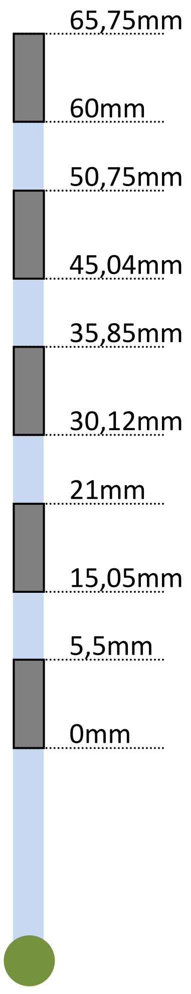
Measurement of Standard Gravity g with Arduino_Max32

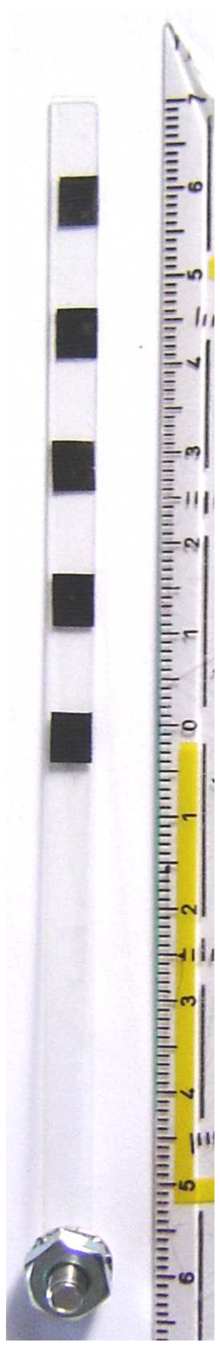
The local standard acceleration of gravity g can be measured in various ways. The swing time T of a pendulum $T = 2\pi \cdot \sqrt{\frac{\ell}{g}}$ is such a possibility. We will use g as the acceleration of a free fall together with the function $s(t) = \frac{1}{2} \cdot g \cdot t^2 + v_0 \cdot t + s_0$, with v_0 as initial speed and s_0 being the initial height. For our purpose we define the initial height as zero; so we need two equations for the rest of the unknowns g and v_0.

$$\begin{array}{l|l} & s(t_2) = \frac{1}{2} \cdot g \cdot t_2^2 + v_0 \cdot t_2 + s_0 \quad \left| \cdot t_1 \right. \\ - & s(t_1) = \frac{1}{2} \cdot g \cdot t_1^2 + v_0 \cdot t_1 + s_0 \quad \left| \cdot t_2 \right. \\ \hline \end{array} \qquad we \;\; choose \;\; s_0 := 0$$

$$s(t_2) \cdot t_1 - s(t_1) \cdot t_2 = \frac{1}{2} \cdot g \cdot (t_2^2 \cdot t_1 - t_1^2 \cdot t_2) \quad and \;\; so$$

$$g = \frac{2 \cdot \left(s(t_2) \cdot t_1 - s(t_1) \cdot t_2 \right)}{t_2^2 \cdot t_1 - t_1^2 \cdot t_2}$$





For all that we use a hybrid light barrier together with a transparent measuring strip with a piece of metal at one end for some weight. The transparent measuring strip has 5 black bars as shown.
The capture input of the Arduino_Max32 works with the transition from transparent to black; let's call that an edge to produce an interrupt.
A phototransistor switch's the capture input pin from 0V up to 3.3V.
The transparent measuring strip has 5 transparent to black transitions and 5 of black to transparent where the phototransistor switch's the capture input pin from 3.3V down to 0V. So we have 10 edges to produce an interrupt to invoke the capture module for a time measurement.

With the first edge (0mm mark) we start our time measurement:

```
WriteTimer23(0);   dt[s++]=0;
```

With the second edge (5.5mm mark) we note the time for the first black bar to pass the phototransistor: `ReadCapture1(&dt[s++]);`

With the third edge (15.05mm mark) we note the time for the first transparent bar to pass the phototransistor:

```
ReadCapture1(&dt[s++]);
```

The last measurement goes with the transition from black to transparent edge (65.75mm mark):

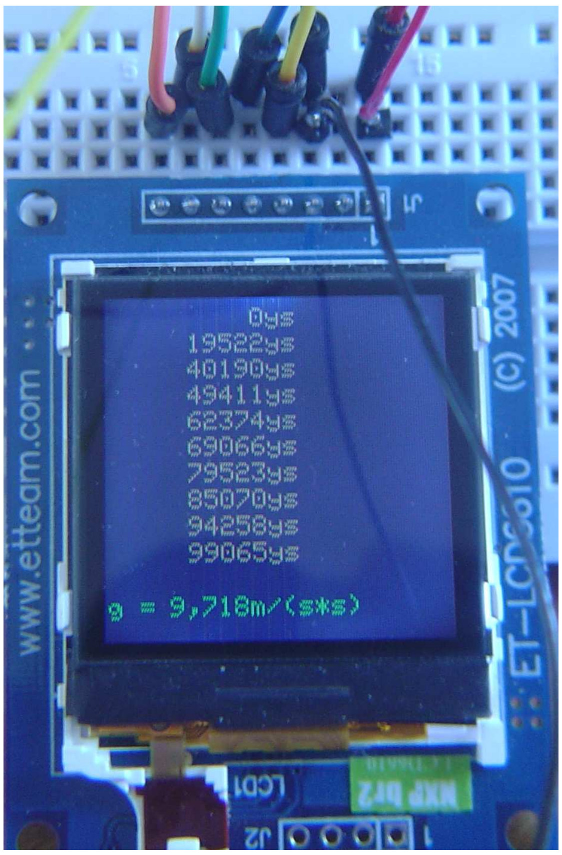
```
                    ReadCapture1(&dt[s++]);
```

Only the first time, stored in dt[0], is zero; all the other nine are not zero. The path marks goes the same way: ds[0] is the 0mm mark and is zero and all the other nine are not zero: ds[9]=0.06575 (65.75mm = 0.06575m). We measure the time in second and the length in meter.

For a g-measurement we need two not zero time marks.
For t_1 we can choose dt[1] and for t_2 dt[2] or dt[3] … or dt[9], or
for t_1 we can choose dt[2] and for t_2 dt[3] or dt[4] … or dt[9], or
…
for t_1 we can choose dt[8] and for t_2 dt[9]. You can see that we get $\sum_{k=1}^{8} k^2 = 204 =: n$ possibilities to calculate a g-value. We display the average value $g = \frac{g_1 + \cdots + g_{n=204}}{n}$ for a local gravitational acceleration on earth:

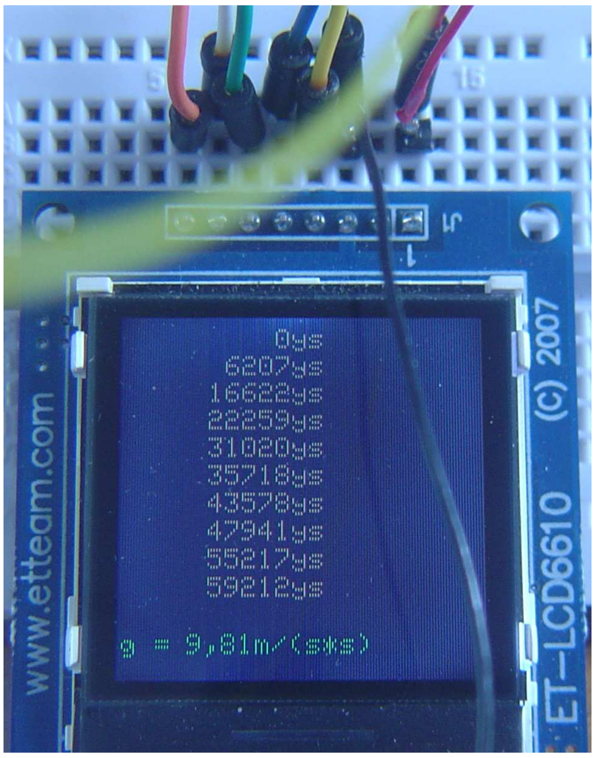
```
for(i=1;i<9;i++)
   for(j=i+1;j<10;j++)
   {
    ds1=ds[i]; ds2=ds[j];//meter
    dt1=dt[i]/f_cpu; dt2=dt[j]/f_cpu;//second because of T2_PS_1_1
    g += 2*(ds2*dt1-ds1*dt2)/(dt2*dt2*dt1-dt1*dt1*dt2);
    n++;
   }
g=g/n;//average value of g
```

and we get for example: $g = 9{,}72\frac{m}{s^2}$ as shown:



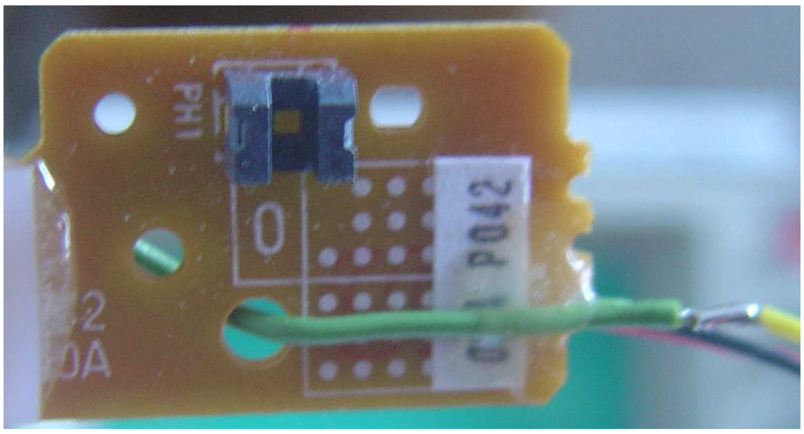
```
      0µs
  19522µs
  40190µs
  49411µs
  62374µs
  69066µs
  79523µs
  85070µs
  94258µs
  99065µs

g = 9,718m/(s*s)
```

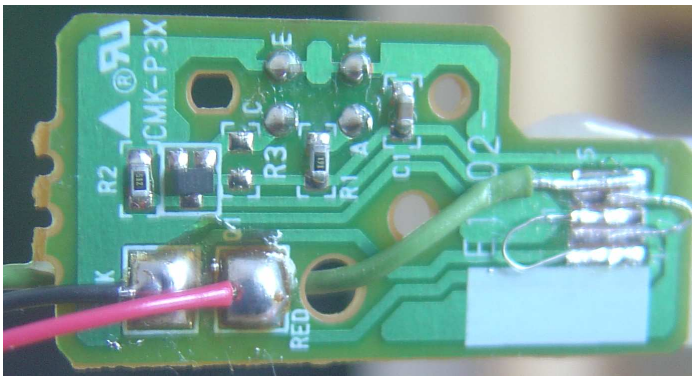
and later with more training in dropping the measuring strip: $g = 9{,}81\frac{m}{s^2}$

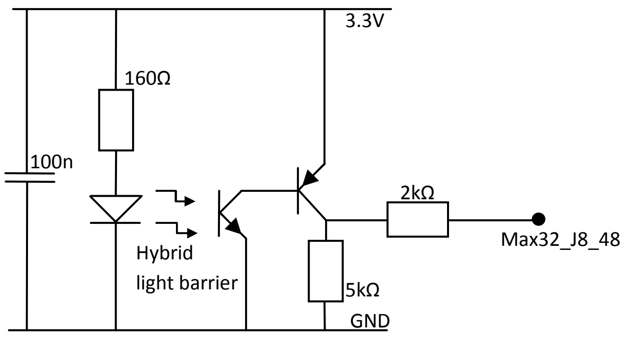


```
       0µs
    6207µs
   16622µs
   22259µs
   31020µs
   35718µs
   43578µs
   47941µs
   55217µs
   59212µs

g = 9,81m/(s*s)
```

The Hybrid light barrier

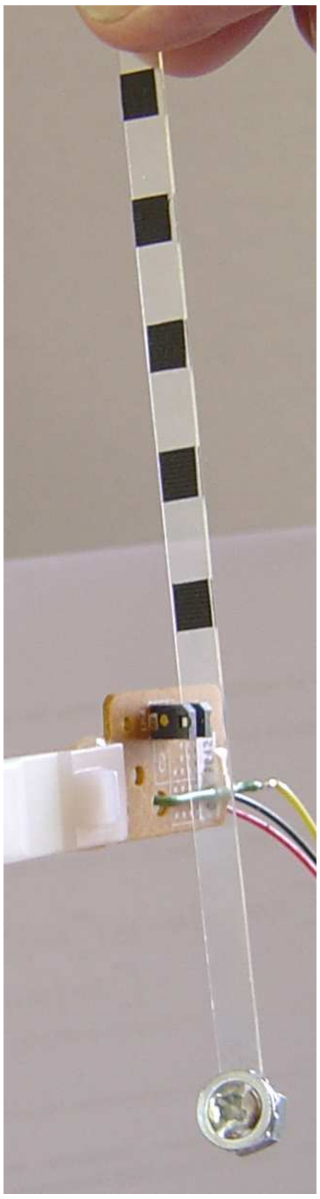






The Hybrid light barrier is part of an old bubble jet printer I disassembled.

Just drop the transparent measuring strip …



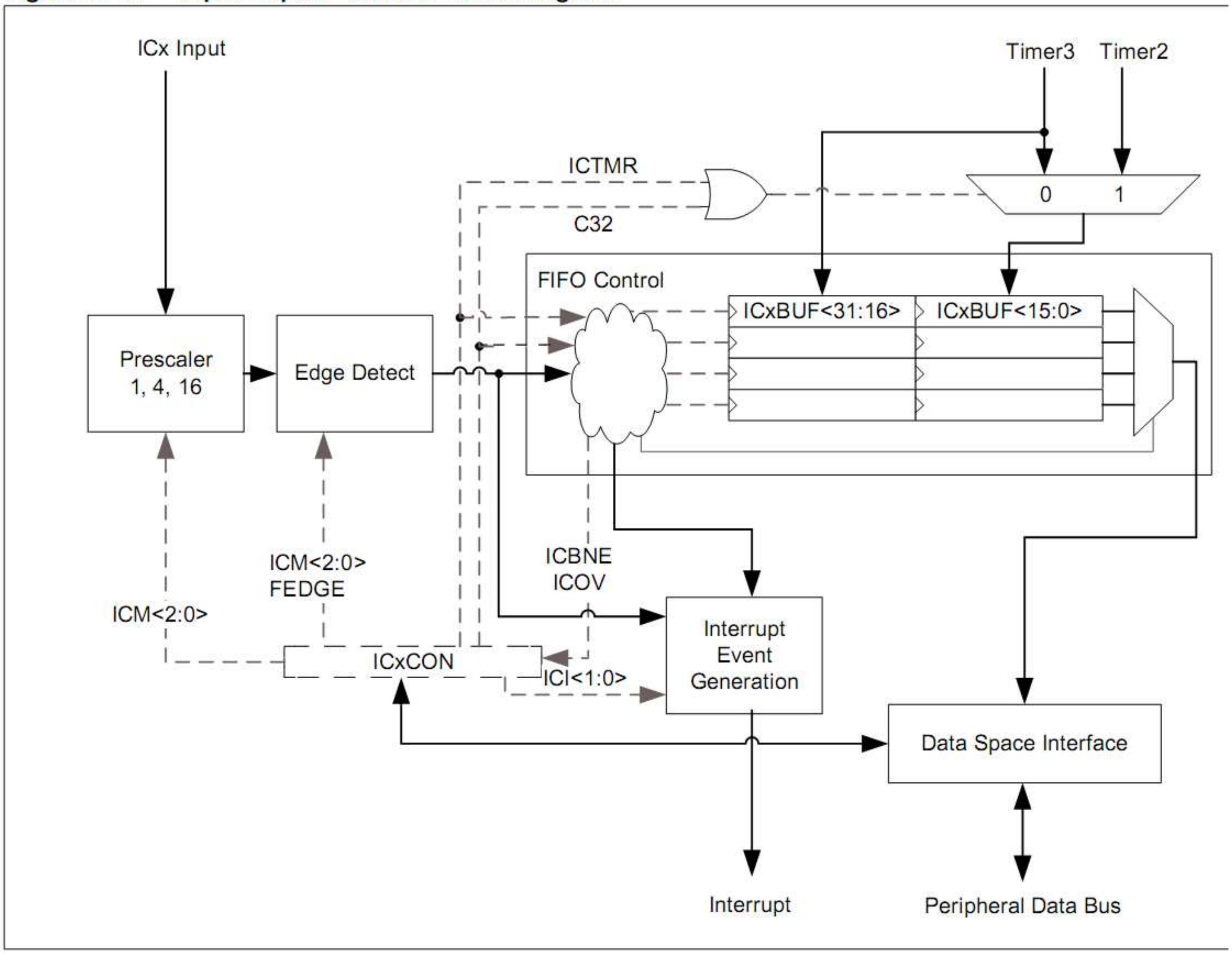
TIMER SELECTION

The Max32_PIC32 device has five Input Capture modules. Each module can select between one of two 16-bit timers for the time base or one 32-bit timer, which is formed by combining two 16-bit timers. Refer to the specific device data sheet for the timers that can be selected.
For 16-bit Capture mode, setting ICTMR (ICxCON<7>) to '0' selects Timer3 for capture. Setting ICTMR (ICxCON<7>) to '1' selects Timer2 for capture.
We configure the Input Capture module to support 32-bit capture and use a 32-bit timer resource for capture.
By setting C32 (ICxCON<8>) to '1', a 32-bit timer resource is captured. The 32-bit timer resource is routed into the module using the existing 16-bit timer inputs. Timer2 provides the lower 16 bits and Timer3 provides the upper 16 bits.
I recommend the use of the PIC32 Peripheral Libraries for all the work of programming:

```
OpenTimer23(T2_ON | T2_32BIT_MODE_ON | T2_PS_1_1, 4000000000);
```

The Capture Module

Figure 15-1: Input Capture Module Block Diagram



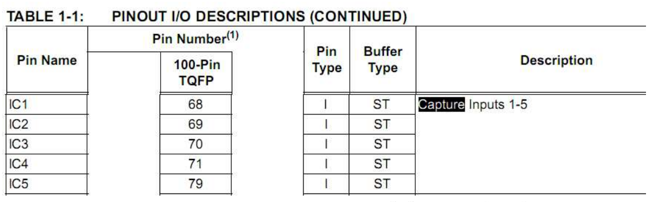


TABLE 1-1: PINOUT I/O DESCRIPTIONS (CONTINUED)

Pin Name	Pin Number[1] 100-Pin TQFP	Pin Type	Buffer Type	Description
IC1	68	I	ST	Capture Inputs 1-5
IC2	69	I	ST	
IC3	70	I	ST	
IC4	71	I	ST	
IC5	79	I	ST	

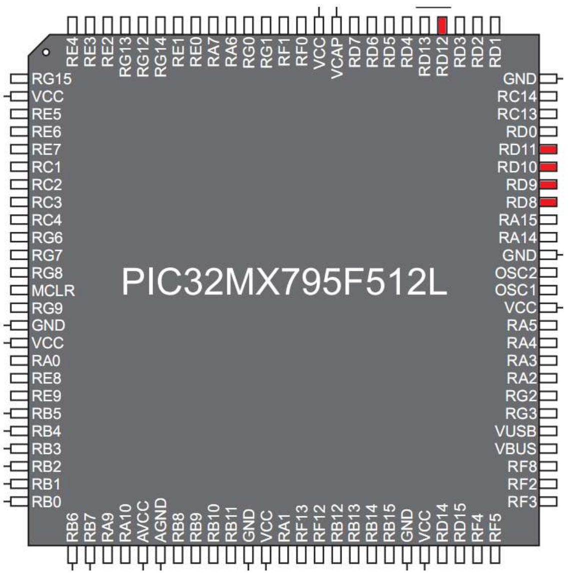
$\underbrace{\text{J3\_A8}}_{\text{Max32}}\underbrace{\text{\_RD12}}_{\text{Pic32}}$,

J8_A38_RD10,

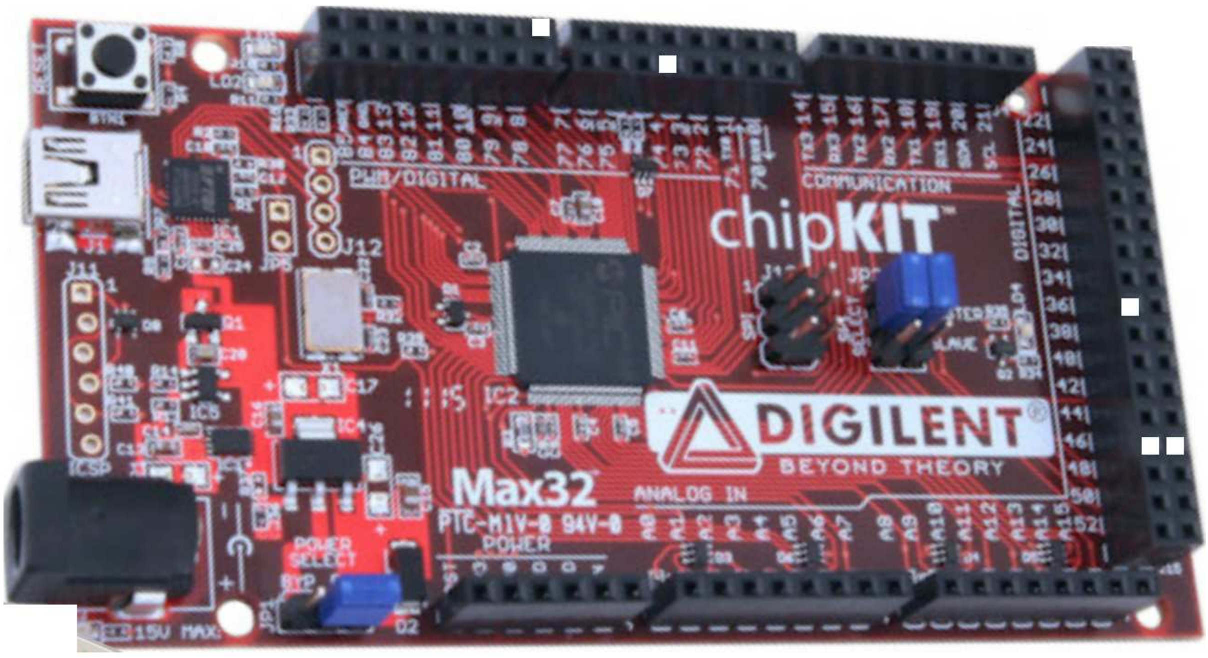
J8_A48_RD8,

J8_A49_RD11,

J14_A74_RD9



PIC32MX795F512L
RE4 RE3 RE2 RG13 RG12 RG14 RE1 RE0 RA7 RA6 RG0 RG1 RF1 RF0 VCC VCAP RD7 RD6 RD5 RD4 RD13 RD12 RD3 RD2 RD1
RG15 VCC RE5 RE6 RE7 RC1 RC2 RC3 RC4 RG6 RG7 RG8 MCLR RG9 GND VCC RA0 RE8 RE9 RB5 RB4 RB3 RB2 RB1 RB0
GND RC14 RC13 RD0 RD11 RD10 RD9 RD8 RA15 RA14 GND OSC2 OSC1 VCC RA5 RA4 RA3 RA2 RG2 RG3 VUSB VBUS RF8 RF2 RF3
RB6 RB7 RA9 RA10 AVCC AGND RB8 RB9 RB10 RB11 GND VCC RA1 RF13 RF12 RB12 RB13 RB14 RB15 GND VCC RD14 RD15 RF4 RF5



The five capture pins (white points): J3_A8_RD12, J8_A38_RD10, J8_A48_RD8, J8_A49_RD11, J14_A74_RD9

Probably you might want to test something without the big setup for any LCD-display. Here is a working sketch together with a small Python program to receive and display something:

Connect Pin Max32_J8_A48_PIC32_RD8_pin68 to GND (you do not need any resistor). If you disconnect, an interrupt occurs; we can read his occurrence with mIC1CaptureReady().

```
unsigned int CaptureTime;

void setup() {
  Serial.begin(9600);
  mIC1ClearIntFlag();
  //OpenTimer2(T2_ON | T2_SOURCE_INT | T2_PS_1_256, 0xffff);
  OpenTimer23(T2_ON | T2_32BIT_MODE_ON | T2_PS_1_256, 4000000000); //or 0xFFFFffff
  //Max32_J8_A48_PIC32_RD8_pin68
  OpenCapture1( IC_EVERY_EDGE | IC_INT_1CAPTURE | IC_CAP_32BIT | IC_TIMER2_SRC | IC_FEDGE_RISE | IC_ON );
}

void loop(){
  //wait for Capture events
  while( !mIC1CaptureReady() ) ;
  //Now Read the captured timer value
  while( mIC1CaptureReady() )
  {
    //CaptureTime = mIC1ReadCapture() & 0x0000FFFF; //necessary for 16Bit mode
    CaptureTime = mIC1ReadCapture(); //now masking out not necessary
    Serial.println(CaptureTime);
    //Serial.println(1234567890);
  }
  mIC1ClearIntFlag();
  TMR2 = 0;//if you drop this line T23 counts up until 4.000.000.000:
  //then CaptureTime ti is: 0 < t1 < t2 < t3 < ... < 4.000.000.000
}
```

… with Python:

```
#Python
import serial

ser = serial.Serial('COM3', 9600, timeout=1.05)

while True:
    msg = ser.readline()
    print(msg)
```

Using Interrupts …

❶ **Provide Service Routines**

```
extern "C"
{
void __ISR(_TIMER_1_VECTOR, ipl2) my_T1_I(void)
{
   LATEINV = 0b1;    //flicker LED1 4Hz at cpu = 80MHz
   mT1ClearIntFlag();    // clear the interrupt flag
}
};
```

❷ **Configure Interrupt Controller**

```
ConfigIntTimer1(T1_INT_ON | T1_INT_PRIOR_2);
```

❸ **Configure Peripheral**

```
OpenTimer1(T1_ON | T1_SOURCE_INT | T1_PS_1_256, 0xFFFF);
```

❹ **Important:** Make sure priorities match between Interrupt Controller and Service Routines

Calculate Time …

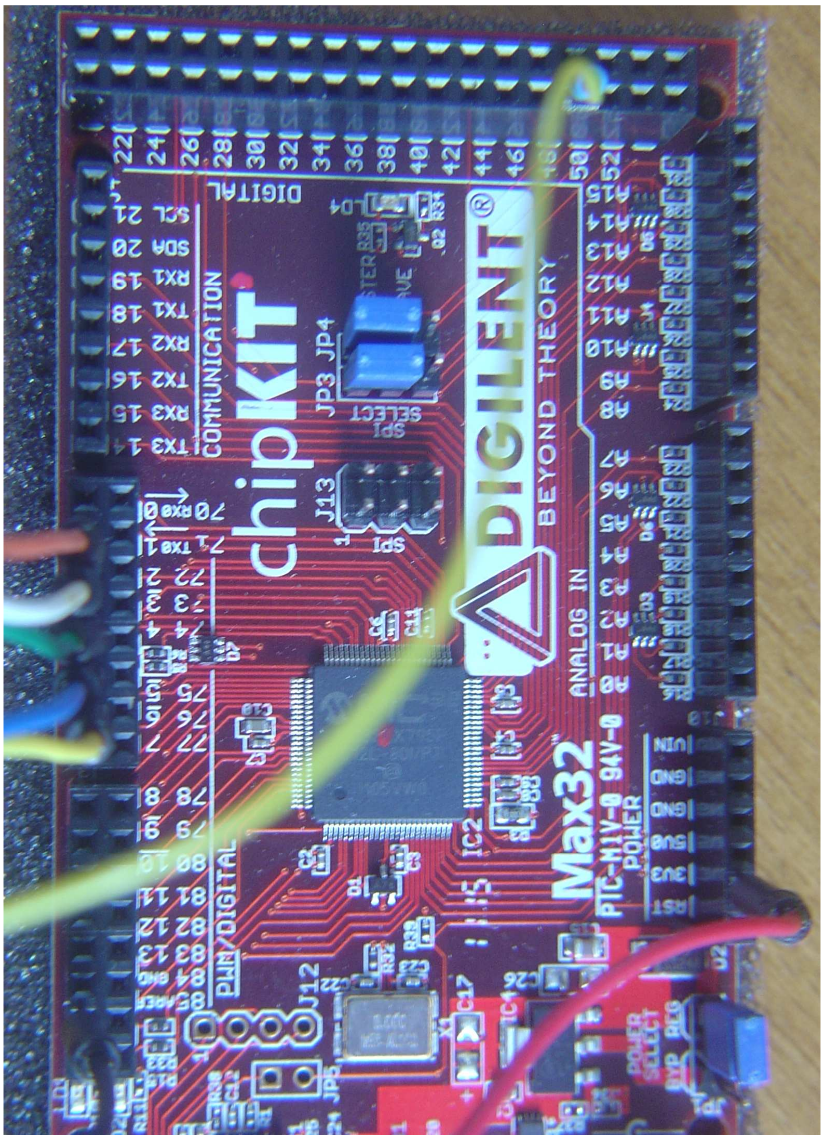
$$\Delta t = f_{Clock} \cdot presacle \cdot TMR \underbrace{=}_{example} \frac{1}{80.000.000} \cdot \underbrace{s}_{second} \cdot 256 \cdot \underbrace{TMR}_{timer\ register\ value}$$

… essential code lines for g:

```
//-------------Arduino-Main Code ---------------------
#define f_cpu 80000000. //don't forget the point
char  txt[32];
byte i,j,n,s;
float dt1, dt2, ds1, ds2, g;
float ds[]={0,0.0057,0.01506,0.0208,0.03012,0.03587,0.0448,0.05075,0.06,0.0656};//meter
unsigned int dt[10];

void setup()
{
 OpenTimer23(T2_ON | T2_32BIT_MODE_ON | T2_PS_1_1, 4000000000);
 ConfigIntCapture1(IC_INT_ON | IC_INT_PRIOR_6 | IC_INT_SUB_PRIOR_3);
 OpenCapture1(IC_INT_1CAPTURE | IC_CAP_32BIT | IC_TIMER2_SRC | IC_EVERY_EDGE | IC_ON );
//IC_EVERY_EDGE is IC1CONbits.ICM == 0b001
 INTEnableSystemMultiVectoredInt();
 //-------------------------------------------------------------------------------------------
 init_max32_for_LCD6610();
 init_LCD6610();
 lcd_clear(BLACK,0,0,131,131);
 LCDPutStr((char*)"  g-measurement ", 5,2,LARGE,ORANGE,BLACK);
 LCDPutStr((char*)"      with      ", 25,2,LARGE,ORANGE,BLACK);
 LCDPutStr((char*)"ArduinoPIC_Max32", 45,2,LARGE,ORANGE,BLACK);
 //----------------------------------
 s = 0;
}

void loop()
{
 if(s == 10){
  lcd_clear(BLACK,0,0,131,131);
  for(i=0;i<10;i++){
   r_utoa((dt[i]/80.), txt);
   txt[10]='y';txt[11]='s';txt[12]=0;
   LCDPutStr(txt,5+i*10,2,SMALL,ORANGE,BLACK);
  }
  //----------------begin of g----------------
  g=0; n=0;
  for(i=1;i<9;i++)//play with i=1; ... i=8;
   for(j=i+1;j<10;j++)
   {
    ds1=ds[i]; ds2=ds[j];//meter
    dt1=dt[i]/f_cpu; dt2=dt[j]/f_cpu;//second because of T2_PS_1_1
    g += 2*(ds2*dt1-ds1*dt2)/(dt2*dt2*dt1-dt1*dt1*dt2);
    n++;
   }
  g=g/n;//average value of g
  my_ftoa(g,txt);
```

```
    LCDPutStr((char*)"g = ", 115,2,SMALL,GREEN,BLACK);
    strcat(txt, "m/(s*s)");
    LCDPutStr(txt,115,25,SMALL,GREEN,BLACK);
    //----------------end of g--------------------
    s = 0;
    OpenTimer23(T2_ON | T2_32BIT_MODE_ON | T2_PS_1_1, 4000000000);
    ConfigIntCapture1(IC_INT_ON | IC_INT_PRIOR_6 | IC_INT_SUB_PRIOR_3);
    OpenCapture1(IC_INT_1CAPTURE | IC_CAP_32BIT | IC_TIMER2_SRC | IC_EVERY_EDGE | IC_ON );
  }
}

extern "C"
{
  void __ISR(_INPUT_CAPTURE_1_VECTOR,ipl6) my_capture1(void)
  {
     switch(s)
     {
       case 0://start
         WriteTimer23(0);
         dt[s++]=0;
         break;
       case 10://end
         CloseCapture1();
         CloseTimer23();
         break;
       default:
         ReadCapture1(&dt[s++]);
     }
    if ( mIC1CaptureReady() ) //1. to do
      mIC1ReadCapture();
    mIC1ClearIntFlag(); //2. to do
  }
};
```

The wiring with the board …



The whole program …

```
#include <NXP_FONT.h>

#define myBL 2          // Digital 2 --> BL       pinMode(BL, OUTPUT);
#define myCS 3          // Digital 3 --> CS       pinMode(CS, OUTPUT);
#define myCLK 4         // Digital 4 --> SCLK     pinMode(CLK, OUTPUT);
#define mySDA 5         // Digital 5 --> SDATA    pinMode(SDA, OUTPUT);
#define myRESET 6       // Digital 6 --> RESET    pinMode(RESET, OUTPUT);

//Philips(NXP):PCF8833 Header */
#define NOP 0x00  // nop
#define SWRESET  0x01  // software reset
#define BSTROFF  0x02  // booster voltage OFF
#define BSTRON   0x03  // booster voltage ON
#define RDDIDIF  0x04  // read display identification
#define RDDST    0x09  // read display status
#define SLEEPIN  0x10  // sleep in
#define SLEEPOUT 0x11  // sleep out
#define PTLON    0x12  // partial display mode
#define NORON    0x13  // display normal mode
#define INVOFF   0x20  // inversion OFF
#define INVON    0x21  // inversion ON
#define DALO     0x22  // all pixel OFF
#define DAL      0x23  // all pixel ON
#define SETCON   0x25  // write contrast
#define DISPOFF  0x28  // display OFF
#define DISPON   0x29  // display ON
#define CASET    0x2A  // column address set
#define PASET    0x2B  // page address set
#define RAMWR    0x2C  // memory write
#define RGBSET   0x2D  // colour set
#define PTLAR    0x30  // partial area
#define VSCRDEF  0x33  // vertical scrolling definition
#define TEOFF    0x34  // test mode
#define TEON     0x35  // test mode
#define MADCTL   0x36  // memory access control
#define SEP      0x37  // vertical scrolling start address
#define IDMOFF   0x38  // idle mode OFF
#define IDMON    0x39  // idle mode ON
#define COLMOD   0x3A  // interface pixel format
#define SETVOP   0xB0  // set Vop
#define BRS      0xB4  // bottom row swap
#define TRS      0xB6  // top row swap
#define DISCTR   0xB9  // display control
//#define DAOR   0xBA  // data order(DOR)
#define TCDFE    0xBD  // enable/disable DF temperature compensation
#define TCVOPE   0xBF  // enable/disable Vop temp comp
#define EC       0xC0  // internal or external oscillator
#define SETMUL   0xC2  // set multiplication factor
#define TCVOPAB  0xC3  // set TCVOP slopes A and B
#define TCVOPCD  0xC4  // set TCVOP slopes c and d
#define TCDF     0xC5  // set divider frequency
#define DF8COLOR 0xC6  // set divider frequency 8-color mode
#define SETBS    0xC7  // set bias system
#define RDTEMP   0xC8  // temperature read back
#define NLI      0xC9  // n-line inversion
#define RDID1    0xDA  // read ID1
#define RDID2    0xDB  // read ID2
#define RDID3    0xDC  // read ID3

// Font sizes
```

```
#define SMALL 0
#define MEDIUM 1
#define LARGE 2

// Booleans
#define NOFILL 0
#define FILL 1

// 12-bit color definitions
#define WHITE 0xFFF
#define BLACK 0x000
#define RED 0xF00
#define GREEN 0x0F0
#define BLUE 0x00F
#define CYAN 0x0FF
#define MAGENTA 0xF0F
#define YELLOW 0xFF0
#define BROWN 0xB22
#define ORANGE 0xFA0
#define PINK 0xF6A

#define CS_0     digitalWrite(myCS, LOW);
#define CS_1     digitalWrite(myCS, HIGH);
#define CLK_0    digitalWrite(myCLK, LOW);
#define CLK_1    digitalWrite(myCLK, HIGH);
#define SDA_0    digitalWrite(mySDA, LOW);
#define SDA_1    digitalWrite(mySDA, HIGH);
#define RESET_0  digitalWrite(myRESET, LOW);
#define RESET_1  digitalWrite(myRESET, HIGH);
#define BL_0     digitalWrite(myBL, LOW);
#define BL_1     digitalWrite(myBL, HIGH);
/* End of Define Philips(NXP):PCF8833 Header */

PROGMEM const unsigned char FONT6x8[] = F_6x8;
PROGMEM const unsigned char FONT8x8[]  = F_8x8;
PROGMEM const unsigned char FONT8x16[] = F_8x16;

//----------------Function prototypes----------------------

void sendCMD(byte);
void sendData(byte);
void shiftBits(byte);
void lcd_init();
void draw_color_bar();
void lcd_clear(uint16_t, byte, byte, byte, byte);
void LCDPutStr(char*, int, int, int, int, int);
void LCDPutChar(char, int, int, int, int, int);
void LCDSetLine(int, int, int, int, int);
void LCDSetRect(int, int, int, int, unsigned char fill, int);
void LCDSetCircle(int, int, int, int);
void LCDSetPixel(byte, byte, int);
void LCDSetXY(byte, byte);

char *my_ftoa(float val, char *str)
{
  //static char buffer[10];
  char *cp; cp=str;
  int v, v0, rest, rest0;
  char c;
  v0 = (int)val; v=v0;
  //rest0=(int)((val-(int)val)*10000000); rest = rest0;
  rest0=(int)((val-(int)val)*1000); rest = rest0;
  do {
    v /= 10;
    cp++;
  } while(v != 0);
  do {
    rest /= 10;
    cp++;
  } while(rest != 0);
  cp++; //wegen ','
  *cp-- = 0;
  do {
    c = rest0 % 10;
    rest0 /= 10;
    c += '0';
    *cp-- = c;
  } while(rest0 != 0);
  *cp-- = ',';
  do {
    c = v0 % 10;
    v0 /= 10;
    c += '0';
    *cp-- = c;
  } while(v0 != 0);
  return  cp;
}

char* my_utoa(unsigned val, char *buffer)
{
  //static char buffer[10];
  char * cp = buffer;
  unsigned v;
  char   c;
  v = val;
  do {
    v /= 10;
    cp++;
  } while(v != 0);
  *cp-- = 0;
  do {
    c = val % 10;
    val /= 10;
    c += '0';
    *cp-- = c;
  } while(val != 0);
  return  buffer;
}

char* r_utoa(unsigned val, char *buffer) //right
{
  //static char buffer[10];
  char * cp = buffer + 10; //also 11 byte
  unsigned v;
  char   c;
  *buffer=' '; //case0: cp=buffer
  v = val;
  *cp-- = 0;
  do {
    c = val % 10;
    val /= 10;
    c += '0';
    *cp-- = c;
  } while(val != 0);
  while(cp>buffer) *cp--=' ';//case0: "cp=buffer" remains to handle
  return  buffer;
}
```

```
//-------------Arduino-Main Code---------------------
#define f_cpu 80000000. //don't forget the point
char  txt[32];
byte i,j,n,s;
float dt1, dt2, ds1, ds2, g;
float
ds[]={0,0.0057,0.01506,0.0208,0.03012,0.03587,0.0448,0.05075,0.06,0.0656};//meter
unsigned int dt[10];

void setup()
{
  OpenTimer23(T2_ON | T2_32BIT_MODE_ON | T2_PS_1_1, 4000000000);
  ConfigIntCapture1(IC_INT_ON | IC_INT_PRIOR_6 | IC_INT_SUB_PRIOR_3);
  OpenCapture1(IC_INT_1CAPTURE | IC_CAP_32BIT | IC_TIMER2_SRC | IC_EVERY_EDGE |
IC_ON );  //IC_EVERY_EDGE is IC1CONbits.ICM == 0b001
  INTEnableSystemMultiVectoredInt();
  //------------------------------------------------------------------------------
-----------------
  init_max32_for_LCD6610();
  init_LCD6610();
  lcd_clear(BLACK,0,0,131,131);
  LCDPutStr((char*)"  g-measurement ", 5,2,LARGE,ORANGE,BLACK);
  LCDPutStr((char*)"      with      ", 25,2,LARGE,ORANGE,BLACK);
  LCDPutStr((char*)"ArduinoPIC_Max32", 45,2,LARGE,ORANGE,BLACK);
  //-----------------------------------
  s = 0;
}

void loop()
{
  if(s == 10){
    lcd_clear(BLACK,0,0,131,131);
    for(i=0;i<10;i++){
      r_utoa((dt[i]/80.), txt);
      txt[10]='y';txt[11]='s';txt[12]=0;
      LCDPutStr(txt,5+i*10,2,SMALL,ORANGE,BLACK);
    }
    //-----------------begin of g----------------
    g=0; n=0;
    for(i=1;i<9;i++)//play with i=1; ... i=8;
      for(j=i+1;j<10;j++)
      {
        ds1=ds[i]; ds2=ds[j];//meter
        dt1=dt[i]/f_cpu; dt2=dt[j]/f_cpu;//second because of T2_PS_1_1
        g += 2*(ds2*dt1-ds1*dt2)/(dt2*dt2*dt1-dt1*dt1*dt2);
        n++;
      }
    g=g/n;//average value of g
    my_ftoa(g,txt);
    LCDPutStr((char*)"g = ", 115,2,SMALL,GREEN,BLACK);
    strcat(txt, "m/(s*s)");
    LCDPutStr(txt,115,25,SMALL,GREEN,BLACK);
    //----------------end of g--------------------
    s = 0;
    OpenTimer23(T2_ON | T2_32BIT_MODE_ON | T2_PS_1_1, 4000000000);
    ConfigIntCapture1(IC_INT_ON | IC_INT_PRIOR_6 | IC_INT_SUB_PRIOR_3);
    OpenCapture1(IC_INT_1CAPTURE | IC_CAP_32BIT | IC_TIMER2_SRC | IC_EVERY_EDGE |
IC_ON );
  }
}

extern "C"
{
  void __ISR(_INPUT_CAPTURE_1_VECTOR,ipl6) my_capture1(void)
  {
     switch(s)
     {
       case 0://start
         WriteTimer23(0);
         dt[s++]=0;
         break;
       case 10://end
         CloseCapture1();
         CloseTimer23();
         break;
       default:
         ReadCapture1(&dt[s++]);
    }
    if ( mIC1CaptureReady() ) //1.
      mIC1ReadCapture();
    mIC1ClearIntFlag(); //2.
  }
};


void sendCMD(byte data)
{
  CS_1
  CLK_0
  CS_0
  SDA_0
  CLK_1
  CLK_0

  shiftBits(data);
  CLK_0
  CS_1
}

void sendData(byte data) {

  CS_1
  CLK_0
  CS_0
  SDA_1
  CLK_1
  CLK_0

  shiftBits(data);
  CLK_0
  CS_1
}

void shiftBits(byte data)
{
  byte Bit;

  for (Bit = 0; Bit < 8; Bit++)     // 8 Bit Write
  {
    CLK_0          // Standby SCLK
    if((data&0x80)>>7)
    {
      SDA_1
    }
    else
    {
      SDA_0
    }
    CLK_1          // Strobe signal bit
    data <<= 1;   // Next bit data
  }
}

void init_max32_for_LCD6610()
{
  pinMode(myBL, OUTPUT);
  pinMode(myCS, OUTPUT);
  pinMode(myCLK, OUTPUT);
  pinMode(mySDA, OUTPUT);
  pinMode(myRESET, OUTPUT);
  digitalWrite(myBL, HIGH);
  digitalWrite(myCS, HIGH);
  digitalWrite(myCLK, HIGH);
  digitalWrite(mySDA, HIGH);
  digitalWrite(myRESET, HIGH);
}

void init_LCD6610()
{
  // Initial state
  CLK_0
  CS_1
  SDA_1

  // Hardware Reset LCD
  RESET_0
  delay(100);
  RESET_1
  delay(100);

  // Sleep out (commmand 0x11)
  sendCMD(SLEEPOUT);

  // Inversion on (command 0x20)
  //sendCMD(INVON);    // seems to be required for this controller
  sendCMD(INVOFF);

  // Color Interface Pixel Format (command 0x3A)
  sendCMD(COLMOD);
  sendData(0x03);    // 0x03 = 12 bits-per-pixel

  // Memory access controler (command 0x36)
  sendCMD(MADCTL);
  sendData(0xC8); // 0xC0 = mirror x and y, reverse rgb

  // Write contrast (command 0x25)
  sendCMD(SETCON);
  sendData(63); // contrast
  delay(1000);

  // Display On (command 0x29)
  sendCMD(DISPON);
  delay(500);
}

void draw_color_bar()
{
  lcd_clear(RED,0,0,131,33);
  lcd_clear(GREEN,0,34,131,66);
  lcd_clear(BLUE,0,67,131,99);
  lcd_clear(WHITE,0,100,131,131);
}
```

```
void lcd_clear(uint16_t color, byte x0, byte y0, byte x1, byte y1)
{
  uint16_t xmin, xmax, ymin, ymax;
  uint16_t i;

  // best way to create a filled rectangle is to define a drawing box
  // and loop two pixels at a time
  // calculate the min and max for x and y directions
  xmin = (x0 <= x1) ? x0 : x1;
  xmax = (x0 > x1) ? x0 : x1;
  ymin = (y0 <= y1) ? y0 : y1;
  ymax = (y0 > y1) ? y0 : y1;

  // specify the controller drawing box according to those limits
  // Row address set (command 0x2B)
  sendCMD(PASET);
  sendData(xmin);
  sendData(xmax);

  // Column address set (command 0x2A)
  sendCMD(CASET);
  sendData(ymin);
  sendData(ymax);

  // WRITE MEMORY
  sendCMD(RAMWR);

  // loop on total number of pixels / 2
  for (i = 0; i < ((((xmax - xmin + 1) * (ymax - ymin + 1)) / 2) + 1); i++)
  {
    // use the color value to output three data bytes covering two pixels
    // For some reason, it has to send blue first then green and red
    sendData((color << 4) | ((color & 0xF0) >> 4));
    sendData(((color >> 4) & 0xF0) | (color & 0x0F));
    sendData((color & 0xF0) | (color >> 8));
  }
}

void LCDPutStr(char *pString, int x, int y, int Size, int fColor, int bColor)
{
  // loop until null-terminator is seen
  while (*pString != 0x00)
  {
    // draw the character
    LCDPutChar(*pString++, x, y, Size, fColor, bColor);

    // advance the y position
    if (Size == SMALL)
    y = y + 6;

    else if (Size == MEDIUM)
    y = y + 8;

    else
    y = y + 8;

    // bail out if y exceeds 131
    if (y > 131) break;
  }
}

void LCDPutChar(char c, int x, int y, int size, int fColor, int bColor)
{
```

```
int i,j;
unsigned int  nCols;
unsigned int  nRows;
unsigned int  nBytes;
unsigned char PixelRow;
unsigned char Mask;
unsigned int  Word0;
unsigned int  Word1;
unsigned char *pFont;
unsigned char *pChar;
unsigned char *FontTable[] = {(unsigned char *)FONT6x8,
                              (unsigned char *)FONT8x8,
                              (unsigned char *)FONT8x16};

// get pointer to the beginning of the selected font table
pFont = (unsigned char *)FontTable[size];

// get the nColumns, nRows and nBytes
nCols = *pFont;
nRows = *(pFont + 1);
nBytes = *(pFont + 2);
// get pointer to the last byte of the desired character
pChar = pFont + (nBytes * (c - 0x1F));
// Row address set (command 0x2B)
sendCMD(PASET);
sendData(x);
sendData(x + nRows - 1);
// Column address set (command 0x2A)
sendCMD(CASET);
sendData(y);
sendData(y + nCols - 1);
// WRITE MEMORY
sendCMD(RAMWR);
// loop on each row, working backwards from the bottom to the top
for (i = nRows - 1; i >= 0; i--)
{
  // copy pixel row from font table and then decrement row
  PixelRow = *pChar++;
  // loop on each pixel in the row (left to right)
  // Note: we do two pixels each loop
  Mask = 0x80;
  for (j = 0; j < nCols; j += 2)
 {
    // if pixel bit set, use foreground color; else use the background color
    // now get the pixel color for two successive pixels
    if ((PixelRow & Mask) == 0)
      Word0 = bColor;
    else
      Word0 = fColor;
    Mask = Mask >> 1;

    if ((PixelRow & Mask) == 0)
      Word1 = bColor;
    else
      Word1 = fColor;
    Mask = Mask >> 1;

    // use this information to output three data bytes
    // For some reason, it has to send blue first then green and red
    sendData((Word0 << 4) | ((Word0 & 0xF0) >> 4));
    sendData(((Word0 >> 4) & 0xF0) | (Word1 & 0x0F));
    sendData((Word1 & 0xF0) | (Word1 >> 8));
  }
}
```

```
  // terminate the Write Memory command
  sendCMD(NOP);
}

void LCDSetLine(int x0, int y0, int x1, int y1, int color)
{
  int dy = y1 - y0;
  int dx = x1 - x0;
  int stepx, stepy;
  if (dy < 0) { dy = -dy; stepy = -1; } else { stepy = 1; }
  if (dx < 0) { dx = -dx; stepx = -1; } else { stepx = 1; }
  dy <<= 1; // dy is now 2*dy
  dx <<= 1; // dx is now 2*dx
  LCDSetPixel(x0, y0, color);
  if (dx > dy)
  {
    int fraction = dy - (dx >> 1); // same as 2*dy - dx
    while (x0 != x1)
    {
      if (fraction >= 0)
      {
        y0 += stepy;
        fraction -= dx; // same as fraction -= 2*dx
      }
      x0 += stepx;
      fraction += dy; // same as fraction -= 2*dy
      LCDSetPixel(x0, y0, color);
    }
  }
  else
  {
    int fraction = dx - (dy >> 1);
    while (y0 != y1)
   {
      if (fraction >= 0)
    {
        x0 += stepx;
        fraction -= dy;
      }
      y0 += stepy;
      fraction += dx;
      LCDSetPixel(x0, y0, color);
    }
  }
}

void LCDSetRect(int x0, int y0, int x1, int y1, unsigned char fill, int color)
{
  int xmin, xmax, ymin, ymax;
  int i;

  // check if the rectangle is to be filled
  if (fill == FILL)
  {
    // best way to create a filled rectangle is to define a drawing box
    // and loop two pixels at a time
    // calculate the min and max for x and y directions
    xmin = (x0 <= x1) ? x0 : x1;
    xmax = (x0 > x1) ? x0 : x1;
    ymin = (y0 <= y1) ? y0 : y1;
    ymax = (y0 > y1) ? y0 : y1;

    // specify the controller drawing box according to those limits
    // Row address set (command 0x2B)
    sendCMD(PASET);
    sendData(xmin);
    sendData(xmax);

    // Column address set (command 0x2A)
    sendCMD(CASET);
    sendData(ymin);
    sendData(ymax);

    // WRITE MEMORY
    sendCMD(RAMWR);

    // loop on total number of pixels / 2
    for (i = 0; i < ((((xmax - xmin + 1) * (ymax - ymin + 1)) / 2) + 1); i++)
    {
      // use the color value to output three data bytes covering two pixels
      // For some reason, it has to send blue first then green and red
      sendData((color << 4) | ((color & 0xF0) >> 4));
      sendData(((color >> 4) & 0xF0) | (color & 0x0F));
      sendData((color & 0xF0) | (color >> 8));
    }
  }
  else
  {
    // best way to draw un unfilled rectangle is to draw four lines
    LCDSetLine(x0, y0, x1, y0, color);
    LCDSetLine(x0, y1, x1, y1, color);
    LCDSetLine(x0, y0, x0, y1, color);
    LCDSetLine(x1, y0, x1, y1, color);
  }
}

void LCDSetCircle(int x0, int y0, int radius, int color)
{
  int f = 1 - radius;
  int ddF_x = 0;
  int ddF_y = -2 * radius;
  int x = 0;
  int y = radius;
  LCDSetPixel(x0, y0 + radius, color);
  LCDSetPixel(x0, y0 - radius, color);
  LCDSetPixel(x0 + radius, y0, color);
  LCDSetPixel(x0 - radius, y0, color);
  while (x < y)
  {
    if (f >= 0)
   {
      y--;
      ddF_y += 2;
      f += ddF_y;
    }
    x++;
    ddF_x += 2;
    f += ddF_x + 1;
    LCDSetPixel(x0 + x, y0 + y, color);
    LCDSetPixel(x0 - x, y0 + y, color);
    LCDSetPixel(x0 + x, y0 - y, color);
    LCDSetPixel(x0 - x, y0 - y, color);
    LCDSetPixel(x0 + y, y0 + x, color);
    LCDSetPixel(x0 - y, y0 + x, color);
    LCDSetPixel(x0 + y, y0 - x, color);
    LCDSetPixel(x0 - y, y0 - x, color);
   }
```

```
}

void LCDSetPixel(byte x, byte y, int color)
{
  LCDSetXY(x, y);
  sendCMD(RAMWR);
  // For some reason, it has to send blue first then green and red
  sendData((color << 4) | ((color & 0xF0) >> 4));
  sendData(((color >> 4) & 0xF0));
  sendCMD(NOP);
}

void LCDSetXY(byte x, byte y)
{
  // Row address set (command 0x2B)
  sendCMD(PASET);
  sendData(x);
  sendData(x);

  // Column address set (command 0x2A)
  sendCMD(CASET);
  sendData(y);
  sendData(y);
}
```

Content of C:\heute\July_2011\Arduino\mpide\hardware\pic32\libraries\myLCD\ NXP_FONT.h

```
#define F_6x8
{0x06,0x08,0x08,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x20,0x20,0x20,0x20,0x20,0x00,0x20,0x00,0x50,0x50,0x50,0x00,0x00,0x00,0x00,0x00,0x50,0x50,0xF8,0x50,0xF8,0x50,0x50,0x00,0x20,0x78,0xA0,0x70,0x28,0xF0,0x20,0
x00,0xC0,0xC8,0x10,0x20,0x40,0x98,0x18,0x00,0x40,0xA0,0xA0,0x40,0xA8,0x90,0x68,0x00,0x30,0x30,0x20,0x40,0x00,0x00,0x00,0x00,0x10,0x20,0x40,0x40,0x40,0x20,0x10,0x00,0x40,0x20,0x10,0x10,0x10,0x20,0x40,0x00,0x00,0x20,0xA8,0x70,0x70,0xA8,0x
20,0x00,0x00,0x20,0x20,0xF8,0x20,0x20,0x00,0x00,0x00,0x00,0x00,0x00,0x30,0x30,0x20,0x40,0x00,0x00,0x00,0xF8,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x30,0x30,0x00,0x00,0x08,0x10,0x20,0x40,0x80,0x00,0x00,0x70,0x88,0x88,0xA8,0x88,0x88
,0x70,0x00,0x20,0x60,0x20,0x20,0x20,0x20,0x70,0x00,0x70,0x88,0x08,0x70,0x80,0x80,0xF8,0x00,0xF8,0x08,0x10,0x30,0x08,0x88,0x70,0x00,0x10,0x30,0x50,0x90,0xF8,0x10,0x10,0x00,0xF8,0x80,0xF0,0x08,0x08,0x88,0x70,0x00,0x38,0x40,0x80,0xF0,0x88,0x
88,0x70,0x00,0xF8,0x08,0x08,0x10,0x20,0x40,0x80,0x00,0x70,0x88,0x88,0x70,0x88,0x88,0x70,0x00,0x70,0x88,0x88,0x78,0x08,0x10,0xE0,0x00,0x00,0x00,0x20,0x00,0x20,0x00,0x00,0x00,0x00,0x00,0x20,0x00,0x20,0x20,0x40,0x00,0x08,0x10,0x20,0x40,0x20
,0x10,0x08,0x00,0x00,0x00,0xF8,0x00,0xF8,0x00,0x00,0x00,0x40,0x20,0x10,0x08,0x10,0x20,0x40,0x00,0x70,0x88,0x08,0x30,0x20,0x00,0x20,0x00,0x70,0x88,0xA8,0xB8,0xB0,0x80,0x78,0x00,0x20,0x50,0x88,0x88,0xF8,0x88,0x88,0x00,0xF0,0x88,0x88,0xF0,0x
88,0x88,0xF0,0x00,0x70,0x88,0x80,0x80,0x80,0x88,0x70,0x00,0xF0,0x88,0x88,0x88,0x88,0x88,0xF0,0x00,0xF8,0x80,0x80,0xF0,0x80,0x80,0xF8,0x00,0xF8,0x80,0x80,0xF0,0x80,0x80,0x80,0x00,0x78,0x88,0x80,0x80,0x98,0x88,0x78,0x00,0x88,0x88,0x88,0xF8,
0x88,0x88,0x88,0x00,0x70,0x20,0x20,0x20,0x20,0x20,0x70,0x00,0x38,0x10,0x10,0x10,0x10,0x90,0x60,0x00,0x88,0x90,0xA0,0xC0,0xA0,0x90,0x88,0x00,0x80,0x80,0x80,0x80,0x80,0x80,0xF8,0x00,0x88,0xD8,0xA8,0xA8,0xA8,0x88,0x88,0x00,0x88,0x88,0xC8,0
xA8,0x98,0x88,0x88,0x00,0x70,0x88,0x88,0x88,0x88,0x88,0x70,0x00,0xF0,0x88,0x88,0xF0,0x80,0x80,0x80,0x00,0x70,0x88,0x88,0x88,0xA8,0x90,0x68,0x00,0xF0,0x88,0x88,0xF0,0xA0,0x90,0x88,0x00,0x70,0x88,0x80,0x70,0x08,0x88,0x70,0x00,0xF8,0xA8,0x2
0,0x20,0x20,0x20,0x20,0x00,0x88,0x88,0x88,0x88,0x88,0x88,0x70,0x00,0x88,0x88,0x88,0x88,0x88,0x50,0x20,0x00,0x88,0x88,0x88,0xA8,0xA8,0xA8,0x50,0x00,0x88,0x88,0x50,0x20,0x50,0x88,0x88,0x00,0x88,0x88,0x50,0x20,0x20,0x20,0x20,0x00,0xF8,0x08,
0x10,0x70,0x40,0x80,0xF8,0x00,0x78,0x40,0x40,0x40,0x40,0x40,0x78,0x00,0x00,0x80,0x40,0x20,0x10,0x08,0x00,0x00,0x78,0x08,0x08,0x08,0x08,0x08,0x78,0x00,0x20,0x50,0x88,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xF8,0x00,0x60,0x
60,0x20,0x10,0x00,0x00,0x00,0x00,0x00,0x00,0x60,0x10,0x70,0x90,0x78,0x00,0x80,0x80,0xB0,0xC8,0x88,0xC8,0xB0,0x00,0x00,0x00,0x70,0x88,0x80,0x88,0x70,0x00,0x08,0x08,0x68,0x98,0x88,0x98,0x68,0x00,0x00,0x00,0x70,0x88,0xF8,0x80,0x70,0x00,0x10
,0x28,0x20,0x70,0x20,0x20,0x20,0x00,0x00,0x00,0x70,0x98,0x98,0x68,0x08,0x70,0x80,0x80,0xB0,0xC8,0x88,0x88,0x88,0x00,0x20,0x00,0x60,0x20,0x20,0x20,0x70,0x00,0x10,0x00,0x10,0x10,0x10,0x90,0x60,0x00,0x80,0x80,0x90,0xA0,0xC0,0xA0,0x90,0x00,0
x60,0x20,0x20,0x20,0x20,0x20,0x70,0x00,0x00,0x00,0xD0,0xA8,0xA8,0xA8,0xA8,0x00,0x00,0x00,0xB0,0xC8,0x88,0x88,0x88,0x00,0x00,0x00,0x70,0x88,0x88,0x88,0x70,0x00,0x00,0x00,0xB0,0xC8,0xC8,0xB0,0x80,0x80,0x00,0x00,0x68,0x98,0x98,0x68,0x08,0x
08,0x00,0x00,0xB0,0xC8,0x80,0x80,0x80,0x00,0x00,0x00,0x78,0x80,0x70,0x08,0xF0,0x00,0x20,0x20,0xF8,0x20,0x20,0x28,0x10,0x00,0x00,0x00,0x88,0x88,0x88,0x98,0x68,0x00,0x00,0x00,0x88,0x88,0x88,0x50,0x20,0x00,0x00,0x00,0x88,0x88,0xA8,0xA8,0x5
0,0x00,0x00,0x00,0x88,0x50,0x20,0x50,0x88,0x00,0x00,0x00,0x88,0x88,0x78,0x08,0x88,0x70,0x00,0x00,0xF8,0x10,0x20,0x40,0xF8,0x00,0x10,0x20,0x20,0x40,0x20,0x20,0x10,0x00,0x20,0x20,0x20,0x00,0x20,0x20,0x20,0x00,0x40,0x20,0x20,0x10,0x20,0x20,
0x40,0x00,0x40,0xA8,0x10,0x00,0x00,0x00,0x00,0x00,0x70,0xD8,0xD8,0x70,0x00,0x00,0x00,0x00}

#define F_8x8
{0x08,0x08,0x08,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x30,0x78,0x78,0x30,0x30,0x00,0x30,0x00,0x6C,0x6C,0x6C,0x00,0x00,0x00,0x00,0x00,0x6C,0x6C,0xFE,0x6C,0xFE,0x6C,0x6C,0x00,0x18,0x3E,0x60,0x3C,0x06,0x7C,0x18,0
x00,0x00,0x63,0x66,0x0C,0x18,0x33,0x63,0x00,0x1C,0x36,0x1C,0x3B,0x6E,0x66,0x3B,0x00,0x30,0x30,0x60,0x00,0x00,0x00,0x00,0x00,0x0C,0x18,0x30,0x30,0x30,0x18,0x0C,0x00,0x30,0x18,0x0C,0x0C,0x0C,0x18,0x30,0x00,0x00,0x66,0x3C,0xFF,0x3C,0x66,0x
00,0x00,0x00,0x30,0x30,0xFC,0x30,0x30,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x30,0x00,0x00,0x00,0x7E,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x00,0x03,0x06,0x0C,0x18,0x30,0x60,0x40,0x00,0x3E,0x63,0x63,0x6B,0x63,0x63
,0x3E,0x00,0x18,0x38,0x58,0x18,0x18,0x18,0x7E,0x00,0x3C,0x66,0x06,0x1C,0x30,0x66,0x7E,0x00,0x3C,0x66,0x06,0x1C,0x06,0x66,0x3C,0x00,0x0E,0x1E,0x36,0x66,0x7F,0x06,0x0F,0x00,0x7E,0x60,0x7C,0x06,0x06,0x66,0x3C,0x00,0x1C,0x30,0x60,0x7C,0x66,0
x66,0x3C,0x00,0x7E,0x66,0x06,0x0C,0x18,0x18,0x18,0x00,0x3C,0x66,0x66,0x3C,0x66,0x66,0x3C,0x00,0x3C,0x66,0x66,0x3E,0x06,0x0C,0x38,0x00,0x00,0x18,0x18,0x00,0x00,0x18,0x18,0x00,0x00,0x18,0x18,0x00,0x00,0x18,0x18,0x30,0x0C,0x18,0x30,0x60,0x
30,0x18,0x0C,0x00,0x00,0x00,0x7E,0x00,0x00,0x7E,0x00,0x00,0x30,0x18,0x0C,0x06,0x0C,0x18,0x30,0x00,0x3C,0x66,0x06,0x0C,0x18,0x00,0x18,0x00,0x3E,0x63,0x6F,0x69,0x6F,0x60,0x3E,0x00,0x18,0x3C,0x66,0x66,0x7E,0x66,0x66,0x00,0x7E,0x33,0x33,0x3E
,0x33,0x33,0x7E,0x00,0x1E,0x33,0x60,0x60,0x60,0x33,0x1E,0x00,0x7C,0x36,0x33,0x33,0x33,0x36,0x7C,0x00,0x7F,0x31,0x34,0x3C,0x34,0x31,0x7F,0x00,0x7F,0x31,0x34,0x3C,0x34,0x30,0x78,0x00,0x1E,0x33,0x60,0x60,0x67,0x33,0x1F,0x00,0x66,0x66,0x66,0x
7E,0x66,0x66,0x66,0x00,0x3C,0x18,0x18,0x18,0x18,0x18,0x3C,0x00,0x0F,0x06,0x06,0x06,0x66,0x66,0x3C,0x00,0x73,0x33,0x36,0x3C,0x36,0x33,0x73,0x00,0x78,0x30,0x30,0x30,0x31,0x33,0x7F,0x00,0x63,0x77,0x7F,0x7F,0x6B,0x63,0x63,0x00,0x63,0x73,0x7B
,0x6F,0x67,0x63,0x63,0x00,0x3E,0x63,0x63,0x63,0x63,0x63,0x3E,0x00,0x7E,0x33,0x33,0x3E,0x30,0x30,0x78,0x00,0x3C,0x66,0x66,0x66,0x6E,0x3C,0x0E,0x00,0x7E,0x33,0x33,0x3E,0x36,0x33,0x73,0x00,0x3C,0x66,0x30,0x18,0x0C,0x66,0x3C,0x00,0x7E,0x5A,0
x18,0x18,0x18,0x18,0x3C,0x00,0x66,0x66,0x66,0x66,0x66,0x66,0x7E,0x00,0x66,0x66,0x66,0x66,0x66,0x3C,0x18,0x00,0x63,0x63,0x63,0x6B,0x7F,0x77,0x63,0x00,0x63,0x63,0x36,0x1C,0x1C,0x36,0x63,0x00,0x66,0x66,0x66,0x3C,0x18,0x18,0x3C,0x00,0x7F,0x6
3,0x46,0x0C,0x19,0x33,0x7F,0x00,0x3C,0x30,0x30,0x30,0x30,0x30,0x3C,0x00,0x60,0x30,0x18,0x0C,0x06,0x03,0x01,0x00,0x3C,0x0C,0x0C,0x0C,0x0C,0x0C,0x3C,0x00,0x08,0x1C,0x36,0x63,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xFF,0x18,
0x18,0x0C,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x3C,0x06,0x3E,0x66,0x3B,0x00,0x70,0x30,0x3E,0x33,0x33,0x33,0x6E,0x00,0x00,0x00,0x3C,0x66,0x60,0x66,0x3C,0x00,0x0E,0x06,0x3E,0x66,0x66,0x66,0x3B,0x00,0x00,0x00,0x3C,0x66,0x7E,0x60,0x3C,0x00,0x
1C,0x36,0x30,0x78,0x30,0x30,0x78,0x00,0x00,0x00,0x3B,0x66,0x66,0x3E,0x06,0x7C,0x70,0x30,0x36,0x3B,0x33,0x33,0x73,0x00,0x18,0x00,0x38,0x18,0x18,0x18,0x3C,0x00,0x06,0x00,0x06,0x06,0x06,0x66,0x66,0x3C,0x70,0x30,0x33,0x36,0x3C,0x36,0x73,0x0
0,0x38,0x18,0x18,0x18,0x18,0x18,0x3C,0x00,0x00,0x00,0x66,0x7F,0x7F,0x6B,0x63,0x00,0x00,0x00,0x7C,0x66,0x66,0x66,0x66,0x00,0x00,0x00,0x3C,0x66,0x66,0x66,0x3C,0x00,0x00,0x00,0x6E,0x33,0x33,0x3E,0x30,0x78,0x00,0x00,0x3B,0x66,0x66,0x3E,0x06,
0x0F,0x00,0x00,0x6E,0x3B,0x33,0x30,0x78,0x00,0x00,0x00,0x3E,0x60,0x3C,0x06,0x7C,0x00,0x08,0x18,0x3E,0x18,0x18,0x1A,0x0C,0x00,0x00,0x00,0x66,0x66,0x66,0x66,0x3B,0x00,0x00,0x00,0x66,0x66,0x66,0x3C,0x18,0x00,0x00,0x00,0x63,0x6B,0x7F,0x7F,0x
36,0x00,0x00,0x00,0x63,0x36,0x1C,0x36,0x63,0x00,0x00,0x00,0x66,0x66,0x66,0x3E,0x06,0x7C,0x00,0x00,0x7E,0x4C,0x18,0x32,0x7E,0x00,0x0E,0x18,0x18,0x70,0x18,0x18,0x0E,0x00,0x0C,0x0C,0x0C,0x00,0x0C,0x0C,0x0C,0x00,0x70,0x18,0x18,0x0E,0x18,0x1
8,0x70,0x00,0x3B,0x6E,0x00,0x00,0x00,0x00,0x00,0x00,0x1C,0x36,0x36,0x1C,0x00,0x00,0x00,0x00}

#define F_8x16
{0x08,0x10,0x10,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x3C,0x3C,0x3C,0x18,0x18,0x18,0x00,0x18,0x18,0x00,0x00,0x00,0
x00,0x00,0x63,0x63,0x63,0x22,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x36,0x36,0x7F,0x36,0x36,0x36,0x7F,0x36,0x36,0x00,0x00,0x00,0x00,0x0C,0x0C,0x3E,0x63,0x61,0x60,0x3E,0x03,0x03,0x43,0x63,0x3E,0x0C,0x0C,0x0
0,0x00,0x00,0x00,0x00,0x00,0x00,0x61,0x63,0x06,0x0C,0x18,0x33,0x63,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x1C,0x36,0x36,0x1C,0x3B,0x6E,0x66,0x66,0x3B,0x00,0x00,0x00,0x00,0x00,0x30,0x30,0x30,0x60,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x0C,0x18,0x18,0x30,0x30,0x30,0x30,0x18,0x18,0x0C,0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x0C,0x0C,0x06,0x06,0x06,0x06,0x0C,0x0C,0x18,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x42,0x66,0x3C,0xFF,0x3C,0x66,0x42,0x00,0x00,0x
00,0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x18,0xFF,0x18,0x18,0x18,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x18,0x30,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xFF,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x03,0x07,0x0E,0x1C,0x38,0x70,0xE0,0xC0,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x63,0x63,0x6B,0x6B,0x63,0x63,0x63,0x3E,0x
00,0x00,0x00,0x00,0x00,0x00,0x0C,0x1C,0x3C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x3F,0x00,0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x03,0x06,0x0C,0x18,0x30,0x61,0x63,0x7F,0x00,0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x03,0x03,0x1E,0x03,0x03,0x03,0x63,0x3E
,0x00,0x00,0x00,0x00,0x00,0x00,0x06,0x0E,0x1E,0x36,0x66,0x66,0x7F,0x06,0x06,0x0F,0x00,0x00,0x00,0x00,0x00,0x00,0x7F,0x60,0x60,0x60,0x7E,0x03,0x03,0x63,0x73,0x3E,0x00,0x00,0x00,0x00,0x00,0x00,0x1C,0x30,0x60,0x60,0x7E,0x63,0x63,0x63,0x63,0x
3E,0x00,0x00,0x00,0x00,0x00,0x00,0x7F,0x63,0x03,0x06,0x06,0x0C,0x0C,0x18,0x18,0x18,0x00,0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x63,0x63,0x3E,0x63,0x63,0x63,0x63,0x3E,0x00,0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x63,0x63,0x63,0x3F,0x03,0x03,0x06
,0x3C,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x00,0x00,0x00,0x18,0x18,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x00,0x00,0x00,0x18,0x18,0x18,0x30,0x00,0x00,0x00,0x00,0x00,0x06,0x0C,0x18,0x30,0x60,0x30,0x18,0
x0C,0x06,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x7E,0x00,0x00,0x7E,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x60,0x30,0x18,0x0C,0x06,0x0C,0x18,0x30,0x60,0x00,0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x63,0x06,0x0C,0x0C,0x0C,0x0
0,0x0C,0x0C,0x00,0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x63,0x6F,0x6B,0x6B,0x6E,0x60,0x60,0x3E,0x00,0x00,0x00,0x00,0x00,0x00,0x08,0x1C,0x36,0x63,0x63,0x63,0x7F,0x63,0x63,0x63,0x00,0x00,0x00,0x00,0x00,0x00,0x7E,0x33,0x33,0x33,0x3E,0x33,0x33,
0x33,0x33,0x7E,0x00,0x00,0x00,0x00,0x00,0x00,0x1E,0x33,0x61,0x60,0x60,0x60,0x60,0x61,0x33,0x1E,0x00,0x00,0x00,0x00,0x00,0x00,0x7C,0x36,0x33,0x33,0x33,0x33,0x33,0x33,0x36,0x7C,0x00,0x00,0x00,0x00,0x00,0x00,0x7F,0x33,0x31,0x34,0x3C,0x34,0x
30,0x31,0x33,0x7F,0x00,0x00,0x00,0x00,0x00,0x00,0x7F,0x33,0x31,0x34,0x3C,0x34,0x30,0x30,0x30,0x78,0x00,0x00,0x00,0x00,0x00,0x00,0x1E,0x33,0x61,0x60,0x60,0x6F,0x63,0x63,0x37,0x1D,0x00,0x00,0x00,0x00,0x00,0x00,0x63,0x63,0x63,0x63,0x7F,0x63
,0x63,0x63,0x63,0x63,0x00,0x00,0x00,0x00,0x00,0x00,0x3C,0x18,0x18,0x18,0x18,0x18,0x18,0x18,0x3C,0x00,0x00,0x00,0x00,0x00,0x00,0x0F,0x06,0x06,0x06,0x06,0x06,0x06,0x66,0x66,0x3C,0x00,0x00,0x00,0x00,0x00,0x00,0x73,0x33,0x36,0x36,0x3C,0
x36,0x36,0x33,0x33,0x73,0x00,0x00,0x00,0x00,0x00,0x00,0x78,0x30,0x30,0x30,0x30,0x30,0x30,0x31,0x33,0x7F,0x00,0x00,0x00,0x00,0x00,0x00,0x63,0x77,0x7F,0x6B,0x63,0x63,0x63,0x63,0x63,0x63,0x00,0x00,0x00,0x00,0x00,0x00,0x63,0x63,0x73,0x7B,0x7
F,0x6F,0x67,0x63,0x63,0x63,0x00,0x00,0x00,0x00,0x00,0x00,0x1C,0x36,0x63,0x63,0x63,0x63,0x63,0x63,0x36,0x1C,0x00,0x00,0x00,0x00,0x00,0x00,0x7E,0x33,0x33,0x33,0x3E,0x30,0x30,0x30,0x30,0x78,0x00,0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x63,0x63,
0x63,0x63,0x63,0x6B,0x6F,0x3E,0x06,0x07,0x00,0x00,0x00,0x00,0x7E,0x33,0x33,0x33,0x3E,0x36,0x36,0x33,0x33,0x73,0x00,0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x63,0x30,0x1C,0x06,0x03,0x63,0x63,0x3E,0x00,0x00,0x00,0x00,0x00,0x00,0xFF,0xDB,0x99,0x
18,0x18,0x18,0x18,0x18,0x18,0x3C,0x00,0x00,0x00,0x00,0x00,0x00,0x63,0x63,0x63,0x63,0x63,0x63,0x63,0x63,0x63,0x3E,0x00,0x00,0x00,0x00,0x00,0x00,0x63,0x63,0x63,0x63,0x63,0x63,0x63,0x36,0x1C,0x08,0x00,0x00,0x00,0x00,0x00,0x00,0x63,0x63,0x63
,0x63,0x63,0x6B,0x6B,0x7F,0x36,0x36,0x00,0x00,0x00,0x00,0x00,0x00,0xC3,0xC3,0x66,0x3C,0x18,0x18,0x3C,0x66,0xC3,0xC3,0x00,0x00,0x00,0x00,0x00,0x00,0xC3,0xC3,0xC3,0x66,0x3C,0x18,0x18,0x18,0x18,0x3C,0x00,0x00,0x00,0x00,0x00,0x00,0x7F,0x63,0
x43,0x06,0x0C,0x18,0x30,0x61,0x63,0x7F,0x00,0x00,0x00,0x00,0x00,0x00,0x3C,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x3C,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0xC0,0xE0,0x70,0x38,0x1C,0x0E,0x07,0x03,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x3C,0x0
C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x3C,0x00,0x00,0x00,0x00,0x08,0x1C,0x36,0x63,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xFF,0x00,0x00,0x00,0x18,0x18,0x0C,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x3C,0x46,0x06,0x3E,0x66,0x66,0x3B,0x00,0x00,0x00,0x00,0x00,0x00,0x70,0x30,0x30,0x3C,0x36,0x33,0x33,0x33,0x33,0x6E,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,0x00,0x00,0x3E,0x63,0x60,0x60,0x60,0x63,0x3E,0x00,0x00,0x00,0x00,0x00,0x00,0x0E,0x06,0x06,0x1E,0x36,0x66,0x66,0x66,0x66,0x3B,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x63,0x7E,0x60,0x63,0x3E,0x00,0x00,0x00,0x00,0x00,0x00
,0x1C,0x36,0x32,0x30,0x7C,0x30,0x30,0x30,0x30,0x78,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x3B,0x66,0x66,0x66,0x66,0x3E,0x06,0x66,0x3C,0x00,0x00,0x00,0x00,0x70,0x30,0x30,0x36,0x3B,0x33,0x33,0x33,0x33,0x73,0x00,0x00,0x00,0x00,0x00,0x0
x00,0x0C,0x0C,0x00,0x1C,0x0C,0x0C,0x0C,0x0C,0x0C,0x1E,0x00,0x00,0x00,0x00,0x00,0x00,0x06,0x06,0x00,0x0E,0x06,0x06,0x06,0x06,0x06,0x66,0x66,0x3C,0x00,0x00,0x00,0x00,0x70,0x30,0x30,0x33,0x33,0x36,0x3C,0x36,0x33,0x73,0x00,0x00,0x00,0x00,0x
00,0x00,0x1C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x1E,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x6E,0x7F,0x6B,0x6B,0x6B,0x6B,0x6B,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x6E,0x33,0x33,0x33,0x33,0x33,0x33,0x00,0x00,0x00,0x0
0,0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x63,0x63,0x63,0x63,0x3E,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x6E,0x33,0x33,0x33,0x33,0x3E,0x30,0x30,0x78,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x3B,0x66,0x66,0x66,0x66,0x3E,0x06,0x06,0x0F,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x6E,0x3B,0x33,0x30,0x30,0x30,0x78,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x38,0x0E,0x03,0x63,0x3E,0x00,0x00,0x00,0x00,0x00,0x00,0x08,0x18,0x18,0x7E,0x18,0x18,0x18,0x18,0x1B,0x0E,0x00,0x00,0x
00,0x00,0x00,0x00,0x00,0x00,0x00,0x66,0x66,0x66,0x66,0x66,0x66,0x3B,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x63,0x63,0x36,0x36,0x1C,0x1C,0x08,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x63,0x63,0x63,0x6B,0x6B,0x7F,0x36,0x00,0x0
0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x63,0x36,0x1C,0x1C,0x1C,0x36,0x63,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x63,0x63,0x63,0x63,0x63,0x3F,0x03,0x06,0x3C,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x7F,0x66,0x0C,0x18,0x30,0x63,0x7F,0x00,
0x00,0x00,0x00,0x00,0x00,0x0E,0x18,0x18,0x18,0x70,0x18,0x18,0x18,0x18,0x0E,0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x18,0x18,0x18,0x00,0x18,0x18,0x18,0x18,0x18,0x00,0x00,0x00,0x00,0x00,0x70,0x18,0x18,0x18,0x0E,0x18,0x18,0x18,0x18,0x70,0x
00,0x00,0x00,0x00,0x00,0x00,0x3B,0x6E,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x70,0xD8,0xD8,0x70,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
```

… have fun!

edgarmarx@t-online.de