0x00ff);
preamble.length = 3;
preamble.ctrlMask = 0x0000;
CyU3PI2cTransmitBytes (&preamble, testbuf, 1, 0);
delay(800);
```

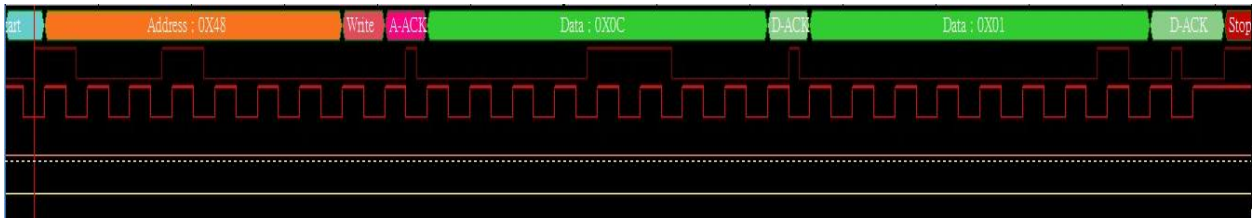
2. Two byte addressing mode with random read operation:



Sample code:

```
        preamble.buffer[0] = SlaveAddr; /* Slave address: Write operation */
        preamble.buffer[1] = (uint8_t)(( MenAddr & 0Xff00) >> 8);
        preamble.buffer[2] = (uint8_t)( MenAddr & 0x00ff);
        preamble.buffer[3] = SlaveAddr | 0x1; /* Slave address: Read operation
*/
        preamble.length = 4;
        preamble.ctrlMask = 0x0004;
        CyU3PI2cReceiveBytes (&preamble, testbuf, 1,0);
        delay(800);
```

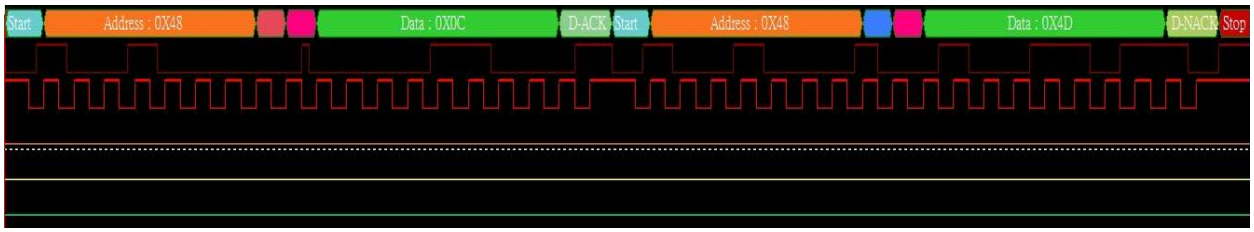
3. One byte addressing mode with random write operation:



Sample code:

```
        preamble.buffer[0] = SlaveAddr; /* Slave address: Write operation */
        preamble.buffer[1] = (uint8_t)MenAddr;
        preamble.length = 2;
        preamble.ctrlMask = 0x0000;
        CyU3PI2cTransmitBytes (&preamble, testbuf, 1, 0);
        delay(800);
```

4. One byte addressing mode with random write operation:



Sample code:

```
        preamble.buffer[0] = SlaveAddr; /* Slave address: Write operation */
        preamble.buffer[1] = (uint8_t)MenAddr;
        preamble.buffer[2] = SlaveAddr | 0x1; /* Slave address: Read operation
*/
```

```
preamble.length = 3;
preamble.ctrlMask = 0x0002;
CyU3PI2cReceiveBytes (&preamble, testbuf, 1,0);
delay(800);
```