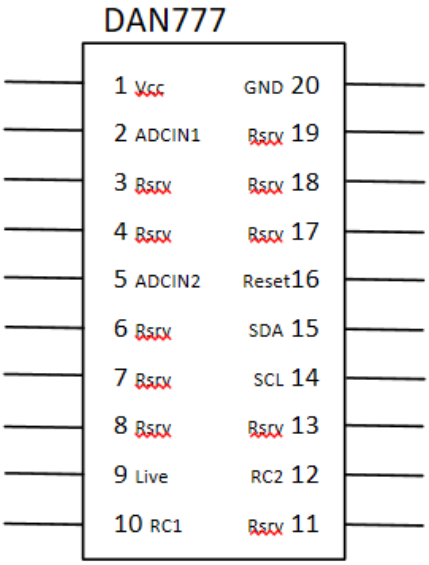
DAN777

Two 10 bit ADC Inputs and Two RC Servo Motor PWM Outputs with I2C Interface

Features:

- Operating Voltage 3.3Volts.
- Two 10 bit SAR ADCs with 0V to 3.3V range.
- Two PWM Outputs for driving hobby RC Servo Motors, with 4000 steps of resolution. 0.6 millisecond pulse to 2.6 millisecond pulse. 2.0ms/4000 = 0.0005ms resolution
- I2C Interface, Standard SDA and SCL connection
- I2C maximum SCLK tested rate of 250 KHz.
- I2C Slave 7-bit Address 0x25



General Description

The DAN777 is an Integrated Circuit chip that adds two 10 bit ADC inputs and two PWM outputs for driving RCservo motors to your embedded system. It uses standard 2-wire I2C serial communication and the DAN777 is an I2C slave device.

The two ADC channels are SAR (Successive Approximation Register) type ADCs with an input range of 0 to 3.3Volts. With 10 bits, that gives an input resolution of 3.3Volts/1023 steps. The DAN777 continuously samples the two ADC channels every 1 millisecond and the most current samples are transmitted to the Master when the I2C Master issues a read command. The two PWM outputs, RCSERVO1 and RCSERVO2, are specifically setup for driving RC servo motors. RC servo motors have an internal potentiometer for angle feedback. This allows the RC servo to move to a desired angle and hold there. To command an RC servo to a desired angle a repeating 3.3V pulse width drives the signal pin. This pulse needs to be repeated at least every 15ms. The DAN777 repeats the pulse every 20ms. Each brand of RC servo motor is slightly different when it comes to what pulse width commands a certain motor angle so a bit of experimenting is needed when driving a new RC servo motor. For example a pulse width around 1.6 ms. should command the RC servo motor to its zero angle. A pulse width around 0.6 ms should command a -90 degree angle and a pulse width around 2.6 ms should command a 90 degree angle. Instead of thinking in terms of the time of the pulse width as the command, %duty cycle of a PWM signal can also be used to describe the input. The PWM signal has a carrier frequency of 50 Hz or 0.02s period and a range of duty cycles from 3% to 13%.

Device Pins

Vcc: 3.3V power for the IC

Gnd: Ground of the 3.3V power

ADC1: ADC channel one's input pin. Accepts a voltage in the range of 0 to 3.3 volts. Value communicated over I2C is an integer with the range of 0 to 1023 where 0 equals 0 volts and 1023 equals 3.3volts.

ADC2: ADC channel two's input pin. Accepts a voltage in the range of 0 to 3.3 volts. Value communicated over I2C is an integer with the range of 0 to 1023 where 0 equals 0 volts and 1023 equals 3.3volts.

RC1: RC servo motor command output one. RC servos are driven by pulse widths in the range of 0.6 ms to 2.6 ms. These pulses are repeated every 20 ms. So you can also think of this as a PWM signal with carrier frequency 50Hz and duty cycles ranging from 3% to 13%. The value communicated over I2C is a an integer with the range of 1200 to 5200 where 1200 equals 3% and 5200 equals 13% duty cycle. 3% duty cycle approximately commands an angle of -90 degrees. 8% approximately commands an angle of 0 degrees. 13% approximately commands an angle of 90 degrees. If the DAN777 receives a value less than 1200, the value of 1200 or 3% duty cycle is commanded. If the DAN777 receives a value greater than 5200, the value of 5200 or 13% duty cycle is commanded.

RC2: RC servo motor command output two. RC servos are driven by pulse widths in the range of 0.6 ms to 2.6 ms. These pulses are repeated every 20 ms. So you can also think of this as a PWM signal with carrier frequency 50Hz and duty cycles ranging from 3% to 13%. The value communicated over I2C is a an integer with the range of 1200 to 5200 where 1200 equals 3% and 5200 equals 13% duty cycle. 3% duty cycle approximately commands an angle of -90 degrees. 8% approximately commands an angle of 0 degrees. 13% approximately commands an angle of 90 degrees. If the DAN777 receives a value less than 1200, the value of 1200 or 3% duty cycle is commanded. If the DAN777 receives a value greater than 5200, the value of 5200 or 13% duty cycle is commanded.

SDA: I2C Data Line. First, the address of the desired chip to communicate with is sent on this data line. Then, either data is sent from the I2C Master to the I2C Slave (Master Wright) or sent from the I2C Slave to the I2C Master. (Slave Wright). The SDA line is held at 3.3V buy a pull up resistor when no serial transfer is occurring.

SCL: I2C Clock line. This pin/clock is always controlled by the master. The I2C slave simply monitors the SCL line. The I2C slave does not have any control over this clock line. The SCL line is held at 3.3V buy a pull up resistor when no serial transfer is occurring.

LED: Is the DAN777 running status. This is a useful pin that indicates if the DAN777 is operating. This pin should be wired to an LED. When powered, the DAN777 will blink on and off this LED.

Reset Button: This is the pushbutton on the DAN777 board, used to reset the program running on the DAN777 back to the beginning of its code. This is a nice feature when debugging the I2C Master's code. Whenever the I2C Master code is restarted, the DAN777 should be reset by pressing this button, in order that the I2C communication is in sync.

Rsrv: Reserved for future use in new releases of the DAN777.

Registers in the DAN777

ADC1LSB:	Register Address 0
	Least Significant byte of ADC1's value (Read Only)
ADC1MSB:	Register Address 1

Most Significant byte of ADC1's value (Read Only)

ADC2LSB: Register Address 2

Least Significant byte of ADC2's value (Read Only)

ADC2MSB Register Address 3

Most Significant byte of ADC2's value (Read Only)

RCSERVO1LSB: Register Address 4

Least Significant byte of RCSERVO1's value

RCSERVO1MSB: Register Address 5

Most Significant byte of RCSERVO1's value

RCSERVO2LSB: Register Address 6

Least Significant byte of RCSERVO2's value

RCSERVO2MSB: Register Address 7

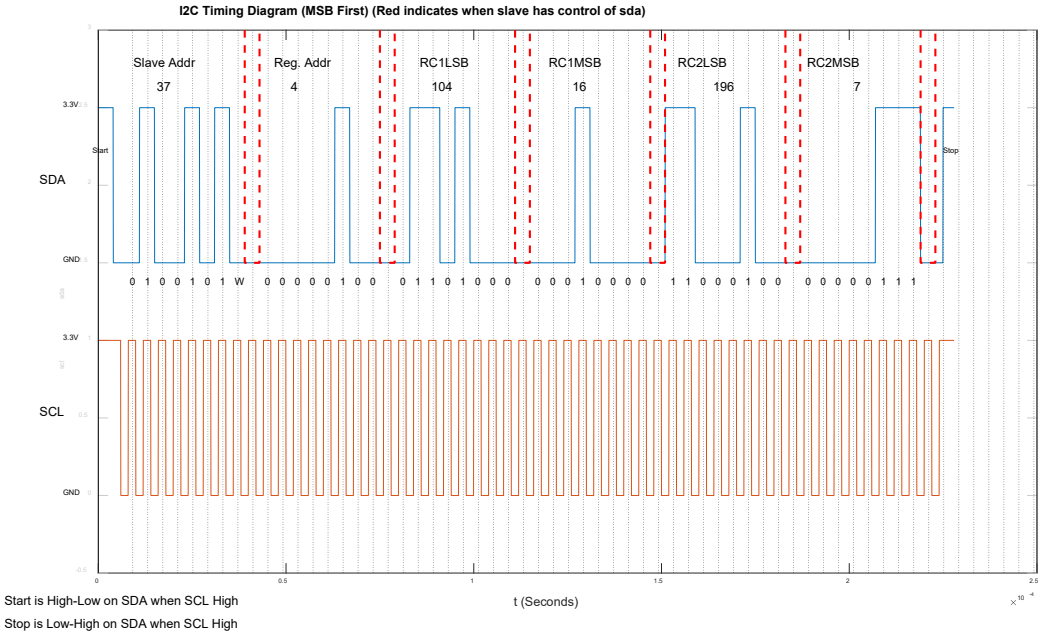
Most Significant byte of RCSERVO2's value

Timing Diagrams / Communication Flow

Below you find the communication protocol using the I2C interface for the DAN777. Note that the 4 byte write and the 4 byte read of the DAN777 are currently the only well tested communication sequence for I2C and the DAN777 chip. Reading/writing just one byte or two bytes at a time should also work but may need some debugging. I am leaving this work for part of a final project at the end of the semester if you are interested in learning more about the I2C serial port.

For this reason I am only documenting the 4 byte read starting at register address 0 and the 4 byte write starting at register address 4. These are the only two communication schemes I would like you to implement for this project.

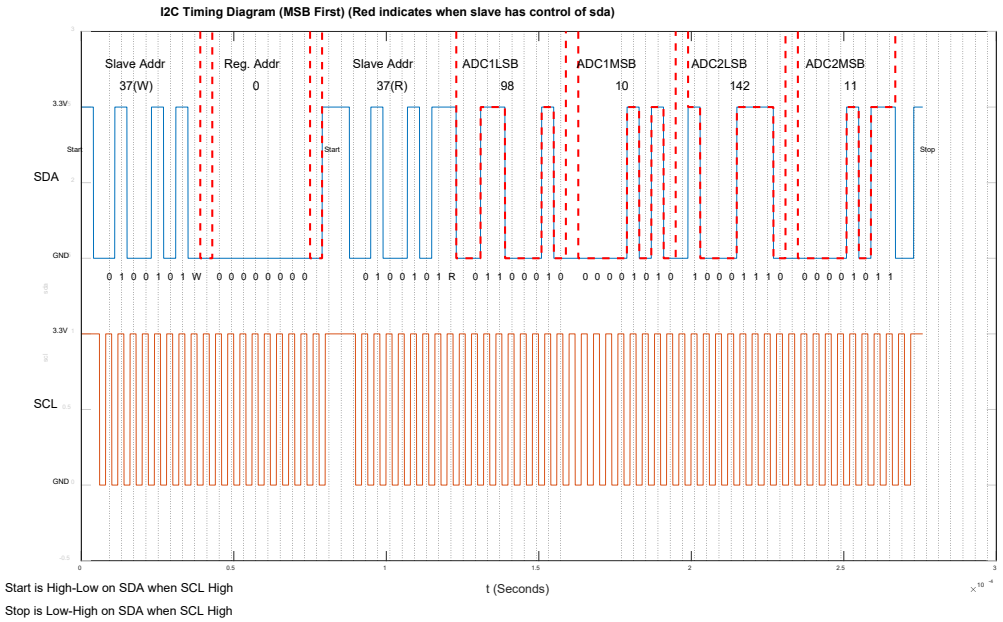
I2C Interface Protocol: Write PWM Command to RCServo1 & RCServo2



To write new RCServo commands to the DAN777 follow the above timing graph along with the below procedure. Each time the I2C master (your code) needs to write new RCSERVO values to the DAN777:

1. Since the RCServo commands are a value between 1200 and 5200 a 16 bit integer is needed to store these values inside the DAN777 chip. The largest bit size for one transfer with the I2C serial port is 8 bits. This is why the DAN777 is requesting an RCservo1 LSB (Least Significant Byte) and RCServo1 MSB (Most Significant Byte) and the same for RCServo2. So as a first step, take the RCServo1 and RCServo2 command and divide them into their LSB and MSB. Probably smart to create variables like rc1lsb, rc1msb, etc.
2. Check if the BB (busy) bit in the I2CSTR is set. If it is set the I2C peripheral is busy. Loop, waiting of the port to not be busy.
3. Loop, waiting for the XRDY bit in I2CSTR to become 1 (Transmit Ready).
4. Set Slave Address to 0x25
5. Setup to send 5 bytes, where Byte1: Register Address = 4, Byte2: RC1LSB, Byte3: RC1MSB, Byte4: RC2LSB, Byte5: RC2MSB.
6. Send Register Address to I2CDXR.
7. Set I2CMDR to issue a Start condition, be in the transmit mode and when all 5 bytes sent issue a Stop condition.
8. Loop, waiting for XRDY bit in I2CSTR to become 1 (Transmit Ready).
9. Send RC1LSB to I2CDXR
10. Loop, waiting for XRDY bit in I2CSTR to become 1 (Transmit Ready).
11. Send RC1MSB to I2CDXR
12. Loop, waiting for XRDY bit in I2CSTR to become 1 (Transmit Ready).
13. Send RC2LSB to I2CDXR
14. Loop, waiting for XRDY bit in I2CSTR to become 1 (Transmit Ready).
15. Send RC2MSB to I2CDXR

I2C Interface Protocol: Read 10bit Conversion from ADC1 & ADC2



To read the two ADC values from the DAN777, follow the above timing graph along with the below procedure. Each time the I2C master (your code) needs to read new ADC values from the DAN777:

1. Since the ADC values are between 0 and 1023, a 16 bit integer is needed to store these values inside the DAN777 chip. The largest bit size for one transfer with the I2C serial port is 8 bits. This is why the DAN777 has an ADC1 LSB (Least Significant Byte) register and an ADC1 MSB (Most Significant Byte) register and the same for ADC2. So as a first step, declare some int16_t variables to receive the four byte. For example create variables like adc1lsb, adc1msb, etc.
2. Check if the BB (busy) bit in the I2CSTR is set. If it is set the I2C peripheral is busy. Loop, waiting of the port to not be busy.
3. Loop, waiting for the XRDY bit in I2CSTR to become 1 (Transmit Ready).
4. Set Slave Address to 0x25
5. Setup to send 1 byte, where Byte1: Register Address = 0
6. Send Register Address to I2CDXR.
7. Set I2CMDR to issue a Start condition, be in the transmit mode and when the 1 byte is sent issue a Stop condition. (But we will be issuing another Start condition so the Stop condition will be delayed until the end of the read transfers).
8. Loop, waiting for XRDY bit in I2CSTR to become 1.
9. Now another Start condition needs to be sent to receive data starting from the Register Address just sent above, so again set the Slave Address to 0x25. Set to receive 4 bytes. Set I2CMDR to issue Start condition, be in receive mode and issue a Stop condition when all bytes have been transferred.
10. Loop, waiting for RRDY bit in I2CSTR to become 1.
11. Read ADC1LSB from I2CDRR
12. Loop, waiting for RRDY bit in I2CSTR to become 1.
13. Read ADC1MSB from I2CDRR
14. Loop, waiting for RRDY bit in I2CSTR to become 1.
15. Read ADC2LSB from I2CDRR
16. Loop, waiting for RRDY bit in I2CSTR to become 1.
17. Read ADC2MSB from I2CDRR
18. Combine the LSB and MSB bytes to form the ADC1 and ADC2 values.