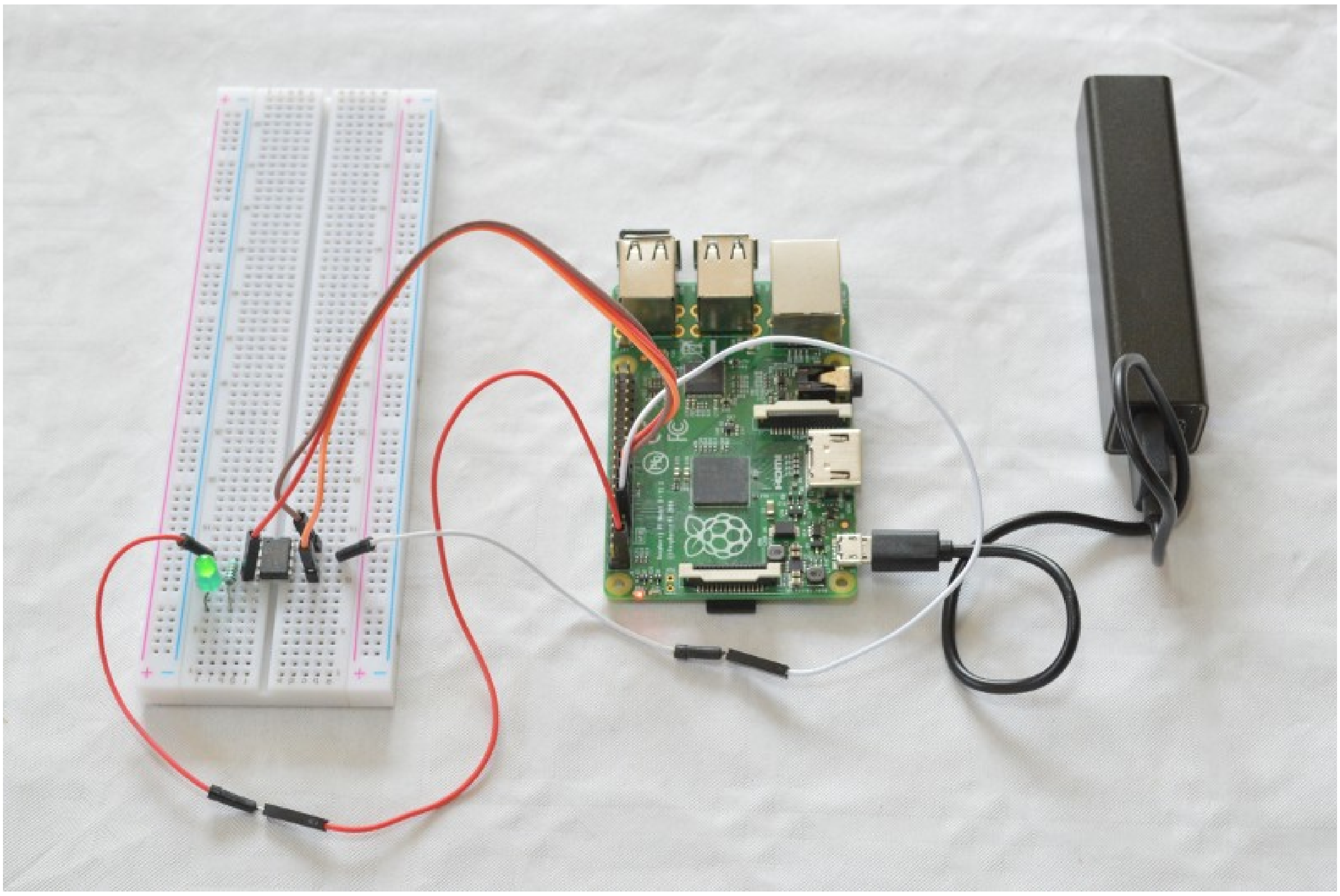
A simple programmer for

PIC LVP’s



Done By:
Henrik Kressner

Table of Contents

1. Intro

It came to my attention, that Microchip was making LVP (Low Voltage Programming) versions, they can be programmed at 3.3 Volts, wich mean, they can be programed directly from the GPIO of a Raspberry Pi.

I read Giorgio Vazzana great page on making af programmer, that inspired my to do this project.

I expect the reader have basic knowledge of electronics and mikrocontrollere, and to know you way around the Raspberry Pi in the CLI. This means you should be able to:

- use simple CLI commands
- ssh to a RPi
- compile a C program

Remember to read the manuals (RTM), its on Microchips website, and a lot of other places. Search for ”DS40001585B” or ”datasheet PIC10f” for the hardware manual, and ”DS41572D” for programmers refence.

I only testet with a 10F322, but I asume the programmer should be usable for all Microchip LVP chip, just by changing the `KEYSEQUENCE` to the one that the specific PIC require. (RTM)

Done By
Henrik Kressner
kressner@synkro.dk
May 2020

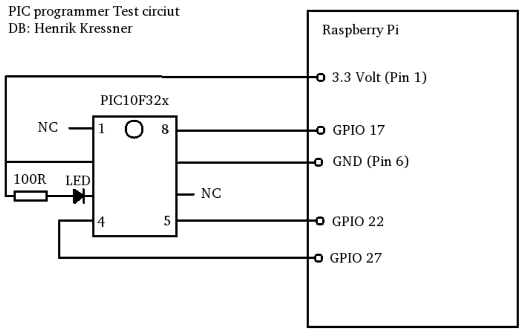
2. The programmer

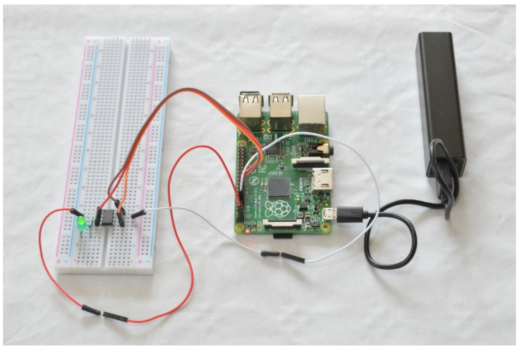
The programmer is a simple peace of hardware, it consist of:

1 RPi
5 wires
1 breadbord, or what ever you prefere
1 gpasm, the GNU assambler for microchip controllers

And the pic program, sourcecode shown in appendix A.

The diagram add a LED for testing.





The photo show a Raspberry Pi 2 operated on battery and WiFi, it can’t be more easy to program a PIC. You can even run it on a A model, with screen and keyboard, or wireless as this one.

Be aware: The sourcecode has to be adjustet to the type of RPi, so the `BCM2708_PERI_BASE` fits the one you are using. See the sourcecode for explanaition.

3 Blink.asm

A simple testprogram.

```
;           gpasm -a inhx8m blink.asm
;           Port 2 ON/OFF
;           DB Henrik Kressner

processor           10F322
include             p10f322.inc

            __CONFIG _WDTE_OFF & _FOSC_INTOSC

;           Variables in ram
i           org     0x40    ; General purpose registers start at 0x40
j           org     0x41
k           org     0x42

            org     H'00'   ; Reset vector
            goto    setup

; Pass interrupt vector
                    org     H'05'

setup       movlw   B'11111011'
            movwf   TRISA           ; Port is 2 output

run         movlw   B'00000100'     ; Port 2 ON
            movwf   PORTA
            call    delay
            movlw   B'00000000'     ; Port 2 OFF
            movwf   PORTA
            call    delay
            goto    run

delay       clrf    i
            clrf    j
            movlw   0x08
            movwf   k
loop        decfsz  i,1
            goto    loop
            decfsz  j,1
            goto    loop
            decfsz  k,1
            goto    loop
            return

            end
```

This program starts by defining 3 variables (i, j and k), place them in the RAM, that start at 40H.

After initialisation, we turn the LED on, take a pause (calling delay), after that we turn the LED off, take a pause and return to run.

The delay is made by 3 loops in each other. We start by setting i = 0, so when we run decfsz i,1, the result is FFH, wich is not zero, therefore we call loop, until we reach zero.

When we reach zero, we go into the next loop and ask: decfsz j,1. This time j = FFH after the decrementation, so we jump to loop, and rund the first loop 256 times, before we try again, and so on.

The inner loop is where we fine tune the delay. Therefore we set k = 8H.

You can calculate the exact delay by counting clockpulses. With this setting, it should blink around ½ Hertz.

Compile this assambler program to a hex file by typing: `gpasm -a inhx8m blink.asm`

When pic.c is compiled (must be root) with: `cc pic.c -o pic` you can download the hex file to the 10F322 from CLI by writing:

(must be root): `cat filename.hex | ./pic`

Appendix A

This sourcecode is for Rpi 2, for later models read the comment about: `#define BCM2708_PERI_BASE`

```
//  Use a RPi 2 as LVP programmer
//  For RPi 3 see comment.
//  Only testet with a 10F322

//  Inspired by: Holden (Giorgio Vazzana) at http://holdenc.altervista.org/rpp/
//  Done by Henrik Kressner 2020

//  This is free software, use at your own risc.

// ToDo:
// Read out in HEX
// Run as extern clock
// Test for error
// Test and use arguments
// Web interface


//  To compile (must be root) :  cc pic.c -o pic
//  To run (must be root): cat filename.hex | ./pic


#define VERSION "0.19"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>
#include <time.h>

//  Access from ARM Running Linux
//  For Raspberry Pi 3, change BCM2708_PERI_BASE to 0x3F000000
#define BCM2708_PERI_BASE        0x20000000
#define GPIO_BASE                (BCM2708_PERI_BASE + 0x200000) /* GPIO controller */
#define PAGE_SIZE (4*1024)
#define BLOCK_SIZE (4*1024)

# define VPP 17
# define ICSPCLK 27
# define ICSPDAT 22
# define TCK 100      // Delay 100 uS
# define TDIS 100     // Delay 5 mS
# define TDLY 1       // Delay 1 uS
# define TERAB 5000   // Delay 5 mS
# define TERAR 2500   // Delay 2.5 mS
# define TEXIT 1      // Delay 1 uS
# define TPEXT 1000   // Delay 1 mS
# define TPINT 2500   // Delay 2.5 mS
# define KEYSEQUENCE "01001101010000110100100001010000" // From programmers manual page 10

int  mem_fd;
char *gpio_mem, *gpio_map;
char *spi0_mem, *spi0_map;

// I/O access
volatile unsigned *gpio;
```

```
// GPIO setup macros. Always use INP_GPIO(x) before using OUT_GPIO(x)
// or SET_GPIO_ALT(x,y)
#define INP_GPIO(g) *(gpio+((g)/10)) &= ~(7<<(((g)%10)*3))
#define OUT_GPIO(g) *(gpio+((g)/10)) |=  (1<<(((g)%10)*3))
#define SET_GPIO_ALT(g,a) *(gpio+(((g)/10))) |= (((a)<=3?(a)+4:(a)==4?3:2)<<(((g)%10)*3))

#define GPIO_SET *(gpio+7)  // sets   bits which are 1 ignores bits which are 0
#define GPIO_CLR *(gpio+10) // clears bits which are 1 ignores bits which are 0
#define GET_GPIO(g) (*(gpio+13)&(1<<g)) // 0 if LOW, (1<<g) if HIGH

void setup_io()
{
   /* open /dev/mem */
   if ((mem_fd = open("/dev/mem", O_RDWR|O_SYNC) ) < 0) {
      printf("can't open /dev/mem \n");
      exit (-1);
   }
   /* mmap GPIO */
   // Allocate MAP block
   if ((gpio_mem = malloc(BLOCK_SIZE + (PAGE_SIZE-1))) == NULL) {
      printf("allocation error \n");
      exit (-1);
   }
   // Make sure pointer is on 4K boundary
   if ((unsigned long)gpio_mem % PAGE_SIZE)
     gpio_mem += PAGE_SIZE - ((unsigned long)gpio_mem % PAGE_SIZE);
   // Now map it
   gpio_map = (unsigned char *)mmap(
      (caddr_t)gpio_mem,
      BLOCK_SIZE,
      PROT_READ|PROT_WRITE,
      MAP_SHARED|MAP_FIXED,
      mem_fd,
      GPIO_BASE
   );

   if ((long)gpio_map < 0) {
      printf("mmap error %d\n", (int)gpio_map);
      exit (-1);
   }
   // Always use volatile pointer!
   gpio = (volatile unsigned *)gpio_map;
} // setup_io


void help()
{
       printf("\nPIC 10Fxxx Lov Voltage Programmer version: %s : Syntax:\n", VERSION);
       printf("pic f <filename.hex> -> ");
       printf("Write hex file to 10F32x (Not implementet, use piping)\n");
       printf("pic rb <a> <b> -> ");
       printf("Read binary from 10F32x address a to address b\n");
       printf("pic rh <a> <b> -> ");
       printf("Read hex from 10F32x address a to address b (Not implementet\n");
       printf("No argument -> asuming hex file is piped:  cat hexfile.hex | pic\n");
       printf("DB Henrik Kressner 2020.\n");
}
```

```
// Convert txt based Hex on pos a to b to a int
int getIntFromString(char tmp[], int a, int b)
{
        int i, sum = 0;

        for (i = a; i <= b; i++)
        {
                sum = sum * 16;
                if (tmp[i] >= '0' && tmp[i] <= '9')
                        sum = sum + (tmp[i] - '0');
                else if (tmp[i] >= 'A' && tmp[i] <= 'F')
                        sum = sum + (tmp[i] - 'A' + 10);
        }
       return sum;
}

// Convert one byte to a array of char,
// with one bit per char
void CharToBit(char x, char holder[9])
{
        char i;

        holder[8] = '\0';
        for (i = 0; i < 8; ++i)
                if ( (x << i) & 128 )
                        holder[i] = '1';
                else
                        holder[i] = '0';
}       // CharToBit


// Send a command, with data, to PIC10F32x
// Command = 00H : Load Configuration
// Command = 02H : Load Data For Program Memory
// Command = 04H : Read Data From Program Memory
// Command = 06H : Increment Address
// Command = 16H : Reset Address
// Command = 08H : Begin Internally Timed Programming
// Command = 18H : Begin Externally Timed Programming
// Command = 0AH : End Externally Timed Programming
// Command = 09H : Bulk Erase Program Memory
// Command = 11H : Row Erase Program Memory
void SendCommand(char Command[])
{
        int i;

   INP_GPIO(ICSPCLK); // Must use INP_GPIO before we can use OUT_GPIO
   OUT_GPIO(ICSPCLK);
   INP_GPIO(ICSPDAT);
   OUT_GPIO(ICSPDAT);
        GPIO_CLR = 1<<ICSPCLK;
   for (i = 7; i > 1; i--)
        {
                if (Command[i] == '1')
                        GPIO_SET = 1<<ICSPDAT;
                else
                        GPIO_CLR = 1<<ICSPDAT;
                fflush(stdout);
                usleep(TCK);
                // One clock cycle
                GPIO_SET = 1<<ICSPCLK;
                usleep(TCK);
                GPIO_CLR = 1<<ICSPCLK;
                usleep(TCK);
        }
}       // SendCommand
```

```
void IncrementPC()
{
        char tmp[9];

        CharToBit(0x06, tmp);     // Increment PC
        SendCommand(tmp);
        usleep(TCK);
}

void WriteConfig(char ConfigWord[], int Address)
{
        int i;
        char tmp[9];
        char WriteData[8];

        printf("Config address = %x\n", Address);
        GPIO_CLR = 1<<ICSPCLK;
        usleep(TPINT);
        CharToBit(0x00, tmp);         // Load Configuration
        SendCommand(tmp);                           // Now PC = 0x2000
        printf("Write config to device at address %x:\t", Address);
        // Data is 14 bit
        usleep(TPINT);
        for (i=15; i>=0; i--)
        {
                GPIO_SET = 1<<ICSPCLK;
                if (ConfigWord[i] == '1')
                        GPIO_SET = 1<<ICSPDAT;
                else
                        GPIO_CLR = 1<<ICSPDAT;
                printf("%c", ConfigWord[i]);
                usleep(TCK);
//              fflush(stdout);
                GPIO_CLR = 1<<ICSPCLK;
                usleep(TCK);
        }
        // Inc PC
        for (i=0; i<Address-0x2000; i++)
                IncrementPC();

        CharToBit(0x08, WriteData);     // Begin Internally Timed Programming
        SendCommand(WriteData);
        usleep(TPINT);
} // WriteConfig

// Manual page 12
void WriteData(char x[])
{
        int i;
        char WriteData[8];

        GPIO_CLR = 1<<ICSPCLK;
        usleep(TPINT);
        CharToBit(0x02, WriteData);     // Load Data For Program Memory
        SendCommand(WriteData);
        printf("Write data to device :\t\n");
        // Selve data (14 bit)
        usleep(TPINT);
        for (i=15; i>=0; i--)
        {
                // Clock up
                GPIO_SET = 1<<ICSPCLK;
                if (x[i] == '1')
                        GPIO_SET = 1<<ICSPDAT;
                else
                        GPIO_CLR = 1<<ICSPDAT;
                usleep(TCK);
//              fflush(stdout);
                // Clock down
                GPIO_CLR = 1<<ICSPCLK;
                usleep(TCK);
        }
        CharToBit(0x08, WriteData);     // Begin Internally Timed Programming
        SendCommand(WriteData);
        usleep(TPINT);
}  // WriteDate


// Manual page 12
void ReadData(char x[])
{
        int i;
        char READDATA[8];

        GPIO_CLR = 1<<ICSPCLK;
        CharToBit(0x04, READDATA);
        SendCommand(READDATA);
        usleep(TDLY);
        printf("Reading from device :\t");
        INP_GPIO(ICSPDAT); // Dont need out, this is read.
        for (i=15; i>=0; i--)
        {
                GPIO_SET = 1<<ICSPCLK;
                usleep(TCK);
                GPIO_CLR = 1<<ICSPCLK;
                usleep(TCK);
                if (GET_GPIO(ICSPDAT))
                        x[i] = '1';
                else
                        x[i] = '0';
                if (i== 0 || i == 15)
                        x[i] = 'x';
                if (i== 7)
                        printf("%c", ' ');
                printf("%c", x[i]);
                fflush(stdout);
                usleep(TCK);
        }
}  // ReadDate


// Transmit n bit to port, one bit at the time LSB first
void ByteToPin(char x[], int n)
{
        int i;

        n--;  // Array starts at 0
        GPIO_CLR = 1<<ICSPCLK;
         for (i = n; i >= 0; i--)
         {
                if (x[i] == '1')
                        GPIO_SET = 1<<ICSPDAT;
                else
                        GPIO_CLR = 1<<ICSPDAT;
//                      fflush(stdout);
                        // One clock cycle
                        GPIO_SET = 1<<ICSPCLK;
                        usleep(TCK);
                        GPIO_CLR = 1<<ICSPCLK;
                        usleep(TCK);
         }
}       // ByteToPin

void ReadPIC(int from, int to)
{
        int i;
        char tmp[9];
        char holder[17];

        // Read i address
        printf("Reading from %d to %d\n", from, to);
        CharToBit(0x16, tmp);     // Reset PC
        SendCommand(tmp);
        for (i = 0; i <from; i++)
                IncrementPC();
        for (i = from; i <=to; i++)
        {
                printf("PC = %d : ",i);
                ReadData(holder);
                printf("\n");
                IncrementPC();
        }
} // ReadPIC

void WritePIC(char Line[], char BulkErase)
{
        char tmp[9], tmp1[9], tmp2[9];
        char DataWord[17];
        int i,j,k,l,m;
        int ByteCount, Address;

        i = 0;
        ByteCount = getIntFromString(Line,1,2)/2;          // Programmeres ref. page 25
        Address   = getIntFromString(Line,3,6)/2;
        if (BulkErase)
        {
           CharToBit(0x09, tmp);     // Bulk erase
           SendCommand(tmp);
                usleep(TERAB);
        }

        // Set PC to Address
        CharToBit(0x16, tmp);     // Reset PC
        SendCommand(tmp);

        if (Address < 0x2000)
                for (i=0; i<Address; i++)
                        IncrementPC();

        // Find DataWords
        for (i=0; i<ByteCount*4; i+=4)
        {
                DataWord[0] = '\0';
                j = getIntFromString(Line, i+9, i+9);
                k = getIntFromString(Line, i+10, i+10);
                j = j << 4 | k;
                CharToBit(j, tmp1);
                j = getIntFromString(Line, i+11, i+11);
                k = getIntFromString(Line, i+12, i+12);
                j = j << 4 | k;
                CharToBit(j, tmp2);
                strcat(DataWord, tmp2);
                strcat(DataWord, tmp1);

                // Move one bit left
                for (l=0; l<16; l++)
                        DataWord[l] = DataWord[l+1];
                DataWord[0] = DataWord[15] = 'x';

                if (Address >= 0x2000)               // Its Config
                        WriteConfig(DataWord, Address);
                else
                        WriteData(DataWord);

                usleep(TDLY);
                IncrementPC();
                usleep(TCK);
        }
} // WritePIC
```

```
void InitGPIO()
{
        setup_io();
        INP_GPIO(ICSPCLK);      // ICSPCLK as input
        INP_GPIO(ICSPDAT);      // ICSPDAT som input
        INP_GPIO(VPP);                  // must use INP_GPIO before we can use OUT_GPIO
        OUT_GPIO(VPP);
        GPIO_SET = 1<<VPP;
        usleep(10);
        GPIO_CLR = 1<<VPP;      // Now controller is reset, we can enable ICSPCLK & ICSPDAT
        OUT_GPIO(ICSPCLK);
        OUT_GPIO(ICSPDAT);
        usleep(10);

        // Init for lov voltage programming
        ByteToPin(KEYSEQUENCE,32);
        // KEYSEQUENCE needs 33 clockcycle
        GPIO_SET = 1<<ICSPCLK;
        usleep(TCK);
        GPIO_CLR = 1<<ICSPCLK;
        usleep(TCK);
} //InitGPIO

void writeToPIC()
{
        char test[256];

        scanf("%s", test);
        if (test[0] == ':')
                WritePIC(test,1);
        else
        {
                printf("ERROR: Missing : in hex file first line\n");
                return;
        }
        scanf("%s", test);
        while( strcmp(test, ":00000001FF") )
        {
                printf("\n! %s\n", test);
                if (test[0] == ':')
                        WritePIC(test,0);
                else
                        printf("ERROR: Missing :\n");
                scanf("%s", test);
        }
        printf("\n");
} // writeToPIC
```

```
void main(int a, char *s[])
{
        int i;

        printf("PIC10Fxxx LVP programmer version: %s\n", VERSION);
        InitGPIO();
        if ( a == 1 )
                writeToPIC();
        if ( a == 2 )
        {
                if ( !strcmp("f", s[1]) )
                        printf("Read from hex file: Not implementet\n");
                if ( !strcmp("h", s[1]) || !strcmp("h", s[1]) )
                        help();
        }
        if ( a == 4 )
        {
                if ( !strcmp("rb", s[1]) )
                        ReadPIC( atoi(s[2]), atoi(s[3]) );
                        else if ( !strcmp("rh", s[1]) )
                                printf("Hex out: Not implementet\n");
                                else help();
        }

        // Set CLK and DAT to input and exit program mode
        INP_GPIO(ICSPCLK);
        INP_GPIO(ICSPDAT);
        usleep(TCK);
        // Now we can raise Vpp
        OUT_GPIO(VPP);
        GPIO_SET = 1<<VPP;
}
```