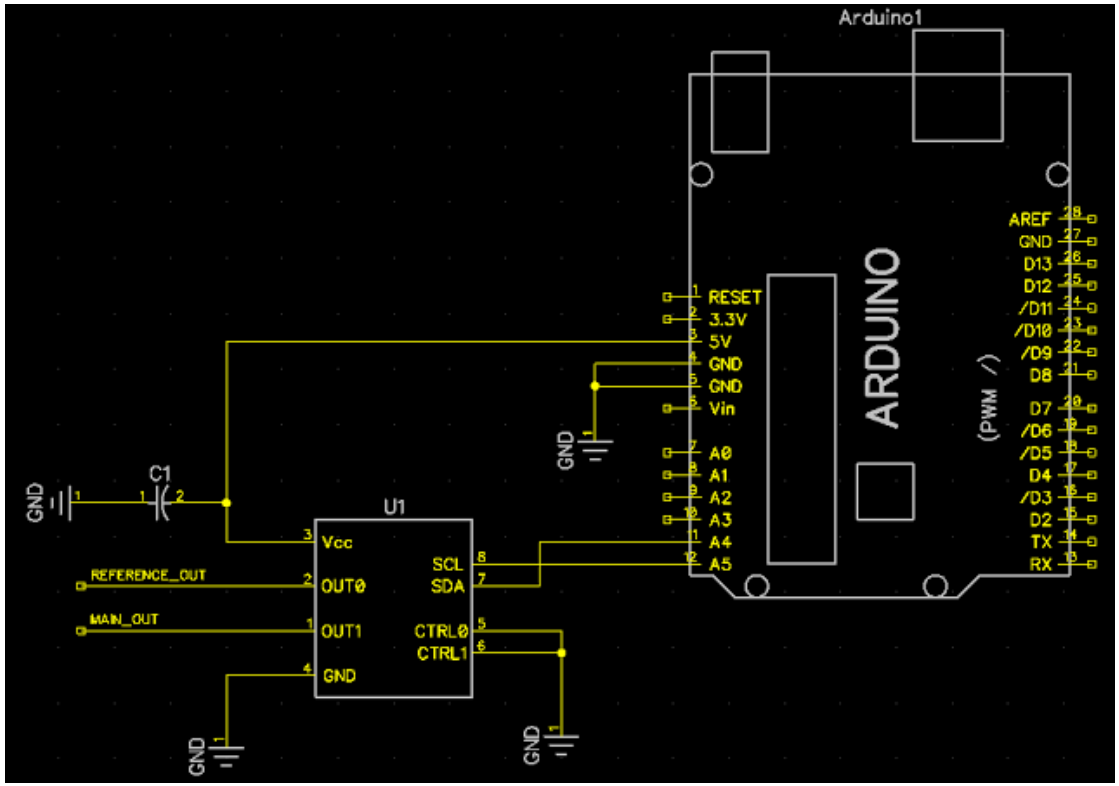
Arduino + DS1077

DS1077 permanently attached to Arduino

In this case you can use the Arduino to set directly the wanted frequency output.
The circuit to get it working is very simple, just hook up the Arduino as the I2C master, I used a 10K pullup resistor on both the SCL and SDA lines. I tied CTR0 and CTR1 on the DS1077 to GND to enable the chip all the time. For the following video, I put the scope probe onto the OUT1 pin...

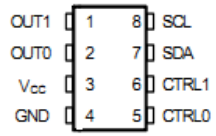
This chip produces rather good quality square waves signals when under ideal circumstances. Here's a quick schematic for that hastily breadboarded demo circuit...



I like the pinout of this chip, as it clearly demonstrates deference to signal integrity design considerations for the PCB designer. Notice how the VCC and GND pins are side by side. That makes it very convenient to place a decoupling cap right on those lines as close to the chip as possible.

DS1077
EconOscillator/Divider

PIN ASSIGNMENT



150mil SO
118mil µSOP Package

PIN DESCRIPTION

OUT1	- Main Oscillator Output
OUT0	- Reference Output
V_{CC}	- Power Supply Voltage
GND	- Ground
CTRL1	- Control Pin for OUT1
CTRL0	- Control Pin for OUT0
SDA	- 2-Wire Serial Data Input/Output
SCL	- 2-Wire Serial Clock

(Excerpt from Maxim/Dallas Datasheet)

And of course here's the obligatory Arduino source code to drive the chip. All it does is run through the entire frequency range supported by this chip.

```
#include <Wire.h>           //address: the 7-bit slave address (optional);

const int ds_address = 0xB0 >> 1; //DS1077 default address

#define MUX_HI_PDN1_BIT (B01000000)
#define MUX_HI_PDN0_BIT (B00100000)
#define MUX_HI_SEL0_BIT (B00010000)
#define MUX_HI_EN0_BIT  (B00001000)
#define MUX_HI_0M1_BIT  (B00000100)
#define MUX_HI_0M0_BIT  (B00000010)
#define MUX_HI_1M1_BIT  (B00000001)

#define MUX_LO_1M0_BIT             (B10000000)
#define MUX_LO_DIV1_BIT            (B01000000)

#define BIT_ON (0xff)
```

```
#define BIT_OFF (0x00)


void setup()
{
  Wire.begin();
}

void loop()
{
  long i;
  for(i = 0 ; i <= 1025 ; i+=1)
  {
    i2c_write(ds_address, 0x02,
      ( 0
        | (MUX_HI_PDN1_BIT & BIT_OFF)
        | (MUX_HI_PDN0_BIT & BIT_OFF)
        | (MUX_HI_SEL0_BIT & BIT_ON)
        | (MUX_HI_EN0_BIT  & BIT_ON)
        | (MUX_HI_0M1_BIT  & BIT_ON)
        | (MUX_HI_0M0_BIT  & BIT_ON)
        | (MUX_HI_1M1_BIT  & BIT_ON)
      )
      ,
      ( 0
        | (MUX_LO_1M0_BIT  & BIT_ON)
        | (MUX_LO_DIV1_BIT & BIT_OFF)
      )
    ); //mux
    delayMicroseconds(100);

    i2c_write(ds_address, 0x01, (i >> 2), (i << 6) & 0xC0); //div
    delayMicroseconds(100);
    i2c_write(ds_address, 0x3F);

    delayMicroseconds(10000 - i);
  }
}

void i2c_write(int device, byte address) {
     Wire.beginTransmission(device); //start transmission to device
     Wire.write(address);        // send register address
     Wire.endTransmission(); //end transmission
}

void i2c_write(int device, byte address, byte val1) {
```

```
     Wire.beginTransmission(device); //start transmission to device
     Wire.write(address);        // send register address
     Wire.write(val1);        // send value to write
     Wire.endTransmission(); //end transmission
}

void i2c_write(int device, byte address, byte val1, byte val2) {
     Wire.beginTransmission(device); //start transmission to device
     Wire.write(address);        // send register address
     Wire.write(val1);        // send value to write
     Wire.write(val2);        // send value to write
     Wire.endTransmission(); //end transmission
}
```

This chip lets you output a 50% duty cycle square wave at any frequency given by 133,000,000 divided by any number between 1 and 1025 inclusive. This divider is selected by sending a short I2C sequence to the chip from whatever microcontroller you prefer. I'm using an Arduino here because there was some code easily available for it.

Programming the Arduino for DS1077

In this case The code referes to a program executed by Arduino 2 for programming DS1077 chip. This chip can be detached from Arduino and used as a single precision squarewave generator.

```
/*
Colonna (RM) ITALY ---> 07 - 20 - 2014  Rev. 1.0  time: 17:28
This is an "Arduino DUE"  Programmable Frequency Generator project and run on Arduino IDE 1.5.7
The MAIN Reference is the DS1077 original data sheet in pdf format. There you find all the programming details...even if in a low understandable format!
Ideas taken from following web sites and Arduino forum:
http://playground.arduino.cc/Main/InterfacingWithHardware#i2c
http://forum.arduino.cc/index.php?PHPSESSID=ndhn4hh97dk97tnkovuf3m1dj1&topic=75323.msg569607#msg569607
http://playground.arduino.cc/Main/I2CBi-directionalLevelShifter
http://hackaday.com/2008/11/28/parts-133mhz-162khz-programmable-oscillator-ds1077/
http://jondontdoit.blogspot.it/2012/08/using-ds1077-programmable-oscillator.html
http://wardyprojects.blogspot.it/2013/03/ds1077-arduino.html

The DS1077 is a 5volt, 133MHz to 16kHz programmable clock source. The internal frequency divider is configured over a simple I2C/TWI serial bus interface, and the chip requires no external parts.
It is possible to buy the chip already soldered in a breakout small board...that can be easily soldered on Arduino empty board shield.
Of course you could need Arduino DUE (as for my case) only for first programming as once programmed the chip outputs automatically  the desired frequency (that is stored in chip's internal EEPROM)!
Connecting the DS1077 to Arduino DUE I2C serial bus is  a problem as the first it is a 5V device while the second is a 3.3V device.  The best solution is to use a 'bidirectional level translator' or a 3.3V DS1077 version (difficult found...for the moment). Note: now 5v versions are also available on the market.
DS1077 is available also as a breakout part! Any way, even if the Arduino Due SDA and SCL pin are internally pulled up to 3.3V... there could be problems I tried to pull them up externally to 5V using a 6.8KOhm resistors: it works for me!.

Pin connections:
DS1077  (pin)
SCL      SCL (21)
SDA      SDA(20)
AUX      OUT1 (1)
+5volts Vcc (3)
GND      GND, CTRL (4,5,6)

Note:
R1 and R2 (6.8k), pull-up the I2C bus to 5volts when it's not in use.
```

Vcc /GND Bypass Capacitor C1 is 0.1uF, as recommended by the datasheet.
Control pins provide some additional functions, but I connected them to ground in order to let output enabled.
Out11 is the primary clock signal pin. Out2 is the so called Reference output.

Interfacing:

Address	Purpose:
0b10110000	Default base address (0xB0)
0xB0	Write address
0xB1	Read address

Arduino 'Wire' library allows you to communicate with I2C / TWI devices. On most Arduino boards, SDA (data line) is pin 20, and SCL (clock line) on pin 21.
Frequency output is on OUT1 pin (OUT0 pin is for reference frequency).

Address	Bytes	Register
0×01	2	10 bit clock divider, n+2 (DIV)
0×02	2	Prescaler, CTRL pin functions. (MUX)
0x0D	1	Address select, write control. (BUS)
0x3F	0	Save settings to EEPROM (E2)

The DS1077 is controlled by writing values to the locations shown in the table (in data sheet). It is mandatory to follow this sequence:
1 - first initialize the DS1077 issuing a Write.begin()...now you joined the I2C bus
2 - define the DS1077 base address that is: ...0xB0 or if you want: B1011 0000
3 - initialize properly, by setting the appropriate bit of MUX register (as per your necessity)...remembering to set also the DIV1 bit!
4 - set properly the 0M0, 1M1, 1M0, PDN0, PDN1, SEL0 and EN0 bit as necessary
5 - set the desired DIV factor for the DIV1 register (a 10 bit word divided into two bytes: HiByte and LowByte)

By default, the DS1077 saves all changes to the EEPROM. We don't need this during testing session, so we disable it by setting bit 3 (0b1000) of the BUS register (0x0d).
The first four bits must be left as 0, the last three bits select the address to accommodate multiple DS1077s on the same I2C bus (if any). See datasheet page 7.

Programming Examples;
The complete values for MUX/prescaler are: 0xb0 0×02 B00011000 B00000000 <–set the default 16bit MUX value.
The MUX register controls the prescalers, CTRL pin functions, and frequency divider.
The MUX register is explained on page 5 of the datasheet. Specific frequencies are generated by dividing the 133MHz reference frequency through the prescalers and a 10bit (1025 level) programmable divider. The clock is divided by the amount specified in the DIV register, plus two. When DIV=0, the output is 133MHz/2=66MHz.

This scheme gives the best frequency resolution in low ranges, and no steps between 133MHz and 66MHz
0xb0 1 0b11111111 0b11000000 <–DIV=1025
Example: set all the bits in the DIV register to 1 for maximum frequency division.

With DIV=1025, the frequency is about 16kHz.
0xb0 1 0 0 <– DIV=0 +2, 133MHz divide by
————-
0xb0 1 0 B10000000 <– DIV=2
…
0xb0 1 B00000001 B00000000} <–DIV=4
We can play with the divider and generate a range of frequencies.
The output is always equal to the reference frequency (133MHz) divided by Prescaler(if set...) * (DIV+2).
0xb0 0x3f <–write E2 register (write EEPROM)
Finally, we can write the E2 register (0x3f) to save these setting in the EEPROM. The DS1077 will now return to these settings at power-on.
A good alternative to Maxim DS1077 is the DS1085L a 3.3volt, 66MHz version available at Digikey.

And of course here's the Arduino DUE source code to drive the chip. It does program it in order to obtain a near 6MHz output (freq needed by my project).
*/

```
#include <Wire.h>                       //address: the 7-bit slave address (optional);
const int ds_address = 0xB0 >> 1;     //DS1077 default address

// MUX WORD SETTINGS - look at table 4 of datasheet(as per your needing)
#define MUX_HI_PDN1_BIT (B01000000)  //Setting to 1 both PDN! and PDN0 -->shut down control enabled with CTRL0 or CTRL1
#define MUX_HI_PDN0_BIT (B00100000)
#define MUX_HI_SEL0_BIT (B00010000)  //setting both SEL0 and PDN0 to 1 --> enable CTRL0 to operate for device's shut down
#define MUX_HI_EN0_BIT  (B00001000)  //if EN0=1 and PDN0=0 The CTRL0 pin functions as Output Enable for OUT0 pin
#define MUX_HI_0M1_BIT  (B00000000)  // set prescaler DIVISOR M Setting 0M0, 0M1 to 0 and 1 --> Set Prescaler P0 divisor M to :2
#define MUX_HI_0M0_BIT  (B00000010)
#define MUX_HI_1M1_BIT  (B00000000)  // set prescaler DIVISOR M Setting 1M1, 1M0 to 0 and 1 --> Set Prescaler P1 divisor M to :2
#define MUX_LO_1M0_BIT  (B10000000)
#define MUX_LO_DIV1_BIT (B00000000)  // This bit allows  the output of the Prescaler P1 to be routed directly to OUT1 pin (DIV1=1),
                                     // the N divider is bypassed so the programmed value of N is ignored. If DIV1=0 (default) the N divider functions normally.

#define BIT_ON (0xff)              //all bit set to 1
#define BIT_OFF (0x00)              //all bit set to 0

void setup()
{
  Wire.begin();                  //Initiate the Wire library and join the I2C bus as a master . This should normally be called only once

  //debugging
  Serial.begin(9600);
  Serial.println("\ Sending setup data to I2C device DS1077");
}


//this is the main code. It makes use of some Functions...
void loop()
{
  i2c_write(ds_address, 0x02,
          ( 0
            | (MUX_HI_PDN1_BIT & BIT_OFF)
            | (MUX_HI_PDN0_BIT & BIT_OFF)
            | (MUX_HI_SEL0_BIT & BIT_ON)
            | (MUX_HI_EN0_BIT  & BIT_ON)
            | (MUX_HI_0M1_BIT  & BIT_ON)
```

```
          | (MUX_HI_0M0_BIT  & BIT_ON)
          | (MUX_HI_1M1_BIT  & BIT_ON)
         )
         ,
         ( 0
          | (MUX_LO_1M0_BIT  & BIT_ON)
          | (MUX_LO_DIV1_BIT & BIT_OFF)
         )
        );
  delay(10);

  i2c_write(ds_address, 0x01, B0000010, B1100000);  //DIV factor=3 + 2(fisso) = 5. The GLOBAL DIVISION FACTOR is then prescaler(:2) and DIV = (:11)
  delay(10);                          //Out1 frequency=133000000/((Prescaler(P1) divisor * DIV)) =6045450Hz = 6.045MHz...actually I read on counter 6.0334MHz...pretty close!

  i2c_write(ds_address, 0x3F);                  //Write to E2 Register (EEPROM write)

  delay(2000);
}


void i2c_write(int device, byte address) {
  Wire.beginTransmission(device);      //start transmission to device
  Wire.write(address);               // send register address
  Wire.endTransmission();             //end transmission
  byte error;
  if (error == 0)                                       //DEBUGGING CODE
   {
    Serial.print("I2C device found at address 0x");
    if (address < 16)
      Serial.print("0");
    Serial.print(address, HEX);
    Serial.println("  !");
   }
   else if (error == 4)
   {
    Serial.print("Unknow error at address 0x");
    if (address < 16)
      Serial.print("0");
```

```
    Serial.println(address, HEX);
   }

}

void i2c_write(int device, byte address, byte val1) {
  Wire.beginTransmission(device);   //start transmission to device
  Wire.write(address);            // send register address
  Wire.write(val1);              // send value to write
  Wire.endTransmission();         //end transmission
  byte error;
  if (error == 0)                                        //DEBUGGING CODE
   {
    Serial.print("I2C device found at address 0x");
    if (address < 16)
      Serial.print("0");
    Serial.print(address, HEX);
    Serial.println("  !");
   }
   else if (error == 4)
   {
    Serial.print("Unknow error at address 0x");
    if (address < 16)
      Serial.print("0");
    Serial.println(address, HEX);
   }
}

void i2c_write(int device, byte address, byte val1, byte val2) {
  Wire.beginTransmission(device);  //start transmission to device
  Wire.write(address);            // send register address
  Wire.write(val1);              // send value to write
  Wire.write(val2);              // send value to write
  Wire.endTransmission();         //end transmission
  byte error;
  if (error == 0)                                        //DEBUGGING CODE
   {
    Serial.print("I2C device found at address 0x");
```

```
    if (address < 16)
      Serial.print("0");
    Serial.print(address, HEX);
    Serial.println("  !");
   }
   else if (error == 4)
   {
    Serial.print("Unknow error at address 0x");
    if (address < 16)
      Serial.print("0");
    Serial.println(address, HEX);
   }
}
```