MIX-fpga

Michael Schröder
mi.schroeder@netcologne.de

July 13, 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].

1.1. A look inside

The MIX computer is composed of two little boards:

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

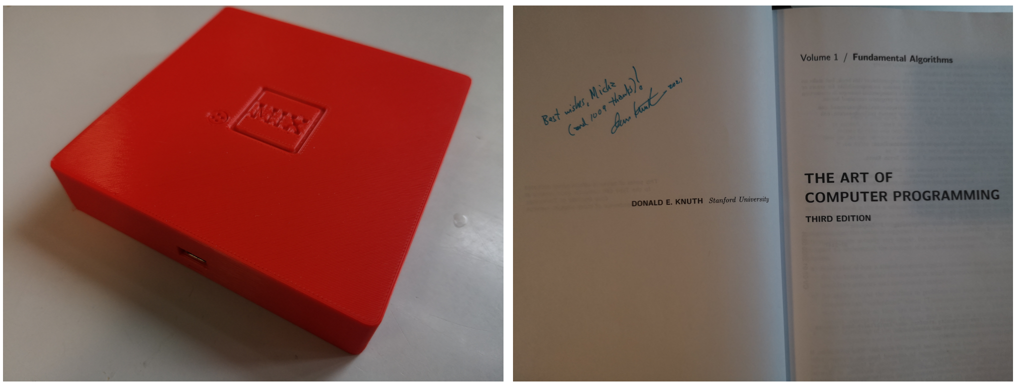


Figure 1: A real MIX for Don Knuth

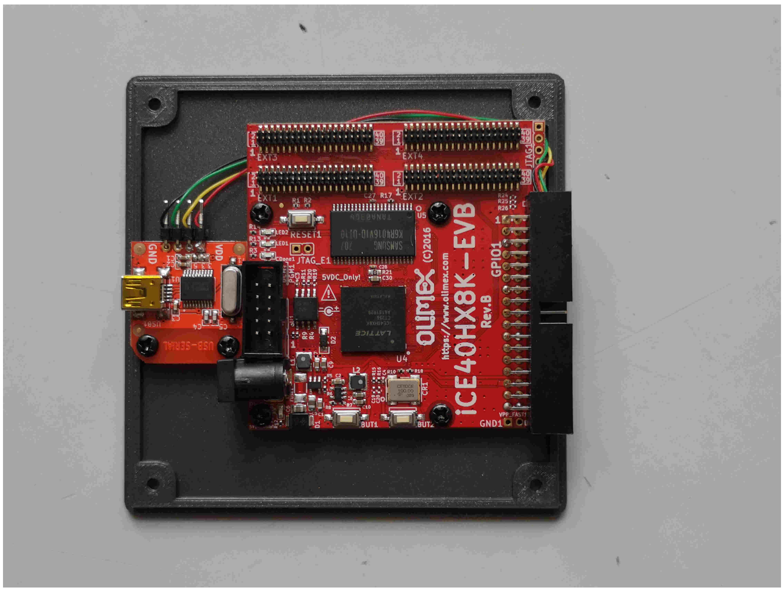


Figure 2: The two boards interconnected with wire wrap technique typical for computers of the 60s era

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.

1.1.1. Basic unit of time

MIX runs on iCE40HX8K-EVB clocked at 23.8 MHz. The basic unit of time u corresponds to $1u = 42\,\mathrm{ns}$, so according to Knuth it's a relatively high priced machine.

1.1.2. Character based I/O units U16-U20

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).

1.1.3. The block based I/O unit U0

A block based device is implemented on I/O unit 0. The device acts as a tape with 1024 blocks à 100 words. The data is stored in the SRAM chip included on the iCE40HX8K-

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

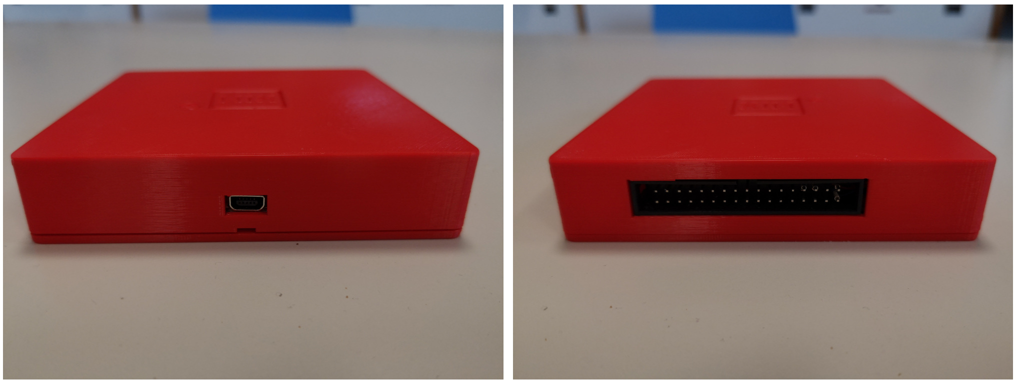


Figure 3: All character based I/O is done through a USB to serial adapter @115200 baud (8N1). MIX can be expanded through a GPIO connector placed at the rear side

EVB board. The speed of read and write operations is exactly $401u$ for reading or writing one complete block of data (no `JBUS` is needed). The data stored to unit U0 can be retrieved also after a reset (GO-Button). But on a full shutdown, when removing power supply (USB connector) data is lost.

1.1.4. MIX commands

All commands described in TAOCP Vol. 1 are implemented (s. table 1.1.4) 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 modulo 10 computation are executed in the defined timing of two cycles.

The system can (easily) be extended in various ways:

1. add more commands:
   - logic operators: `AND`, `OR`, `XOR`, `NOT`
   - `JrE`,`JrO` jump if Register r is even/odd (done)
   - Floating point arithmetic: `FADD`, `FSUB`, `FMUL`, `FDIV` (done)
   - Floating point compare: `FCMP` (to do)
2. add more hardware:
   - add leds to run the traffic light example (s. chapter 5 of this manual)
   - add more I/O unit

1.1.5. 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.

Table 1: MIX commands

Command	OP	Field	Timing
`NOP`	0	0	$1u$
`ADD`, `SUB`, `MUL`, `DIV`	1, 2, 3, 4	0:5	$2u$, $2u$, $10u$, $12u$
`FADD`, `FSUB`, `FMUL`, `FDIV`	1, 2, 3, 4	6	$4u$, $4u$, $9u$, $11u$
`NUM`, `CHAR`	5	0, 1	$10u$
`HLT`	5	2	$1u$ or ∞?
`SLA`, `SRA`, `SLAX`, `SRAX`, `SLC`, `SRC`	6	0—5	$2u$
`MOVE`	7	F	$(1+2F)u$
`LDr`, `LDNr`	16—23	0:5	$2u$
`STr`	24—31	0:5	$2u$
`STJ`	32	0:2	$2u$
`STZ`	33	0:5	$2u$
`JBUS`, `JRED`	34,38	U	$1u$
`IOC`	35	U	$1u$
`IN`, `OUT`	36,37	U	$(1+T)u$
`JMP`, `JSJ`, `JOV`, `JNOV`, `JL`, `JE`, `JG`, `JGE`, `JNE`, `JLE`	39	0—9	$1u$
`JrN`, `JrZ`, `JrP`, `JrNN`, `JrNZ`, `JrNP`, `JrE`, `JrO`	40-47	0—7	$1u$
`INCr`, `DECr`, `ENTr`, `ENNr`	48—53	0—3	$1u$
`CMPr`	54—63	0:5	$2u$

1.1.6. 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. tools

To run mixal programs on your MIX the following tools found in `mixal/tools` [5] are needed:

- `asm.py`: translates the mixal program into numeric representation (`.mls`).
- `mls2card.py`: translate the machine code (`.mls`) to punchcard format. The first two cards beeing the loading routine (1.3.1.ex26) and the last card beeing the transfer card.

[5] www.gitlab.com/x653/mix-fpga

- `mls2char.py`: translate to character codes, which can directly be read by MIX. (This is only necessary for the card-loading routine written on the first two punch cards).
- `mls2bin.py`: translate the `.mls` file to binary. This is only necessary for the GO button, which is hardcoded into fpga ROM.

2.2. V1: upload and run in a screen session

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. You should see the MIX welcome message:

```
WELCOME TO MIX. 1U = 40NS. U16–U20 TO UART @115200 BAUD (8N1).
```

If `screen` does not open the terminal try with `sudo screen`. This might happen, when the user is not allowed to acces the device `/dev/ttyUSB0`. Check the user setting of `/dev/ttyUSB0` and add the user to the dialout group `sudo usermod –a –G $USER dialout`.

Within the screen session you can upload the mixal programs written on punch cards:

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

2.3. V2: Upload and run using the Makefile

As an alterntive to `screen` you can send jobs to MIX using a `Makefile`. Every tested program comes in a subfolder with a corresponding `Makefile`, which can be used to compile, upload, and run the program on MIX.

- Connect MIX to USB
- Press the GO button
- run the Makefile
- the output is stored in a `.out` file

```
$ make clean
$ make
```

2.4. Verify

MIX has been verified with the following programms. The numbering correspond to the sections in TAOCP.

- `1.3.1ex26` card-loading routine

- `1.3.2P` table of primes
- `1.3.2E` easter dates
- `1.3.2.ex13` cryptanalyst problem (classified)
- `1.3.2.ex16` sum of harmonic series
- `1.3.2.ex20` josephus problem
- `1.3.2.ex22` traffic signal problem (driving real LEDs)
- `1.3.3A` multiply permutations in cycle form
- `1.4.2` character input routine
- `1.4.3` trace routine (uses tape U0)

3. Example program 1.3.2P

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/1.3.2P/p.mixal`:

```
$ cd mixal/1.3.2P
$ cat p.mixal
```

```
* EXAMPLE PROGRAM ... TABLE OF PRIMES
*
L          EQU  500
PRINTER    EQU  18
PRIME      EQU  -1
BUF0       EQU  2000
BUF1       EQU  BUF0+25
           ORIG 3000
START      IOC  0(PRINTER)
           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 the translator `tools/asm.py`.

```
$ ../tools/asm.py p.mixal
$ cat p.mls
```

```
                     0001 * EXAMPLE PROGRAM ... TABLE OF PRIMES
                     0002 *
                     0003 L          EQU  500
                     0004 PRINTER    EQU  18
                     0005 PRIME      EQU  -1
                     0006 BUF0       EQU  2000
                     0007 BUF1       EQU  BUF0+25
                     0008            ORIG 3000
3000 + 0000 00 18 35 0009 START      IOC  0(PRINTER)
3001 + 2050 00 05 09 0010            LD1  =1-L=
3002 + 2051 00 05 10 0011            LD2  =3=
3003 + 0001 00 00 49 0012 2H@0012    INC1 1
```

```
3004 + 0499 01 05 26 0013            ST2  PRIME+L,1
3005 + 3016 00 01 41 0014            J1Z  2F
3006 + 0002 00 00 50 0015 4H@0015    INC2 2
3007 + 0002 00 02 51 0016            ENT3 2
3008 + 0000 00 02 48 0017 6H@0017    ENTA 0
3009 + 0000 02 02 55 0018            ENTX 0,2
3010 - 0001 03 05 04 0019            DIV  PRIME,3
3011 + 3006 00 01 47 0020            JXZ  4B
3012 - 0001 03 05 56 0021            CMPA PRIME,3
3013 + 0001 00 00 51 0022            INC3 1
3014 + 3008 00 06 39 0023            JG   6B
3015 + 3003 00 00 39 0024            JMP  2B
3016 + 1995 00 18 37 0025 2H@0025    OUT  TITLE(PRINTER)
3017 + 2035 00 02 52 0026            ENT4 BUF1+10
3018 - 0050 00 02 53 0027            ENT5 -50
3019 + 0501 00 00 53 0028 2H@0028    INC5 L+1
3020 - 0001 05 05 08 0029 4H@0029    LDA  PRIME,5
3021 + 0000 00 01 05 0030            CHAR
3022 + 0000 04 12 31 0031            STX  0,4(1:4)
3023 + 0001 00 01 52 0032            DEC4 1
3024 + 0050 00 01 53 0033            DEC5 50
3025 + 3020 00 02 45 0034            J5P  4B
3026 + 0000 04 18 37 0035            OUT  0,4(PRINTER)
3027 + 0024 04 05 12 0036            LD4  24,4
3028 + 3019 00 00 45 0037            J5N  2B
3029 + 0000 00 02 05 0038            HLT
                        0039 * INITIAL CONTENTS OF TABLES AND BUFFERS
                        0040            ORIG PRIME+1
0000 + 0000 00 00 02 0041            CON  2
                        0042            ORIG BUF0-5
1995 + 0393 19 22 23 0043 TITLE      ALF  FIRST
1996 + 0006 09 25 05 0044            ALF   FIVE
1997 + 0008 24 15 04 0045            ALF   HUND
1998 + 1221 04 00 17 0046            ALF  RED P
1999 + 1225 14 05 22 0047            ALF  RIMES
                        0048            ORIG BUF0+24
2024 + 0000 00 31 51 0049            CON  BUF1+10
                        0050            ORIG BUF1+24
2049 + 0000 00 31 26 0051            CON  BUF0+10
2050 - 0000 00 07 51      =1-L       CON  499
2051 + 0000 00 00 03      =3         CON  3
                        0052            END  START
                                        TRANS 3000
                         * SYMBOL TABLE
                          *          EQU  2050
                          2H@0012    EQU  3003
                          2H@0025    EQU  3016
                          2H@0028    EQU  3019
                          4H@0015    EQU  3006
                          4H@0029    EQU  3020
                          6H@0017    EQU  3008
                          =1-L       EQU  2050
                          =3         EQU  2051
```

```
                BUF0      EQU  2000
                BUF1      EQU  2025
                L         EQU  500
                PRIME     EQU  −1
                PRINTER   EQU  18
                START     EQU  3000
                TITLE     EQU  1995
```

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 numeric codes and writes them in the file `p.card` in puncher card format.

Every line of `p.card` holds 80 chars of a card. The first two cards contain the card loading routine discussed in exercise 26 in chapter 1.3.1 of TAOCP. The last card is the so called transfer card, which tells the bootloader to start execution at memory location 3000.

```
$ ../tools/mls2card.py p.mls
$ 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  CLU  ABG Z EH E BB J B. A  9
     730000000001187053739552905376576740000262193013081429807906264090000524338
     73007000052446700000001760000008375000027475007880049750000274 80Q0000262195
     7301407885295750787218471052297846905334632200013107 38J0131334197000028295K
     730210000000006900000171830000262260001310731707916750530000017573000 6308172
     230280791412781000000013300000000000000000000000000000000000000000000000000
     100000000000002000000000000000000000000000000000000000000000000000000000000
     519950103101847000161133300021964200320094225032118408600000000000000000000
     120240000000203500000000000000000000000000000000000000000000000000000000000
     32049000000020100000000049R000000000300000000000000000000000000000000000000
TRANS030000000000000000000000000000000000000000000000000000000000000000000000000
```

3.1.4. go!

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:

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

3.1.5. 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.

```
FIRST FIVE HUNDRED PRIMES
     3000 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
```

3.2. Congratulation

You have run your first program on a *real* MIX.

3.3. Using Makefile

With the included `Makefile` the above described steps can be done in one pass:

- Connect MIX to USB
- Press the GO button
- run the Makefile: `$ make`
- the output is stored in `p.out` file

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

The project runs on iCE40HX8K-EVB fpga from Olimex Ltd. 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:

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

```
$ sudo pip install -U apio
$ apio install ice40
$ apio install iverilog
$ apio install scons
$ apio install yosys
$ sudo apt install gtkwave
$ cd build/iceprogduino
$ make
$ sudo make install
$ sudo apt install screen
```

4.2. FPGA iCE40HX8K-EVB

Build the project and upload the circuit design to the fpga:

1. cd into the directory `rtl` and build the project

   ```
   $ cd rtl
   $ apio clean
   $ apio build -v
   ```

2. Connect the fpga board with olimexino-32u4 programmer to upload the bitstream file. Use two USB cables to power both boards as shown in fig. 4.

3. Upload the bitstream file to fpga board.

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

4.3. The USB-serial adapter

The serial adapter BB-CH340T is used to:

1. Power MIX with 5Volt from USB cable
2. Serial communication to MIX over I/O Unit 16-20.

To use USB-Serial adapter as power source we must do a little modification on the board according to Fig. 5:

1. cut with a cutter knife the connection marked in GREEN, and
2. 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 form USB, 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.

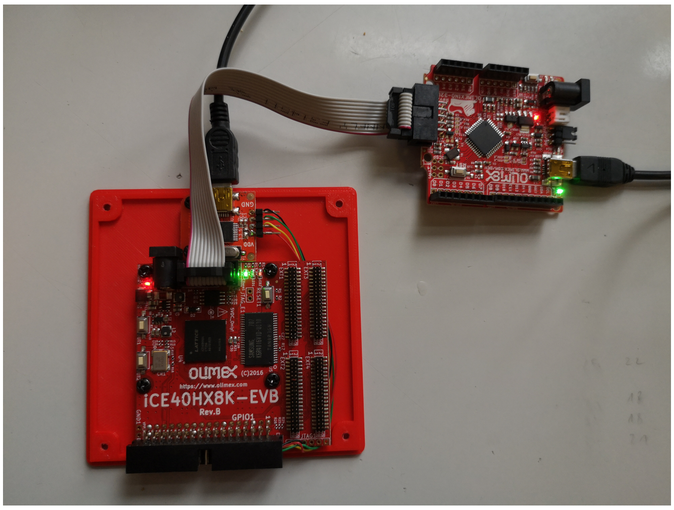


Figure 4: Connect programmer Olimexino-32u4 with fpga board iCE40HX8K-EVB

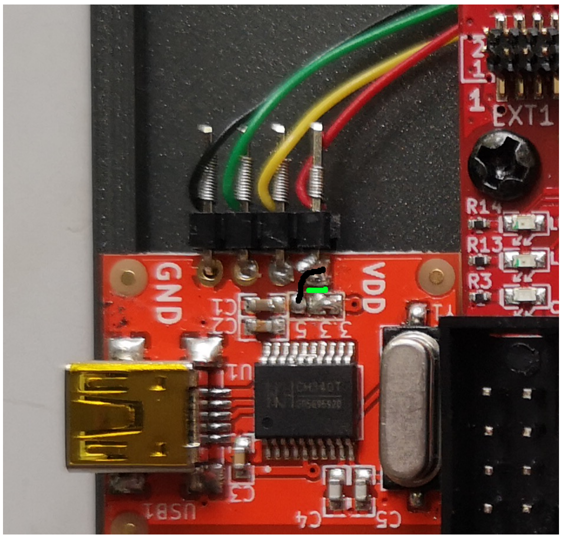


Figure 5: Cut the pcb track (GREEN) and solder a bridge from 5V pad to the right most connector pin.

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 B.

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/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.

```
            ORIG 4000
            JMP  START
WELCOME     ALF  WELCO
            ALF  ME TO
            ALF   MIX.
            ALF   1U =
            ALF   42NS
            ALF  . U16
            ALF  -U20
            ALF  USB-U
            ALF  ART.
            ALF  U0 10
            ALF  24 BL
            ALF  OCKS.
            ALF   FPU.
            ORIG 4091
START       OUT  WELCOME(17)
            JBUS *(17)
NEXT        IN   0(16)
            JBUS *(16)
            JMP  0
            END  START
```

First we translate the mixal programm to binary code with `mixal/tools/asm.py`.

```
$ cd build/go
$ ../../mixal/tools/asm.py go.mixal
$ cat go.mls
```

```
                     0001             ORIG 4000
4000 + 4091 00 00 39 0002             JMP  START
4001 + 1669 13 03 16 0003 WELCOME     ALF  WELCO
4002 + 0901 00 23 16 0004             ALF  ME TO
4003 + 0014 09 27 40 0005             ALF   MIX.
4004 + 0031 24 00 48 0006             ALF   1U =
4005 + 0034 32 15 22 0007             ALF   42NS
4006 + 2560 24 31 36 0008             ALF  . U16
4007 + 2904 32 30 00 0009             ALF  ␣U20
4008 + 1558 02 45 24 0010             ALF  USB␣U
4009 + 0083 23 40 00 0011             ALF  ART.
4010 + 1566 00 31 30 0012             ALF  U0 10
4011 + 2082 00 02 13 0013             ALF  24 BL
4012 + 1027 12 22 40 0014             ALF  OCKS.
4013 + 0006 17 24 40 0015             ALF   FPU.
                     0016             ORIG 4091
4091 + 4001 00 17 37 0017 START       OUT  WELCOME(17)
4092 + 4092 00 17 34 0018             JBUS *(17)
4093 + 0000 00 16 36 0019 NEXT        IN   0(16)
4094 + 4094 00 16 34 0020             JBUS *(16)
4095 + 0000 00 00 39 0021             JMP  0
                     0022             END  START
                                      TRANS 4091
                         * SYMBOL TABLE
                          *           EQU  4096
                          NEXT        EQU  4093
                          START       EQU  4091
                          WELCOME     EQU  4001
```

Next we must translate the binary code into a binary format readable by the fpga toolchain. This can be done with the python script `mixal/tools/mls2bin.py`.

```
$ ../../mixal/tools/mls2bin.py go.mls
$ cat go.bin
```

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

Inspect the binary file 'go.bin'

```
0000000000000000000000000000000
0000000000000000000000000000000
0111110100001000000010001100101
0111111111110000000010001100010
0000000000000000000010000100100
0111111111111000000010000100010
0000000000000000000000000100111
```

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...

Figure 6: MIX fits in nice toast like case

4.5. build and flash to iCE40HX8K-EVB

- Copy the binary file `go.bin` 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.

4.6. print toast case with a 3D printer

In the subfolder `build/toast` you find printing files for the case. The case consists of three parts:

- `bottom.blend`: the bottom part of the case
- `top.blend`: top part of case
- `go.blend`: go bottom

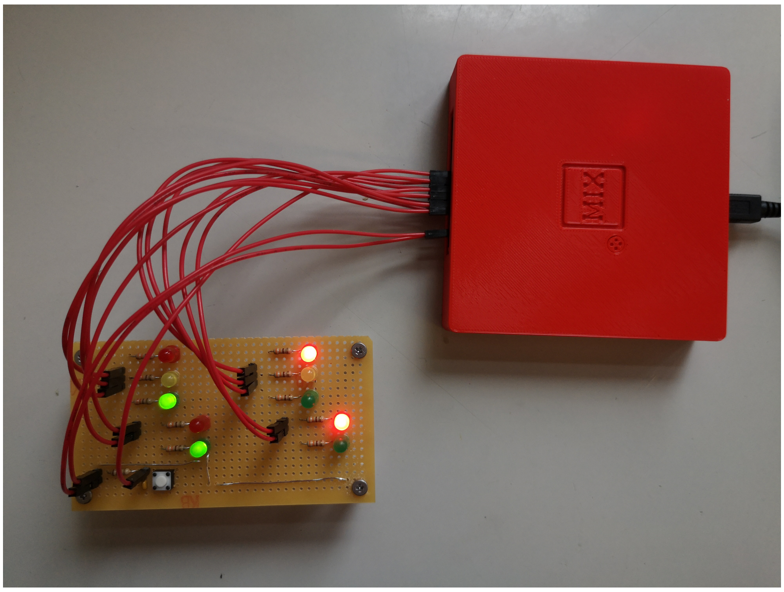


Figure 7: Connect LEDs and push button over the GPIO connector directly to register rX to control the traffic light

5. Example program 1.3.2.ex20T

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.

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 to get access to the GPIO pins 9-19.

```
#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 (28–38 and 47–56) in `mix.v` and uncomment:

```
20 // MIX
21 // Don Knuths computer architecture described in "The Art of Computer Programming"
22
23 `default_nettype none
24 module mix(
25     input wire clk_in,
26     output wire tx,
27     input wire rx,
28 //  output wire dmgreen,
29 //  output wire dmamber,
30 //  output wire dmred,
31 //  output wire bgreen,
32 //  output wire bamber,
33 //  output wire bred,
34 //  output wire dmw,
35 //  output wire dmdw,
36 //  output wire bw,
37 //  output wire bdw,
38 //  input wire button,
39     output wire hlt,
40     output [17:0] sram_addr,
41     inout [15:0] sram_data,
42     output sram_cen,
43     output sram_wen,
44     output sram_oen
45
46 );
47 //  assign dmgreen = RegisterX[19:18] == 2'd1;
48 //  assign dmamber = RegisterX[19:18] == 2'd2;
49 //  assign dmred = RegisterX[19:18] == 2'd3;
50 //  assign bgreen = RegisterX[13:12] == 2'd1;
51 //  assign bamber = RegisterX[13:12] == 2'd2;
52 //  assign bred = RegisterX[13:12] == 2'd3;
53 //  assign dmw = RegisterX[7:6] == 2'd1;
54 //  assign dmdw = RegisterX[7:6] == 2'd2;
55 //  assign bw = RegisterX[1:0] == 2'd1;
56 //  assign bdw = RegisterX[1:0] == 2'd2;
```

Control the overflow toggle with the button by uncommenting line 239 in the code snippet:

```
    always @(posedge clk)
        if (reset|clearof) overflow <= 0;
//      else if (button) overflow <= 1;     //the traffic signal button controls the overflow
        else if (add2 & addof) overflow <= 1;
        else if (div2 & divof) overflow <= 1;
        else if (ide & (rA|rX) & ideof) overflow <= 1;
        else if (fadd2 & faddof) overflow <= 1;
            else if (fmul2 & fmulof) overflow <=  1;
        else if (fdiv2 & fdivof) overflow <= 1;
```

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:

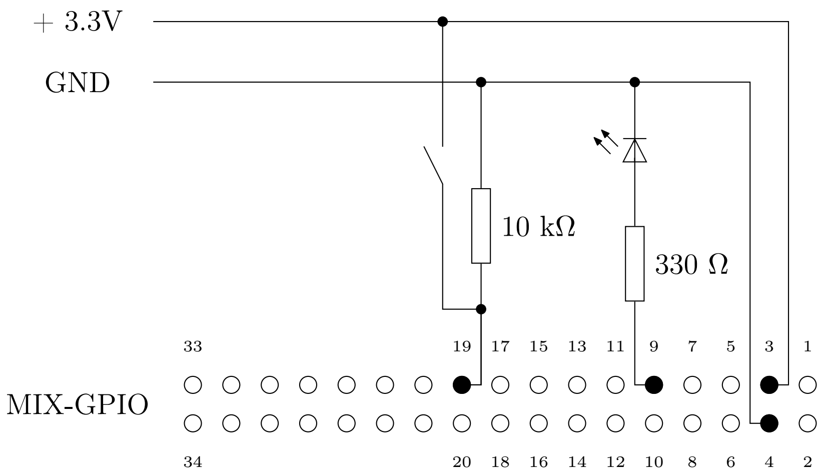


Figure 8: connect LED and button to GPIO connector

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

5.4. t.mixal

Compile and write `mixal/1.3.2.ex20/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[9] with a precise description of the problem.

7. License

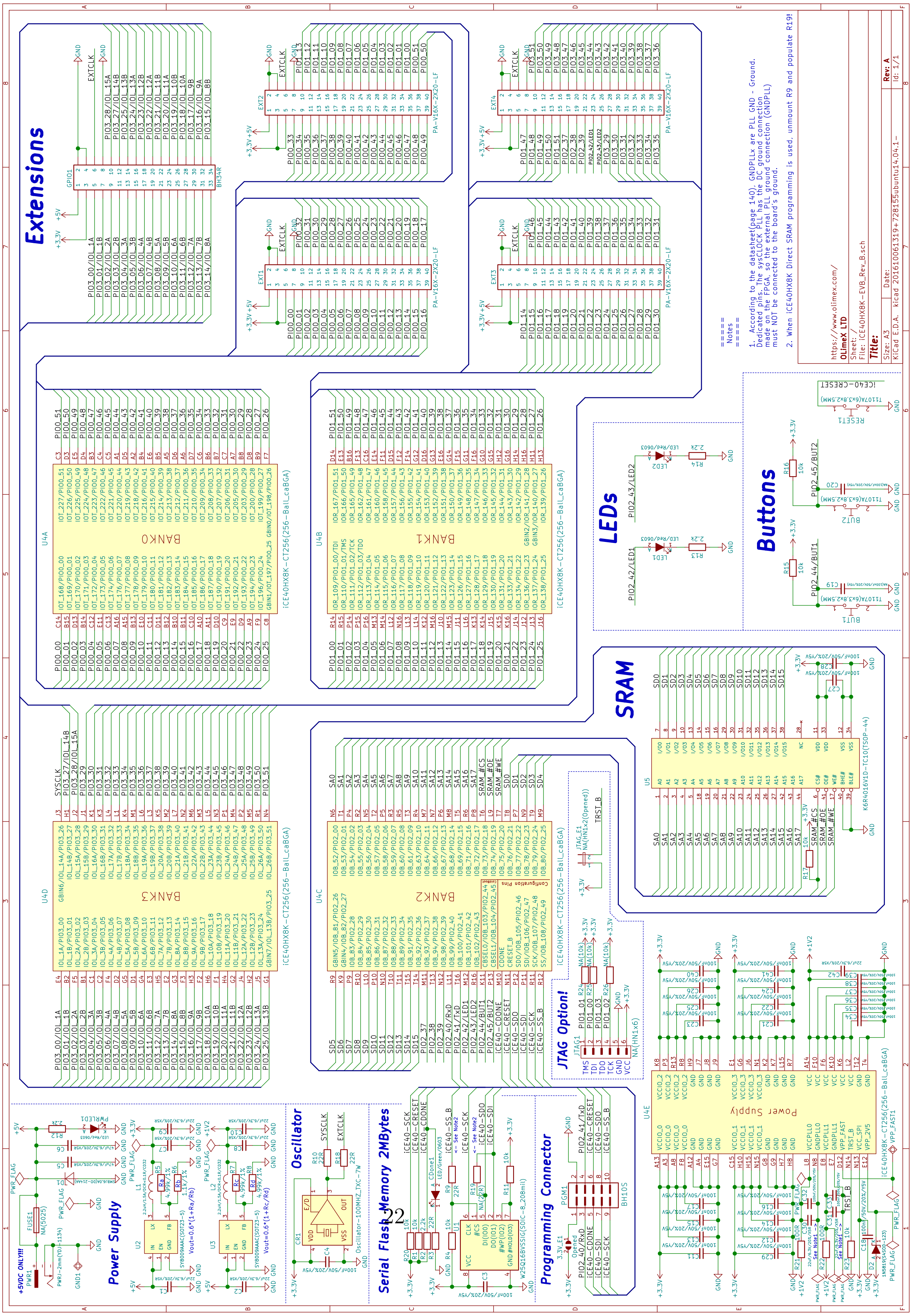
7.1. mix-fpga

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

[9] mi.schroeder@netcologne.de

A. Schematic iCE40HX8K-EVB



B. Schematic BB-CH340T