MIX-fpga

Michael Schröder
mi.schroeder@netcologne.de

May 10, 2021

1. A real MIX

Have you ever heard of Don Knuths (hypothetical) first polyunsaturated computer MIX, the 1009? In this project we will build a binary version of the MIX-Computer as described in "The Art of Computer Programming, Vol. 1" by Donald E. Knuth running on an fpga-board.

The presented implementation is based on the fpga development board iCE40HX8K-EVB from the company Olimex Ltd.[1], which has the nice property of being completely open source. The whole project uses only FOSS free and open source hard- and software, so everybody can build their own MIX following the instructions on the project site[2].

Figure 1: The real MIX

[1] www.olimex.com
[2] www.gitlab.com/x653/mix-fpga

1.1. A look inside

The MIX computer is composed of two little boards.

1. iCE40HX8K-EVB[3], the fpga development board from the company Olimex Ltd.
2. USB-serial adapter[4]. Used to power the board with 5V and to in-/output data over the serial interface.

Figure 2: Look inside and see the two boards interconnected with wire wrap technique

1.2. Specifications

1.2.1. clock

MIX runs on iCE40HX8K-EVB clocked at 25 MHz. The fundamental unit of time u corresponds to $1u = 40$ ns, so according to Knuth it's a relatively high priced machine.

1.2.2. I/O units

In our MIX implementation all character based I/O units (U16 – U20) are connected to the USB-connector and can be accessed as serial data streams. You can connect MIX with any PC running a terminal emulator (e.g. screen for linux). The terminal should be set to 115200 baud (8N1). A conversion between ASCII and Knuths character codes is done in hardware according to Knuths specification (see TAOCP p. 128).

[3] www.olimex.com/Products/FPGA/iCE40/iCE40HX8K-EVB
[4] www.olimex.com/Products/Breadboarding/BB-CH340T

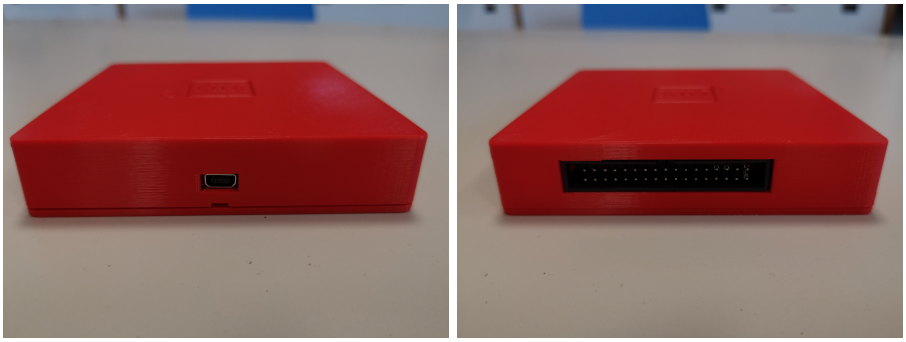


Figure 3: All character based I/O is done through a USB to serial adapter @115200 baud (8N1). MIX can be expanded through a GPIO connector placed at the rear side

1.2.3. commands

All commands exept the floating point arithmetic are implemented with execution times corresponding to Knuth's specifications. Special care is given to the correct timings. Even the *sofisticated* commands `SRC` and `SLC`, which need a fancy modulo 10 computation are executed in the defined timing of two cycles. The system can (easily) be extended in various ways:

1. more commands:
   - easy: add logic operators (`AND`, `OR`, `XOR`, `NOT`)
   - not so easy: add floating point arithmetic (`FADD`, `FSUB`, `FMUL`, `FDIV`)
2. more hardware:
   - easy: add leds to run the traffic light example (s. chapter 5 of this manual)
   - not so easy: add more I/O units

1.2.4. The GO button

MIX comes with the *GO button* attached to USB-UART. So after pressing the *GO button* MIX-programms can be uploaded by sending the *punched cards* to USB-UART.

1.2.5. The toast case

MIX comes in a nice case with formfactor of a slice of toast (10 cm × 10 cm × 2 cm), so your complete MIX computer system will easily fit into your lunch box. The case can be printed with a 3D printer. Design files can be found in the directory `build/toast` of the project site.

2. Running programs on MIX

2.1. Requirement

To run mixal programs on your MIX you need a few things:

- `mixasm`: Assembler provided in the package mdk
- `python`: Some tools are written in python
- `gitlab.com/x653/mix-fpga`: The authors repository of this project
- `screen`: a terminal emulator to connect to MIX

Open a terminal and install the required packages:

```
sudo apt install mdk
sudo apt install python
sudo apt install python-pip
git clone https://gitlab.com/x653/mix-fpga
```

2.2. mixasm

To run mixal programs with MIX you first have to translate the mixal programs to machine language. This is done with the GNU tanslator `mixasm` distributed with the mix software package `mdk`.

```
mixasm <filename>.mixal -l
```

The `-l` option is needed to create the listing file `<filename>.mls`.

2.3. tools

To upload the code onto MIX we have to translate the listing in a computer readable format. For this you can use the python scripts in the subdirector `mix-fpga/tools`:

- `mls2card.py`: translate the machine code listing (`<filename>.mls`) to punched card format.
- `mls2char.py`: translate the listing to character codes, which can directly be read by MIX. (This is only necessary for the bootloader written on the first two punched cards).
- `mls2bin.py`: translate the listing file to binary. This is only necessary for the program *go-button*, which must be uploaded directly into fpga ROM.

2.4. upload and run

Finally you can connect your PC to MIX with a USB cable and start the terminal emulator (`screen`):

```
screen /dev/ttyUSB0 115200
```

Press the *go button* to start MIX. Within the screen session you can upload the mixal programs written on punch cards:

```
<ctr–a> : readreg p p.card
<ctrl–a> : paste p
```

2.5. Verify

The following programms in the folder `mix–fpga/mixal` have been verified to run on MIX:

- `p`: compute the first 500 primes
- `e`: compute easter dates from 1950 – 2000
- `t`: control traffic signals
- `go`: the go button
- `boot`: the bootloader

3. Example program p

We will run programm `p` of chapter 1.3.2 TAOCP (p. 148) on MIX. Programm `p` computes the first 500 primes and outputs them in a table on the line printer U18.

3.1. Prepare the input

3.1.1. p.mixal

The mixal program can be found in `mixal/p/p.mixal`:

```
cd mixal/p
cat p.mixal
```

```
1 * EXAMPLE PROGRAM ... TABLE OF PRIMES
2 *
3 L       EQU     500     The number of primes to find
4 PRINTER EQU     18      Unit number of the line printer
5 PRIME   EQU     2999
6 BUF0    EQU     2000    Memory area for BUFFER[0]
7 BUF1    EQU     BUF0+25 Memory area for BUFFER[1]
8         ORIG    1000
```

```
START   IOC     0(PRINTER) Skip to new page
        LD1     =1-L=
        LD2     =3=
2H      INC1    1
        ST2     PRIME+L,1
        J1Z     2F
4H      INC2    2
        ENT3    2
6H      ENTA    0
        ENTX    0,2
        DIV     PRIME,3
        JXZ     4B
        CMPA    PRIME,3
        INC3    1
        JG      6B
        JMP     2B
2H      OUT     TITLE(PRINTER)
        ENT4    BUF1+10
        ENT5    -50
2H      INC5    L+1
4H      LDA     PRIME,5
        CHAR
        STX     0,4(1:4)
        DEC4    1
        DEC5    50
        J5P     4B
        OUT     0,4(PRINTER)
        LD4     24,4
        J5N     2B
        HLT
* INITIAL CONTENTS OF TABLES AND BUFFERS
        ORIG    PRIME+1
        CON     2
        ORIG    BUF0-5
TITLE   ALF     "FIRST"
        ALF     "␣FIVE"
        ALF     "␣HUND"
        ALF     "RED␣P"
        ALF     "RIMES"
        ORIG    BUF0+24
        CON     BUF1+10
        ORIG    BUF1+24
        CON     BUF0+10
        END     START
```

3.1.2. p.mls

First we translate the mixal programm to binary code. This is done with `mixasm` (included in the GNU library `mdk`). Add the option `-l` to create the listing file `p.mls`.

Attention: Notice the output at line 48: "*** Startadresse: 1000". This might differ depending on the language set in your locales. It will become important in the next step!

```
mixasm p.mixal -l
cat p.mls
```

```
*** p.mixal: Kompilerzusammenfassung ***

-------------------------------------------------------------------------------
Src     Address  Compiled word           Symbolic rep
-------------------------------------------------------------------------------
009     01000   + 00 00 00 18 35    IOC 0,0(2:2)
010     01001   + 32 02 00 05 09    LD1 2050,0
011     01002   + 32 03 00 05 10    LD2 2051,0
012     01003   + 00 01 00 00 49    INC1    1,0
013     01004   + 54 43 01 05 26    ST2 3499,1
014     01005   + 15 56 00 01 41    J1Z 1016,0
015     01006   + 00 02 00 00 50    INC2    2,0
016     01007   + 00 02 00 02 51    ENT3    2,0
017     01008   + 00 00 00 02 48    ENTA    0,0
018     01009   + 00 00 02 02 55    ENTX    0,2
019     01010   + 46 55 03 05 04    DIV 2999,3
020     01011   + 15 46 00 01 47    JXZ 1006,0
021     01012   + 46 55 03 05 56    CMPA    2999,3
022     01013   + 00 01 00 00 51    INC3    1,0
023     01014   + 15 48 00 06 39    JG  1008,0
024     01015   + 15 43 00 00 39    JMP 1003,0
025     01016   + 31 11 00 18 37    OUT 1995,0(2:2)
026     01017   + 31 51 00 02 52    ENT4    2035,0
027     01018   - 00 50 00 02 53    ENT5    -50,0
028     01019   + 07 53 00 00 53    INC5    501,0
029     01020   + 46 55 05 05 08    LDA 2999,5
030     01021   + 00 00 00 01 05    CHAR    0,0
031     01022   + 00 00 04 12 31    STX 0,4(1:4)
032     01023   + 00 01 00 01 52    DEC4    1,0
033     01024   + 00 50 00 01 53    DEC5    50,0
034     01025   + 15 60 00 02 45    J5P 1020,0
035     01026   + 00 00 04 18 37    OUT 0,4(2:2)
036     01027   + 00 24 04 05 12    LD4 24,4
037     01028   + 15 59 00 00 45    J5N 1019,0
038     01029   + 00 00 00 02 05    HLT 0,0
043     01995   + 06 09 19 22 23    ALF "FIRST"
044     01996   + 00 06 09 25 05    ALF "␣FIVE"
045     01997   + 00 08 24 15 04    ALF "␣HUND"
046     01998   + 19 05 04 00 17    ALF "RED␣P"
047     01999   + 19 09 14 05 22    ALF "RIMES"
049     02024   + 00 00 00 31 51    CON 2035
051     02049   + 00 00 00 31 26    CON 2010
000     02050   - 00 00 00 07 51    CON 1073742323
000     02051   + 00 00 00 00 03    CON 0003
041     03000   + 00 00 00 00 02    CON 0002
-------------------------------------------------------------------------------

*** Startadresse:   1000
*** Endadresse: 2050

*** Symboltabelle
```

```
PRINTER             :  18
START               :  1000
BUF0                :  2000
BUF1                :  2025
PRIME               :  2999
TITLE               :  1995
L                   :  500

*** Ende der Zusammenfassung ***
```

3.1.3. p.card

Next we must write the binary code onto punched cards. This can be done with the python script `mixal/tools/mls2card.py`. The python scripts reads the listing file `p.mls`, extracts the code and writes it in the file `p.card`. Every line of `p.card` holds 80 chars of a card. The first to cards contain the bootloader discussed in exercise 26 in chapter 1.3.1 of TAOCP (see p. 510)[5]. The last cards is the so called transfer card, which tells the bootloader to start execution at memory location 1000.

Attention: You might miss the transfer card because your output has no "*** Startadresse ...". In this case edit the python script `mls2card.py` and adjust to the apropriate String found in your `*.mls` output.

```
../../tools/mls2card.py < p.mls > p.card
cat p.card
```

```
 O O6 Z O6    I C O4 0 EH A  F F CF 0  E   EU 0 IH G BB   EJ  CA. Z EU   EH E BA
   EU 2A–H S BB  C U 1AEH 2AEN V  E  ABG  CLU Z EH E BB J B. A  9
0000271000000000118705373955290537657674000026219309172462980266338409000052433 8
0000371007000052446700000001760000008375078618246802637169750786182520000026219 5
0000471014026424157502629304710522978469053346322000131073 8J013133419707861906 4
0000571021000000006900000171830000262260001310731702673870530000017573000630817 2
0000621028026712478100000001330000000000000000000000000000000000000000000000000 0
0000751995010310184700016113330002196420032009422503211840860000000000000000000 0
0000812024000000203500000000000000000000000000000000000000000000000000000000000 0
0000932049000000201000000049R000000003000000000000000000000000000000000000000000 0
0000013000000000000200000000000000000000000000000000000000000000000000000000000 0
TRANS01000000000000000000000000000000000000000000000000000000000000000000000000 0
```

3.2. How to send the cards to MIX

3.2.1. go button

Power MIX with USB cable connected to your computer. Start a screen session with 115200 baud (8N1)

```
screen /dev/ttyUSB0 115200
```

Press the *Go button* on MIX. You should see the welcome message on your terminal:

[5] notice the `J` in card 2!

```
WELCOME TO MIX. 1U = 40NS. U19 @115200 BAUD (8N1).
```

3.2.2. input the cards to U16

You can now send the punched cards to MIX within the screen terminal session. Read cards into a screen-buffer (called p) and send the buffer to MIX.

```
<in screen terminal> Ctr–a : readreg p p.card <enter>
<in screen terminal> Ctrl–a : paste p <enter>
```

After a short amount of time MIX spits out the following table to the printer U18. Notice that output to U18 have 120 characters per line, while output to the terminal U19 is only 70 chars wide.

```
WELCOME TO MIX. 1U = 30NS. U19 @115200 BAUD (8N1).
FIRST FIVE HUNDRED PRIMES
     0002 0233 0547 0877 1229 1597 1993 2371 2749 3187
     0003 0239 0557 0881 1231 1601 1997 2377 2753 3191
     0005 0241 0563 0883 1237 1607 1999 2381 2767 3203
     0007 0251 0569 0887 1249 1609 2003 2383 2777 3209
     0011 0257 0571 0907 1259 1613 2011 2389 2789 3217
     0013 0263 0577 0911 1277 1619 2017 2393 2791 3221
     0017 0269 0587 0919 1279 1621 2027 2399 2797 3229
     0019 0271 0593 0929 1283 1627 2029 2411 2801 3251
     0023 0277 0599 0937 1289 1637 2039 2417 2803 3253
     0029 0281 0601 0941 1291 1657 2053 2423 2819 3257
     0031 0283 0607 0947 1297 1663 2063 2437 2833 3259
     0037 0293 0613 0953 1301 1667 2069 2441 2837 3271
     0041 0307 0617 0967 1303 1669 2081 2447 2843 3299
     0043 0311 0619 0971 1307 1693 2083 2459 2851 3301
     0047 0313 0631 0977 1319 1697 2087 2467 2857 3307
     0053 0317 0641 0983 1321 1699 2089 2473 2861 3313
     0059 0331 0643 0991 1327 1709 2099 2477 2879 3319
     0061 0337 0647 0997 1361 1721 2111 2503 2887 3323
     0067 0347 0653 1009 1367 1723 2113 2521 2897 3329
     0071 0349 0659 1013 1373 1733 2129 2531 2903 3331
     0073 0353 0661 1019 1381 1741 2131 2539 2909 3343
     0079 0359 0673 1021 1399 1747 2137 2543 2917 3347
     0083 0367 0677 1031 1409 1753 2141 2549 2927 3359
     0089 0373 0683 1033 1423 1759 2143 2551 2939 3361
     0097 0379 0691 1039 1427 1777 2153 2557 2953 3371
     0101 0383 0701 1049 1429 1783 2161 2579 2957 3373
     0103 0389 0709 1051 1433 1787 2179 2591 2963 3389
     0107 0397 0719 1061 1439 1789 2203 2593 2969 3391
     0109 0401 0727 1063 1447 1801 2207 2609 2971 3407
     0113 0409 0733 1069 1451 1811 2213 2617 2999 3413
     0127 0419 0739 1087 1453 1823 2221 2621 3001 3433
     0131 0421 0743 1091 1459 1831 2237 2633 3011 3449
     0137 0431 0751 1093 1471 1847 2239 2647 3019 3457
     0139 0433 0757 1097 1481 1861 2243 2657 3023 3461
     0149 0439 0761 1103 1483 1867 2251 2659 3037 3463
     0151 0443 0769 1109 1487 1871 2267 2663 3041 3467
     0157 0449 0773 1117 1489 1873 2269 2671 3049 3469
```

```
    0163 0457 0787 1123 1493 1877 2273 2677 3061 3491
    0167 0461 0797 1129 1499 1879 2281 2683 3067 3499
    0173 0463 0809 1151 1511 1889 2287 2687 3079 3511
    0179 0467 0811 1153 1523 1901 2293 2689 3083 3517
    0181 0479 0821 1163 1531 1907 2297 2693 3089 3527
    0191 0487 0823 1171 1543 1913 2309 2699 3109 3529
    0193 0491 0827 1181 1549 1931 2311 2707 3119 3533
    0197 0499 0829 1187 1553 1933 2333 2711 3121 3539
    0199 0503 0839 1193 1559 1949 2339 2713 3137 3541
    0211 0509 0853 1201 1567 1951 2341 2719 3163 3547
    0223 0521 0857 1213 1571 1973 2347 2729 3167 3557
    0227 0523 0859 1217 1579 1979 2351 2731 3169 3559
    0229 0541 0863 1223 1583 1987 2357 2741 3181 3571
```

3.3. Congratulation

You have run your first program on a *real* MIX. Now trun the program `e` to calculate the easter dates.

4. Build your own MIX and/or modify the fpga design

In this section we will show how to build a MIX based on the fpga board iCE40HX8K-EVB. The whole project uses only FOSS soft- and hardware. With little modifications it should run on any fpga board available out in the wild.

4.1. Requirement

1. fpga board (iCE40HX8K-EVB)
2. programmer device (Olimexino 32u4) with idc10 cable (cable-IDC10)
3. USB-UART-board (BB-CH340T).
4. fpga toolchain. The project was developed with `apio` [6], a software suite based on project icestorm[7] from Clifford Wolf.

Consider to buy at Olimex Ltd. [8], the company with the highest number of registered OSHW-projects. Install the tools with:

```
sudo apt install mdk
sudo pip install -U apio
apio install -a
cd tools/iceprogduino
make
sudo make install
```

[6] github.com/FPGAwars/apio
[7] www.clifford.at/icestorm
[8] www.olimex.com

4.2. build and flash the fpga design

Cd into the directory `build/rtl` and build the project. Connect the fpga board with olimexino-32u4 programmer to upload the bitstream file. Use two USB cables to power both boards.

```
cd build/rtl
apio clean
apio build -v
apio upload
```

4.3. The USB-serial adapter

To connect the USB-UART-board (BB-CH340T) with iCE40HX8K-EVB we first have to modify BB-CH340T according to Fig. 4: To use USB-Serial adapter as power source we must cut with a cutter knife the pcb track marked in GREEN and solder a bridge BLACK between the right most terminal (VDD) and the 5V pad. This will ensure that the right most terminal connector (red cable) get's 5Volt, which will be used to power the fpga board. But the UART signals (green and yellow cables) are still leveled to 3.3V, which corresponds to the in-/output signal level of fpga-connector GPIO1.

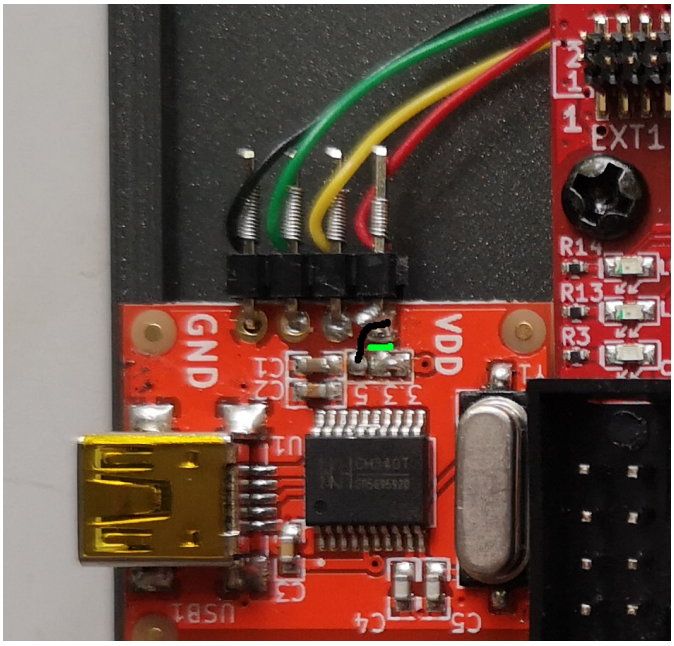


Figure 4: Cut the pcb track (GREEN) and solder a bridge from 5V pad to the right most connector pin.

The 4 wires are connected to the GPIO connector on the right side of iCE40HX8K-EVB according to the following table. Compare with schematic in the appendix A.

color	USB-serial	GPIO (ICE40HX8K-EVB)
black	GND	2
green	RX	7
yellow	TX	5
red	VDD	1

4.4. The Go button

We will implement the code needed by the go button proposed in exercise 26 in chapter 1.3.1 of TAOCP (see p. 510).

4.4.1. prepare the software

The mixal program can be found in `go.mixal`. It starts at location 4000 (which is implemented in fpga but not used by MIX). The programm spits out the welcome message. The JMP instruction at memory cell 4095 will jmp to location 0 storing a +0000 in the J-Register, because beeing a binary version with 12 bit programmcounter the next execution address without the jmp instruction would equally yield 4095 + 1 = 0000.

```
1           ORIG    4000
2           JMP     START
3  WELCOME  ALF     "WELCO"
4           ALF     "ME␣TO"
5           ALF     "␣MIX."
6           ALF     "␣1U␣="
7           ALF     "␣40NS"
8           ALF     ".␣U16"
9           ALF     "−U20␣"
10          ALF     "TO␣UA"
11          ALF     "RT␣@1"
12          ALF     "15200"
13          ALF     "␣BAUD"
14          ALF     "␣(8N1"
15          ALF     ").␣␣␣"
16          ORIG    4091
17 START    OUT     WELCOME(19)
18          JBUS    *(19)
19 NEXT     IN      0(16)
20          JBUS    *(16)
21          JMP     0
22          END     START
```

First we compile the mixal programm to binary code with `mixasm`. Then we translate the listing file to a binary representation, which can be directly written to the fpga board.

```
mixasm go.mixal -l
../../tools/mls2bin.py < go.mls > go.bin
cat go.bin
```

The python scripts reads the listing file `go.mls`, extracts the code and writes it in the file `go.bin`.

```
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
...
...
00000000000000000000000000000000
01111101000010000000010011100101
01111111111000000000010011100010
00000000000000000000010000100100
01111111111100000000010000100010
00000000000000000000000000100111
```

The output contain the program code expressed as binary numbers. These binary numbers can be flashed to the iCE40HX8K-EVB board, so it will be stored permanently in the MIX computer. At every reset (press *Go button*) the code will be executed. The first 4000 zero lines translate to NOP instructions. At the end you find the sequence IN(16),JBUS,JMP...

4.5. rebuild and flash to iCE40HX8K-EVB

- Copy the binary file into the directory `rtl`, where the fpga description files are.
- Rebuild the fpga project and upload. An `apio clean` is needed, because otherwise the preloaded memory will not be updated.

```
cp go.bin ../../rtl/go.bin
apio clean
apio build -v
apio upload
```

Tipp: change the welcome message to ensure the new rom file has been uploaded.

5. Example program t

We will run programm `t` of exercise 20 in chapter 1.3.2 TAOCP (p. 161) on MIX. Programm t controls the traffic signal at corner of Del Mare Boulevard and Berkeley Avenue. This project will connect LEDs directly to the register rX and a push button to the Overflow toggle. This will be done extending the fpga design and routing the appropriate signals to the GPIO connector at the back of MIX.

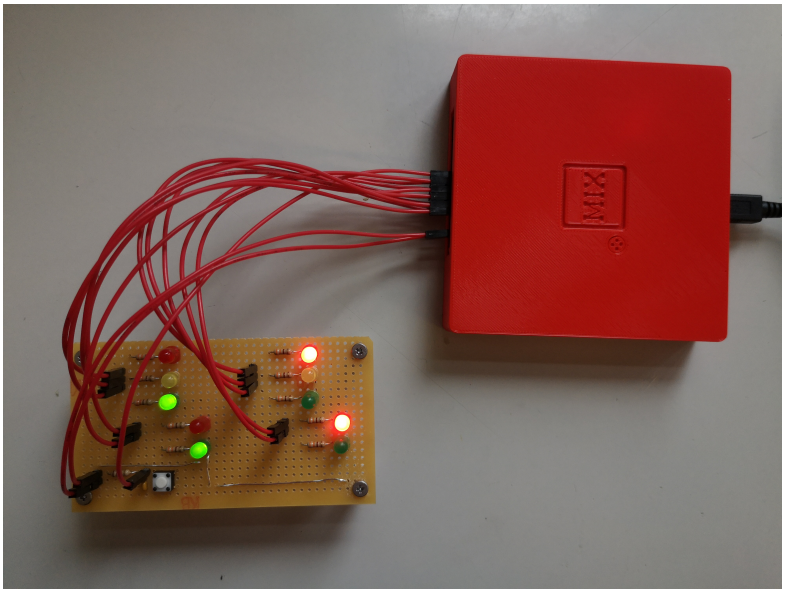


Figure 5: Connect LEDs and push button over the GPIO connector directly to register rX to control the traffic light

5.1. Extending the fpga desing

Make a copy of the folder `rtl` and cd into it.

```
cd build
cp -r rtl rtl_traffic_light
cd rtl_traffic_light
```

5.1.1. mix.pcf

Find the following lines in the physical constraint file `mix.pcf` and uncomment.

```
#at the GPIO connector you find
set_io rx   E4      # GPIO 5
set_io tx   B2      # GPIO 7

set_io dmred    F5  # GPIO 9
set_io bred     J2  # GPIO 10
set_io dmamber  B1  # GPIO 11
set_io bamber   H1  # GPIO 12
set_io dmgreen  C1  # GPIO 13
set_io bgreen   G1  # GPIO 14
set_io dmdw     C2  # GPIO 15
set_io bdw      J5  # GPIO 16
set_io dmw      F4  # GPIO 17
set_io bw       H2  # GPIO 18
set_io button   D2  # GPIO 19
```

5.1.2. mix.v

To connect the Register rX with the traffic signals: find the following lines in `mix.v` and uncomment:

```
// MIX - 1009
// Don Knuths computer architecture described in "The Art of Computer Programming"

`default_nettype none
module mix(
    input wire clk_in,
    input wire rx,
    output wire tx,
    output wire hlt,
    input wire button,
    output wire dmgreen,
    output wire dmamber,
    output wire dmred,
    output wire bgreen,
    output wire bamber,
    output wire bred,
    output wire dmw,
    output wire dmdw,
    output wire bw,
    output wire bdw
);

assign dmgreen = RegisterX[19:18] == 2'd1;
assign dmamber = RegisterX[19:18] == 2'd2;
assign dmred = RegisterX[19:18] == 2'd3;
assign bgreen = RegisterX[13:12] == 2'd1;
assign bamber = RegisterX[13:12] == 2'd2;
assign bred = RegisterX[13:12] == 2'd3;
assign dmw = RegisterX[7:6] == 2'd1;
assign dmdw = RegisterX[7:6] == 2'd2;
assign bw = RegisterX[1:0] == 2'd1;
assign bdw = RegisterX[1:0] == 2'd2;
```

Control the overflow toggle with the button by uncommenting line 7 in the code snippet:

```
reg overflow;
reg less;
reg equal;
reg greater;
always @(posedge clk)
    if (reset) overflow <= 0;
    else if (button) overflow <= 1;     #traffic signal button
    else if (add2) overflow <= addof;
    else if (sub2) overflow <= subof;
    else if (ide) overflow <= (rA|rX)? ideout[30] : ideout[12];
```

5.2. rebuild and flash to iCE40HX8K-EVB

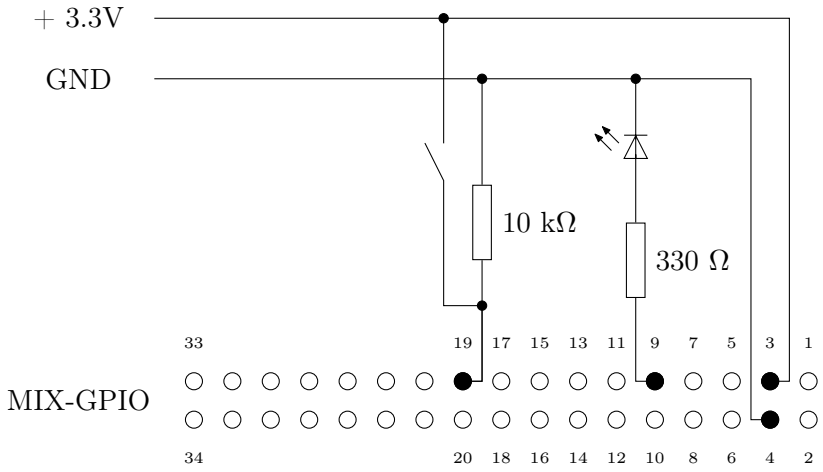
Rebuild the fpga project and upload.

```
apio clean
apio upload -v
```

Tipp: change the welcome message to ensure the new rom file has been uploaded.

5.3. add leds and button to GPIO

Connect leds and button (don't forget resistors) to the appropriate GPIO connectors as described in the `mix.pcf` file. For simplicity only one LED is shown below:



Attention: gpio pins 1,2,5 and 7 are already used by the internal USB-serial converter.

5.4. t.mixal

Compile and write `t.mixal` to puching card format. Start MIX by pressing the *go button*. Upload `t.card` to MIX and see the traffic signals blinking.

6. Service

In case you encounter an issue with MIX:

- don't panic
- plese send an email to the author with a precise description of the problem.

7. License

7.1. mix-fpga

The project as a whole is licensed under the GPL 3 and can be found at www.gitlab.com/x653/mix-fpga.

7.2. Manual

A. Schematic iCE40HX8K-EVB