Wiring code

4.5.5 DDC Port

The Display Data Channel (DDC) bus is a two-wire communication interface compatible with the I²C standard (Integrated Circuit). See our on-line product selector matrix for availability.

Unlike all other interfaces, the DDC is not able to communicate in full-duplex mode, i.e. TX and RX are mutually exclusive. u-blox receivers act as a slave in the communication setup, therefore they cannot initiate data transfers on their own. The host, which is always master, provides the data clock (SCL), and the clock frequency is therefore not configurable on the slave.

The receiver's DDC address is set to 0x42 by default.

As the receiver will be run in slave mode and the DDC physical layer lacks a handshake mechanism to inform the master about data availability, a layer has been inserted between the physical layer and the UBX and NMEA layer. The receiver DDC interface implements a simple streaming interface that allows the constant polling of data, discarding everything that is not parse-able. The receiver returns 0xFF if no data is available. The TX-ready feature can be used to inform the master about data availability and can be used as a trigger for data transmission.

4.5.5.1 Read Access

The DDC interface allows 256 slave registers to be addressed. As shown in Figure *DDC Register Layout* only three of these are currently implemented. The data registers 0 to 252, at addresses 0x00 to 0xFC, each 1 byte in size, contain information to be defined later - the result of reading them is undefined. The currently available number of bytes in the message stream can be read at addresses 0xFD and 0xFE. The register at address 0xFF allows the data stream to be read. If there is no data awaiting transmission from the receiver, then this register will deliver the value 0xff, which cannot be the first byte of a valid message. If message data is ready for transmission, then successive reads of register 0xff will deliver the waiting message data.

The registers 0x00 to 0xFC are reserved for future use and may be defined in a later firmware release. Do not use them, as they don't provide any meaningful data!

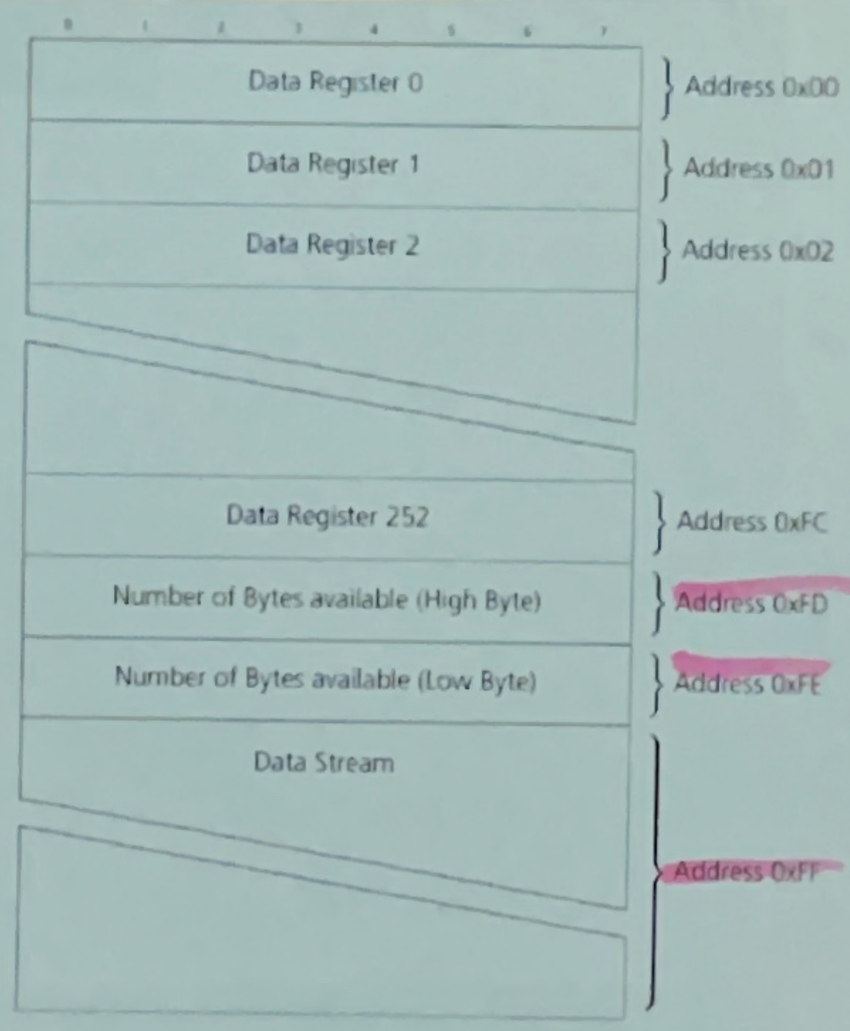

Figure 21: DDC Register Layout

Connected the ZED-F9P to the arduino giga using the I2C bus.

} we want to read these

88

Reading the Number of Available Bytes ZEDF9P

06/09/25
Wrong code

Code:

```ino
#include <Wire.h>
const int gps_add = 0x42;

uint16_t GPSAvailable()
{
  // This method reads the number of available bytes that we can read from the GPS.

  Wire.beginTransmission(gps_add);
  Wire.write(0xFD);              // The FD register is the high byte. We will read FD, and FE in order.
  Wire.endTransmission(false);  // Must not release the bus. Must finish reading.

  Wire.requestFrom(gps_add, 2); // We need 2 bytes from the GPS

  byte msb = Wire.read();       // Read Register 0xFD
  byte lsb = Wire.read();       // Read Register 0xFE
                                // After this operation, the GPS read pointer will be back at 0xFF.
  uint16_t bytesAvailable = (msb<<8) | lsb; // Put humpty dumpty back together
  return bytesAvailable;
}

void setup()
{
  Wire.begin();       // Start the I2C bus
  Serial.begin(9600); // Start serial comms
}

void loop()
{
  // If the reciever is configured to poll
  // We should expect 0 every time.
  uint16_t availableBytes = GPSAvailable();
  Serial.println(availableBytes);
  delay(1500);
}
```