uTile Programmable Logic Controller
Application program m328-uTile

Instruction Manual

Revision 1
Date: 2021 October 8

Table of Contents

1 Introduction

This instruction manual provides information to set up and operate the uTile Programmable Logic Controller (PLC). The manual describes the uTile features, specifications and architecture.

Guidelines are presented for setting up a uTile system and preparing the programming environment for writing and testing PLC application programs. A terminal emulator program provides the user interface for communicating with uTile. A test rig design is presented which is suitable for simulating real-world inputs and outputs for uTile.

A major part of the manual is devoted to the programming of uTile applications. The uTile commands are described in detail and illustrated by numerous examples.

1.1 uTile brief description

uTile is a PLC application program written in assembly language for running on a Microchip/Atmel ATmega328P MCU device. The binary executable image of the m328-uTile application program (Intel hex file format) is programmed into the micro-controller unit (MCU) flash memory.

uTile runs on just about any controller board with an ATmega328P and an external 16 MHz clock crystal. The m328-uTile program is ported to run on the Arduino Nano or UNO boards.

uTile communicates with a terminal emulator program running on a host PC machine and connected via a USB cable between the PC and the controller board.

For the purposes of discussion this manual uses the Nano controller board as the platform for running uTile. The host PC is running a Fedora Linux operating system.

The Minicom communications program running on the host PC provides the VT102 terminal emulation for the PLC user interface. Refer to Section 11 for Minicom installation and setup details.

1.2 uTile highlights

The uTile PLC is a complete, ready-to-run controller that provides all the necessary functions for implementinging small automation projects. Major uTile highlights are listed below.

- Programmable Logic Controller application program for use on Arduino Nano controller board with an ATmega328P MCU.
- Fast and efficient bit stack architecture.
- Six general purpose digital inputs and eight digital outputs.
- Sixteen one-shot programmable timers.
- Sixteen Set/Reset flip-flops.
- Sixteen virtual storage bits.
- Complete set of control logic commands, AND, OR, XOR, invert etc.
- Integrated programming editor, no need for separate external program development software tools.
- Large user file space (UF0), up to 256 program lines.
- Store and Load user programs and delay timer settings to EEPROM.
- Read, Write and List user applications to host PC disc file.
- Autostart stored user program on power-up or board reset.
- Simple Minicom VT102 terminal emulator user interface.

2 Putting the uTile system together

This section describes how to assemble and run a uTile system on an Arduino Nano controller board. A new Nano board is factory initialised with a pre-installed bootloader and a Blinky program. Both of these programs are not needed and are overwritten during the uTile installation.

The m328-uTile program installation requires Minicom to be set up and running. The AVRdude package must be installed on the host PC. A USBasp ISP programmer and connecting cable is required to flash the binary image to the Nano board.

The general procedure for installing uTile on the Nano controller board is as follows:

1. Download the latest version m328-utile.hex binary image from the GitHub repository https://github.com/lynf/Arduino-PLC.

2. Using a USBasp ISP programmer for Atmel AVR controllers and the AVRdude program, flash the m328-utile.hex image to the ATmega328P device.

   The AVRdude command line entry for flashing the image is:

   avrdude -c usbasp -p atmega328p -P usb -u -U flash:w:m328-utile.hex:i

3. Using a USBasp programmer change the ATmega328P fuse bits in the low, high and extended bytes as follows:

   lfuse= 0xFF
   hfuse = 0xDF
   efuse = 0xFD

   The AVRdude command line entry for flashing the fuse bytes is:

   avrdude -c usbasp -p atmega328p -P usb -U lfuse:w:0xff:m -U hfuse:w:0xdf:m -U efuse:w:0xFD:m

The uTile program installation is now complete and the USBasp ISP programmer can be disconnected from the Nano board. The programming of the m328-utile.hex image is only done once.

2.1 Basic configuration

An Arduino Nano board programmed with the m328-uTile PLC application

requires connection to the Minicom communication program to access the user interface. The user interface allows you to program and execute all of the uTile commands.

The procedure for setting up a basic uTile operating environment is as follows:

1. Install and configure Minicom on the host PC.
2. Connect a USB cable between the PC and the Nano board. The board is powered by the 5 Vdc supplied over the USB cable.
3. Start Minicom. If Minicom is configured properly, uTile's main editor screen appears on the terminal. This completes the basic configuration for running the uTile PLC.

2.2 First time power-up

When power is first applied to the Nano board uTile's main editor screen appears on the terminal:

```
             <<<<  m328-uTile Programmable Logic Controller  >>>>
--------------------------------------------------------------------------------
         Bit Stack                                        Port Byte
       7 6 5 4 3 2 1 0             Data              7 6 5 4 3 2 1 0
       ===============             =====             ===============
       0 0 0 0 0 0 0 0             00000
--------------------------------------------------------------------------------

      000  .....        008  .....        016  .....        024  .....
      001  .....        009  .....        017  .....        025  .....
      002  .....        010  .....        018  .....        026  .....
      003  .....        011  .....        019  .....        027  .....
      004  .....        012  .....        020  .....        028  .....
      005  .....        013  .....        021  .....        029  .....
      006  .....        014  .....        022  .....        030  .....
      007  .....        015  .....        023  .....        031  .....



    Enter:
--------------------------------------------------------------------------------
 *** User File 0 ***
```

uTile is ready to accept user commands at the ‘Enter:’ prompt.

2.3 Integrated editor

The uTile integrated editor is accessed through the user interface that provides the terminal screen and keyboard functions. The Minicom serial communications program running on the host PC provides the user interface.

The uTile editor screen has four main screen sections, each separated by a dashed line.

1. Screen title
2. Display area showing:
   a) Bit stack display
   b) Data display
   c) Port Byte display
3. Main program display area and the ‘Enter:’ prompt
4. Status message line

The bit stack display is associated with uTile operation and it shows the current contents of the bit stack.

The data display shows numerical data entered for various purposes like delay timer settings or program line number target for the editor’s jump-to-line-number command.

The main program display area shows four columns of eight rows of program lines (a display page of 32 program lines) of the user file (UF0) space. There are eight display pages (0...7) covering the 256 program lines of the UF0 space.

The reverse-highlight line cursor shows the current selected program line number.

You can scroll around the UF0 space with the arrow keys, or jump to a specified program line in the UF0 program with the Home key.

2.3.1 Editor navigation keys

The Up, Down, Left and Right arrow keys and the Home key control the position of the program line cursor on the editor screen.

The Up arrow key moves the line cursor up to the previous line number. If the line cursor is at line 000, the Up arrow moves the line cursor on page 0 to line 255 on page 7.

The Down arrow key moves the cursor down to the next line number. If the line cursor is at line 255, the Down arrow moves the line cursor on page 7 to line 000 on page 0.

The Home key moves the line cursor from its current position to the new line number shown in the Data display. The target line number has to be entered at the ‘Enter:’ prompt before pressing the Home key.

2.3.2 Entering commands

Commands are entered at the ‘Enter:’ prompt. The editor is not case sensitive and commands can be entered using upper or lower case characters. A ‘CR’ entry marks the end of the command entry.

2.3.3 Entering numerical data values

The editor accepts decimal number inputs up to 5 digits in length (0 … 99999). When the editor processes the decimal number entry it caps the number to a maximum value of 65535. The reason for this capping has to do with the 16 bit binary word size of the AVR architecture. The maximum value of a 16 bit word is hexadecimal FFFF, or decimal 65355.

Several uTile commands expect numerical data for their inputs. These commands generally use byte size binary numbers (hexadecimal 00 … FF), or equivalent decimal numbers 0 … 255.

Numerical data for commands that expect byte size data, such as Timer settings and editor jump-to-line targets, are automatically capped to 255 maximum. For example, if you enter a number greater than 255 for a setting a Timer, the Timer will be set at 255.

3 Architecture

This section provides an introduction to uTile architecture. uTile is an acronym for “Micro Threaded Interpretive Logic Engine”.

3.1 Command interpreter

uTile command words are organised in a dictionary structure. The dictionary is comprised of a linked list of each uTile command word.

Each command word consists a namestring header, the address of the executable part of the command (the execution vector), followed by the binary code routine that performs the command’s function.

A uTile command entered at the editor ‘Enter:’ prompt is just the namestring part of the command word in the dictionary. Each time a command namestring is entered, the command interpreter processes the entry by searching for a matching namestring in the dictionary. The threaded (linked list) structure of the dictionary makes the search fast and efficient.

If the interpreter finds a matching command word namestring, the command’s routine is executed. If the search fails, the interpreter reports an error.

The interpreter examines the command and determines if it is an EDIT mode or a RUN mode type command. if it is a RUN mode type of command, the command’s execution vector is stored in the UF0 program space at the current program line number. If it is an EDIT mode command, no execution vector is stored in the UF0 space.

3.2 Input/Output ports

External devices attach to the Nano by wiring to the board I/O connections. The uTile I/O ports are mapped to the board’s I/O ports.

3.3 UF0 program space

uTile programs are entered in the 500 byte UF0 program space, located in RAM data space. UF0 space can storing up to 256 command words. One command in a UF0 program takes up 2 bytes of storage.

The UF0 and the timer settings spaces are stored in EEPROM with the ***store*** command. The stored EEPROM data is loaded back to the UF0 and timers settings RAM spaces with the ***load*** command.

3.4 uTile operating modes

The uTile operates in one of two modes, the EDIT mode or the RUN mode.

The EDIT mode accepts commands for:

- Emtering a new program into the UF0 program space
- Editing an existing program in the UF0 program space
- Entering uTile Housekeeping commands

In the RUN mode a UF0 program starts and runs in a continuous execution loop.

3.4.1 EDIT mode operation

uTile enters the EDIT mode immediately after a board power-up or a board reset.

When a command word is entered in the editor, the interpreter searches uTile's word dictionary to find the word. If the word is not found, an error message '*** Unrecognized Input! ***' is displayed on the editor's status line.

If the word is found, it is executed immediately. The interpreter determines if the word is a RUN mode type command or if it is an EDIT mode type command. For RUN mode type commands, the address of the executable code (the execution vector) is stored in the UF0 address space, then the word's namestring is printed in the UF0 page display

at the current program line cursor position. The cursor then advances to the next program line.

EDIT mode type commands are executed when entered, but their execution vectors are not stored in the UF0 space. The program line cursor does not advance to the next program line.

The incremental execution of each command as it is entered allows for a faster real-time program development process. The user gets immediate feedback on the action of each command as it is entered. Unlike other compile-load-execute PLC architectures, uTile greatly speeds up program development and debug cycles.

3.4.2 RUN mode operation

uTile enters the RUN mode from the EDIT mode by running the ***ex0*** or ***run*** commands.

In the RUN mode the user's program stored in the UF0 space is run repeatedly at full execution speed. The interpreter does not operate during the RUN mode operation.

3.4.3 Autostart run mode

uTile enters the autostart run mode following a board power-up or reset provided the enable Autostart control input on D8 pin is grounded and if a UF0 image is stored in EEPROM.

Autostart is enabled whenever stand-alone and un-attended PLC operation is required. In autostart mode uTile operates headless, without the need for a user interface terminal connection.

3.5 The bit stack

A central feature of the uTile architecture is the bit stack, or

BSTK, an 8 bit data structure with a last-in-first-out storage mechanism. Logic data bits are pushed to or popped from the ***BSTK***, depending on whether the data bits are source, destination or both source and destination type operands. Input bits are source operands, output bits are destination operands while virtual bits are both source and destination type operands.

Data from source operands such as input port bits are pushed onto the ***BSTK***. Data popped from the ***BSTK*** are written to destination operands such as output port bits.

All logic operations are performed on data stored on the ***BSTK*** and the results of the operations stored back on the ***BSTK***.

All data transfer operations between source and destination operands go via the ***BSTK***.

The bit stack consist of one byte of eight bits, namely bits 0 through 7. The figure below shows the bit stack, and bit position 0 is called the top-of-stack (TOS). Data bits enter and exit the stack (pushed and popped from the stack) via the TOS position. The stack is a last-in-first-out data structure.

position	BSTK bits	Comments
0	[TOS]	← Top-of-stack
1	[TOS+1]	
2	[TOS+2]	
3	[TOS+3]	
4	[TOS+4]	
5	[TOS+5]	
6	[TOS+6]	
7	[TOS+7]	← Bottom-of-stack

Data is pushed onto the top of the stack (TOS) by a logic operation that reads a data bit from a source operand. The push operation moves all bits in the stack down by one bit position; TOS moves to TOS+1, TOS+1 moves to TOS+2, and so on. TOS+7 is pushed off the end of the stack and is lost. The new data bit is then moved into the TOS position.

The source operand is specified by the operation that invokes the push. If you push more data onto a full ***BSTK***, the TOS+7 data is lost with every additional push to the full stack.

In general ***BSTK*** operations must be balanced so that all pushes are balanced by an equal number of pops.

Data is popped off the stack from the TOS to a destination operand. The pop operation moves all bits in the stack up by one bit position; TOS is popped and written to the destination operand specified by the command that invoked the pop. TOS+1 moves to TOS, TOS+2 moves to TOS+1, and so on. A zero bit moves into TOS+7.

The Bit Stack display shows the ***BSTK*** contents as commands are entered and executed in the EDIT mode. The Bit Stack display is not active in the RUN operating mode.

Logic operations ***AND***, ***OR*** and ***XOR*** pop TOS and TOS+1 bits from the ***BSTK***, performs the operation, then pushes the result back to TO***S***. The operations are performed in “Reverse Polish” notation (RPN) as opposed to the more widely known “Algebraic” notation.

In RPN notation, the two operands are first stated, then the operation to be performed is specified.

In Algebraic notation The first operand is stated, the operation is specified, then the second operand is stated. A program using the RPN notation turns out to be more efficient compared an equivalent program using the Algebraic notation.

The unary logic complement ***!*** operation operates on only one operand, the TOS bit. The TOS bit is popped from the ***BSTK***, complemented, and then pushed back to the ***BSTK***.

3.5.1 Ports and bit labels

Several registers, data bytes and specific bits in the ATmega328P MCU are used by uTile. Labels are assigned to these resources so they can be referenced in a simplified and abstract manner by the uTile virtual machine.

uTile interfaces to the external world through input and output (I/O) ports. The input port is labeled ***PA*** while the output port is labeled ***PY***. Individual bits in ***PA*** and ***PY*** are also assigned specific bit labels.

The uTile port and bit labels are shown in the table below:

Output Port	***PY***	***Y7...Y0***
Input Port	***PA***	***A5...A0***
Virtual Ports:	***PU*** ***PV***	***U7...U0*** ***V7...V0***
Timers: Trigger input: Reset input: Timer output:	***Tm*** ***TK*** ***TR*** ***TQ***	 ***TKf...TK0*** ***TRf...TR0*** ***TQf...TQ0***
Flip-Flops Set input: Reset input : Output:	 ***S*** ***R*** ***Q***	 ***Sf...S0*** ***Rf...R0*** ***Qf...Q0***
Counter: Gate input:	 ***CNT***	 ***CNT***
Toggle Gate input: Output:	 ***TG*** ***TGQ***	 ***TG0...TG7*** ***TGQ0...TGQ7***
Clock signals:		***CLKA, CLKB, CLKC***

Table 1 - uTile Port & Bit Labels

3.5.2 The '.' command noation

uTile bit stack operations involve the bi-directional transfer of data between the bit stack ***BSTK*** and a uTile source or destination operand. The ***BSTK*** is implicit in every data transfer operation.

uTile commands use a '***.***' symbol as a prefix or postfix to a bit or byte label to signify if the ***BSTK*** is acting as a source or destination for the transfer operation.

A bit label with a postfix '***.***' indicates the bit stack is the destination of the data transfer. For example, the command ***A0.*** transfers the port bit ***A0*** data to the TOS.

A prefix '***.***' with a bit label indicates the ***TOS*** is the source for the data transfer operation. For example, the command ***.Y0*** writes the TOS to the ***Y0*** port bit.

3.6 Input/Output ports

uTile has six digital inputs and eight digital outputs. The input/output (I/O) ports on the ATmega328P port pins are wired to the connection terminals on the Nano board.

Input port pins are scanned regularly and pre-processed before they are used in uTile for logic processing. All input signals are de-bounced to remove any noise caused by a rapidly changing signal from a bouncing mechanical contact. Important characteristics of the input signal, such as leading edge and trailing edge transitions, are captured and saved by the input pre-processor and are accessible by bit labels in the user program .

The input port pins are held high by pull-up resistors internal to the ATmega328P chip when the port pins are configured as inputs. Input port pins source current into any device connected to the pin. The input port pin characteristics with pull-up resistor activated are specified in the ATmega data sheet in the section on Pin Pull-Up.

Open circuit input port pins are read as a logic 1 level because of the input pull-up resistors.

Processed Input port data is stored in port byte ***PA***. Individual bits of ***PA*** are accessed by their specific bit labels.

Output port pins are updated regularly from information stored in the output port byte ***PY***. Individual bits of ***PY*** are accessed by their specific bit labels.

The next sub-sections describe in more detail the I/O port and associated bit

labels.

3.6.1 Input port PA

uTile supports one input port ***PA*** with six specific port bit labels ***A0, A1, A3, A4,*** and ***A5***. The remaining unused port bits ***A6*** and ***A7*** are defined but not used.

The uTile commands to read the port pins are composed of the the bit labels plus the postfix '.' notation. The commands transfer the data read from the input to the TOS.

Ai. ***[TOS]*** ← ***Ai***, where i = 5...0

Tile's input pre-processor produces transition types of input signals for each scanned input pin. Leading edge and trailing edge transition bits indicate the positive and negative going edges of an input signal respectively. Transition bits are set by the pre-processor when it determines a leading or trailing edge input transition. Upon reading a transition bit to the ***TOS***, the transition bit is cleared automatically. The commands to read transition bits to the ***TOS*** are as follows:

ALi. ***[TOS]*** ← ***ALi***, where ni= 5...0 (Leading edge)

ATi. ***[TOS]*** ← ***ATi***, where i = 5...0 (Trailing edge)

Double action (toggle) bits are produced for each input signal pin as well. The commands to transfer these bits to the TOS are:

ADi. ***[TOS]*** ← ***ADi***, where ni= 5...0 (Double action)

The uTile command to read all eight bits of the input port ***PA*** to the bit stack is:

PA. ***BSTK*** ← ***PA***

Since only six bits ***PA*** are mapped to ATmega328P port lines, the unused ***A7*** and ***A6*** bits are cleared to 0. The ***BSTK*** display will show 0 in bits locations 6 and 7.

The programming examples section of the manual shows how to read the

input port byte and bits.

3.6.2 Output port PY

uTile supports one output port ***PY*** with eight output port bits ***Y0, Y1, Y2, Y3, Y4, Y5, Y6*** and ***Y7.***

The uTile commands to write the TOS to the output port pins are composed of the the prefix '***.***' notation plus the destination output port bit.

.Yn ***Yn ← [TOS]***, where n = 7...0

The uTile command to write the ***BSTK*** to the eight output bits of port ***PY*** is:

.PY ***PY ← BSTK***

Since ***PY*** is eight bits wide, 8 bits of the ***BSTK*** are transferred to ***PY***.

The programming examples section of the manual will show how to use the ouptut port byte and bits.

3.6.3 uTile Virtual bytes PU and PV

uTile supports two virtual bytes ***PU*** and ***PV***. Since both bytes are identical in operation, the following description for ***PU*** applies equally well to ***PV***.

A virtual byte ***PU*** is provided for storing intermediate results. The individual bits of this byte are accessed by specific bit labels. The specific bit labels are ***Un*** where (n = 0..7). Bi-directional transfer between the virtual byte and ***BSTK*** is supported.

The uTile commands to read individual bits of ***PU*** to the TOS are as follows:
Un. ***[TOS] ← Un***, where n = 7...0

The uTile commands to write the TOS to individual bits of ***PU*** are as follows:

.Un ***Un*** ← ***[TOS]***, where n = 7...0

The uTile command to read the port ***PU*** to the bit stack is:
PU. ***BSTK*** ← ***PU***

The uTile commands to write the TOS to individual bits of ***PV*** are as follows:
.Vn ***[TOS]*** ← ***Vn***, where n = 7...0

The uTile commands to write the TOS to individual bits of ***PU*** are as follows:
.Vn ***Vn*** ← ***[TOS]***, where n = 7...0

The uTile command to write the ***BSTK*** to ***PU*** is:
.PV ***PV*** ← ***BSTK***

3.6.4 Timers

Sixteen delay timers are available. Timers T0...T3 are 0.1 s resolution, T4...Tb are 1 s resolution, Tc...Te are 1 min resulution and the remaining Tf is 10 min resulution..

The timers must be initialized with the desired timer values before use. The ***ldt*** command is a screen based loader for setting the timer values.

Each timer has two control inputs (trigger and reset) and one timer output. The bit labels associated with the timers are listed in Table 1, uTile Port & Bit Labels. The timer label subscript ***m*** ranges from 0,1...e,f.

The timers are re-triggerable one-shot timers. A timer is triggered by the rising edge of a trigger signal to the ***TKm*** input. The timing cycle begins on the falling edge of the ***TKm*** signal and continues until the timer times out, when the timer setting value counts down to zero.

The timer output signal ***TQm*** goes high as soon as the timing cycle begins. At the end of the timing cycle the ***TQm*** output goes low. If the ***TKm*** input signal is asserted at any time during the timing cycle, the timer is re-triggered and starts a new timing cycle again.

A high signal to the ***TRm*** input resets an active timer and immediately forces the timer output signal ***TQm*** to zero.

3.6.5 Flip-Flops

Sixteen set-reset flip-flops (F/F) are latches for storing inputs. Each flip-flop has an ***Sm*** input, an ***Rm*** input and a ***Qm*** output, (***m*** = 0,1...,e,f).

Applying a logic 1 to the ***Sm*** input causes the ***Qm*** output to go high. The output remains high even when the ***Sm*** input returns to a logic 0. Applying a logic 1 to the ***Rm*** input forces the ***Qm*** output to go low.

The reset input takes precedence over the set input, i.e. if both the ***Sm*** and ***Rm*** inputs are high, the output will be low.

The flip-flops are used in on/off latching control circuits such as motor start/stop circuits.

Although latching circuits can be implemented with basic standard logic elements, the flip-flop complex function is provided and is easier to program with.

The bit labels associated with the flip-flops are listed in Table1 of Section 3.5.1 above.

The flip-flop function is described by the following truth table:

SR Flip-Flop Truth Table		
S	**R**	**Q**
0	**0**	**0**
0	**1**	**0**
1	**0**	**1**
1	**1**	**0**

3.7 Complex functions

In addition to the Timers and Flip-Flops described above, uTile provides the gated counter function CNT and the toggle function TG for easier programming.

3.7.1 CNT command

The complex ***.CNT*** command function implements the counting of internally generated 1 s ticks while the ***.CNT*** input gate signal is high. Counting begins on the rising edge of the gate signal . On the falling edge of the gate signal counting stops and the accumulated counts are stored in an internal count buffer.

During counting while the gate signal is high, the Data display shows the accumulated 1 s tick counts. The count is capped to a 255 maximum value.

3.7.2 LDT command

Sixteen ***LDTm*** (m = 0…f) commands load the accumulated count measured by the ***CNT*** command into the timer ***Tm*** reload buffer. This command is always used in conjunction with the ***CNT*** command.

3.7.3 TG command

Eight complex ***TGn*** (n = 0...7) functions toggle an output bit on the rising edge of the input bit. The toggling function is the same as a divide-by-two function.

Each toggle function has a ***TGn*** input and a ***TGQn*** output.

The ***.TGn*** command reads the current input bit and compares it to the previous input sample state. If a rising edge transition is detected, the state of the output bit is toggled (complemented).

If the input bit is high but the previous sample was high, or if the input is low, the output is not changed.

The commands for reading the toggle input to the ***BSTK*** is ***.TGn*** and for writing the toggle output to the ***BSTK*** is ***.TGQn***.

4 Basic programming examples

This chapter introduces a number of uTile commands that are used to control the starting, running and stopping of application programs entered and executed from the UF0 space.

RUN mode type of commands are used to code application programs. They are the ‘Executable’ commands that are used to build the PLC application program and are entered in the UF0 file space.

EDIT mode commands are used to initialise uTile variables and perform housekeeping functions. These commands are not used in UF0 applications.

This manual shows all commands in boldface type characters. RUN mode executable commands are shown in boldface upper case while EDIT mode commands are shown in boldface lower case.

At the ‘Enter:’ prompt, type the desired command followed by a Carriage Return ‘CR’ key. All commands are entered in this way, the ‘CR’ key terminating the command entry.

4.1 Example 1 – fill, NOP, END and ex0 commands

When the Nano board is first powered up the user UF0 file space is filled with random data in each program line. uTile expects valid commands at each program line but because the random data does not match any command word in the dictionary, the interpreter displays 5 dots instead of a valid namestring.

The ***fill*** command initialises the UF0 program space with the ***NOP*** command.

Enter the ***fill*** command.

The following screen appears.

```
             <<<<  m328-uTile Programmable Logic Controller  >>>>
--------------------------------------------------------------------------------
          Bit Stack                                          Port Byte
       7 6 5 4 3 2 1 0             Data             7 6 5 4 3 2 1 0
       ===============             =====            ===============
       0 0 0 0 0 0 0 0             00000
```

```
--------------------------------------------------------------------------------

       000  NOP          008  NOP          016  NOP          024  NOP
       001  NOP          009  NOP          017  NOP          025  NOP
       002  NOP          010  NOP          018  NOP          026  NOP
       003  NOP          011  NOP          019  NOP          027  NOP
       004  NOP          012  NOP          020  NOP          028  NOP
       005  NOP          013  NOP          021  NOP          029  NOP
       006  NOP          014  NOP          022  NOP          030  NOP
       007  NOP          015  NOP          023  NOP          031  NOP



     Enter:
--------------------------------------------------------------------------------
 *** User File 0 ***
```

The NOP, or NO OPERATION command, is a dummy command that does nothing except to occupy a program line. Use of the ***fill*** command is not mandatory, but it is good practice to initialise the UF0 with NOP commands before starting to write a new program.

Type in the ***END*** command. The following screen appears.

```
              <<<<  m328-uTile Programmable Logic Controller  >>>>
--------------------------------------------------------------------------------
          Bit Stack                                          Port Byte
        7 6 5 4 3 2 1 0              Data              7 6 5 4 3 2 1 0
        ===============              =====             ===============
        0 0 0 0 0 0 0 0              00000
--------------------------------------------------------------------------------

       000  END          008  NOP          016  NOP          024  NOP
       001  NOP          009  NOP          017  NOP          025  NOP
       002  NOP          010  NOP          018  NOP          026  NOP
       003  NOP          011  NOP          019  NOP          027  NOP
       004  NOP          012  NOP          020  NOP          028  NOP
       005  NOP          013  NOP          021  NOP          029  NOP
       006  NOP          014  NOP          022  NOP          030  NOP
       007  NOP          015  NOP          023  NOP          031  NOP



     Enter:
--------------------------------------------------------------------------------
 *** User File 0 ***
```

The ***END*** command is stored at program line 000 and the cursor advances to line 001.

The next command is used to start and continuously execute the user program

stored so far in UF0. Type in the ***ex0*** command at the Enter: prompt. The PLC begins executing the UF0 program continuously.

The following screen appears.

```
             <<<<  m328-uTile Programmable Logic Controller  >>>>
--------------------------------------------------------------------------------
          Bit Stack                                          Port Byte
        7 6 5 4 3 2 1 0              Data              7 6 5 4 3 2 1 0
        ===============              =====             ===============
        0 0 0 0 0 0 0 0              00000
--------------------------------------------------------------------------------

       000  END           008  NOP           016  NOP           024  NOP
       001  NOP           009  NOP           017  NOP           025  NOP
       002  NOP           010  NOP           018  NOP           026  NOP
       003  NOP           011  NOP           019  NOP           027  NOP
       004  NOP           012  NOP           020  NOP           028  NOP
       005  NOP           013  NOP           021  NOP           029  NOP
       006  NOP           014  NOP           022  NOP           030  NOP
       007  NOP           015  NOP           023  NOP           031  NOP




     Enter: ex0
--------------------------------------------------------------------------------
 *** User File 0 *** *** Running User File Program ***
```

The status line changes to indicate the stored User File UF0 is running.

Every uTile program must have an ***END*** command as the last command in the program. uTile begins executing a program stored in UF0 at the program line 000. uTile then advances to the next program line and executes the command at line 001. This process continues until uTile encounters the ***END*** command at the end of the program. The ***END*** command resets uTile's program counter to line 000 and causes execution to start from there again.

Start the program execution by entering the ***ex0*** command to put uTile into the execution mode.

Halt the running program by pressing the Nano board's reset push button.

4.2 Example 2 – run command

The ***run*** command is a short-cut equivalent of entering the ***END*** command followed by the ***ex0*** command. After you finish entering a new program, the ***run*** command saves you the bother of typing in ***END*** and then ***ex0*** commands to start the program.

Clear the UF0 space by entering the ***fill*** command. With the cursor on program line 000, enter the ***run*** command. The following screen appears:

```
              <<<<  m328-uTile Programmable Logic Controller  >>>>
--------------------------------------------------------------------------------
          Bit Stack                                          Port Byte
        7 6 5 4 3 2 1 0              Data                7 6 5 4 3 2 1 0
        ===============              =====               ===============
        0 0 0 0 0 0 0 0              00000
--------------------------------------------------------------------------------

       000  NOP          008  NOP          016  NOP          024  NOP
       001  NOP          009  NOP          017  NOP          025  NOP
       002  NOP          010  NOP          018  NOP          026  NOP
       003  NOP          011  NOP          019  NOP          027  NOP
       004  NOP          012  NOP          020  NOP          028  NOP
       005  NOP          013  NOP          021  NOP          029  NOP
       006  NOP          014  NOP          022  NOP          030  NOP
       007  NOP          015  NOP          023  NOP          031  NOP




     Enter: run
--------------------------------------------------------------------------------
 *** User File 0 *** *** Running User File Program ***
```

Notice the ***run*** command does not appear at program line 000, but the status line indicates that a program is running.

When the ***run*** command is entered, uTile does two things. It first inserts the ***END*** command at line 000. It then immediately executes the ***ex0*** command to start running the UF0 program.

When execution starts the page display remains unchanged since uTile is now in the RUN mode. The page display is updated only after the running program is halted and the EDIT mode is re-entered. Press the reset push button to restart uTile and re-enter the EDIT mode. The page display is updated and the ***END*** command shows up at line 000.

If there is an ***END*** command at the end of a UF0 program, don't use the ***run*** command to start the program, as this would add another ***END*** command. Use the ***ex0*** command instead.

4.3 Example 3 – KEY command

When any UF0 program with an ***END*** command at the end of the program is running, for example in the Example 2 program above, the only way to interrupt the program is to reset the controller by pressing the reset switch or powering down the board.

The ***KEY*** command provides a means to invoke a break-out from a running program from the keyboard.

Enter program below with the ***KEY*** command at line 000.

Start the program by entering the ***run*** command. The screen looks like this:

```
             <<<<  m328-uTile Programmable Logic Controller  >>>>
--------------------------------------------------------------------------------
          Bit Stack                                        Port Byte
        7 6 5 4 3 2 1 0              Data              7 6 5 4 3 2 1 0
        ===============              =====             ===============
        0 0 0 0 0 0 0 0              00000
--------------------------------------------------------------------------------

       000  KEY           008  NOP           016  NOP           024  NOP
       001  NOP           009  NOP           017  NOP           025  NOP
       002  NOP           010  NOP           018  NOP           026  NOP
       003  NOP           011  NOP           019  NOP           027  NOP
       004  NOP           012  NOP           020  NOP           028  NOP
       005  NOP           013  NOP           021  NOP           029  NOP
       006  NOP           014  NOP           022  NOP           030  NOP
       007  NOP           015  NOP           023  NOP           031  NOP




     Enter: run
--------------------------------------------------------------------------------
 *** User File 0 *** *** Running User File Program ***
```

uTile is in the RUN mode and the status line changes to indicate the UF0 program is running.

The ***KEY*** command at line 000 allows you to break-out of the running

program and return to the edit mode by pressing the '/' key. Press the '/' key to break out of the running program. The program halts and the screen looks like:

```
             <<<<  m328-uTile Programmable Logic Controller  >>>>
--------------------------------------------------------------------------------
          Bit Stack                                          Port Byte
        7 6 5 4 3 2 1 0              Data              7 6 5 4 3 2 1 0
        ===============              =====             ===============
        0 0 0 0 0 0 0 0              00000
--------------------------------------------------------------------------------

       000  KEY          008  NOP          016  NOP          024  NOP
       001  NOP          009  NOP          017  NOP          025  NOP
       002  NOP          010  NOP          018  NOP          026  NOP
       003  NOP          011  NOP          019  NOP          027  NOP
       004  NOP          012  NOP          020  NOP          028  NOP
       005  NOP          013  NOP          021  NOP          029  NOP
       006  NOP          014  NOP          022  NOP          030  NOP
       007  NOP          015  NOP          023  NOP          031  NOP




     Enter: run
--------------------------------------------------------------------------------
 *** User File 0 *** *** Running User File Program ***     Strike any key -->
```

Press any key to return to the edit mode. Returning to the edit mode updates the UF0 display and the screen looks like this:

```
             <<<<  m328-uTile Programmable Logic Controller  >>>>
--------------------------------------------------------------------------------
          Bit Stack                                          Port Byte
        7 6 5 4 3 2 1 0              Data              7 6 5 4 3 2 1 0
        ===============              =====             ===============
        0 0 0 0 0 0 0 0              00000
--------------------------------------------------------------------------------

       000  KEY          008  NOP          016  NOP          024  NOP
       001  END          009  NOP          017  NOP          025  NOP
       002  NOP          010  NOP          018  NOP          026  NOP
       003  NOP          011  NOP          019  NOP          027  NOP
       004  NOP          012  NOP          020  NOP          028  NOP
       005  NOP          013  NOP          021  NOP          029  NOP
       006  NOP          014  NOP          022  NOP          030  NOP
       007  NOP          015  NOP          023  NOP          031  NOP




     Enter:
--------------------------------------------------------------------------------
 *** User File 0 ***
```

The ***END*** command now shows up at line 001, it was automatically inserted by the ***run*** command. Use the ***ex0*** command if you want to run the program again.

4.4 Example 4 - '?' and '/' commands

The uTile program has a built-in help screen that is invoked by typing in the ***?*** Command. The help screen is shown below:

```
             <<<<  m328-uTile Programmable Logic Controller  >>>>
--------------------------------------------------------------------------------
           Bit Stack                                        Port Byte
        7 6 5 4 3 2 1 0              Data               7 6 5 4 3 2 1 0
        ===============              =====              ===============
        0 0 0 0 0 0 0 0              00000
--------------------------------------------------------------------------------
  Logic   Bit Stack           I/O - Commands          General Commands    Timers
  -----   ---------    --------------------------    ------------------   ------
   and       1.          In:  Ai. ALi. ATi. ADi.       fill  ex0   run     TQm.
   or        0.         Out: .Yn                       ins   key   end    .Tkm
   xor     dup 2dup    Virt:  Un., .Un, Vn., .Vn       del    /    ver    .TRm
   ! !!     drop     (i = 0..5, n = 0..7, m = 0..f)     ?   load   store   ldt
   nop    swap f. z.                                  list  read   write   LDTm

  ------- Byte Commands -------     -- Flip/Flop --    --- Clock Pulses ---
   PA.  .PY   PU.  .PU  PV. .PV     .Sm  .Rm   Qm.     clka.   clkb.  clkc.
   PPA  PBS   PPU  PPV  PPY          -- Counter --     200 ms  600 ms 1.8 s
  Ground D8 to enable Autostart          .CNT        Timers: T0..T3 (0.1 s)
  Ground D9 to invert outputs         -- Toggle --            T4..Tb (1 s)
  Date: 2021/10/07 Ver 1cx             .TGn  TGQn.   Tc..Te (min), Tf (10 min)

--------------------------------------------------------------------------------
 *** User File 0 ***                                        Strike any key -->
```

The screen shows a list of all the uTile commands for programming a UF0 application and for controlling the operation of the PLC.

The **/** command sets the program line cursor to line 000 and refreshes the editor screen. The command can be invoked with the line cursor at any position and the line cursor will move to line 000.

5 Advanced programming

This section provides program examples to illustrate the use of the uTile commands. The programs will generally be reading uTile input port pins, performing some logic operations on the input data, then writing the results of the operations to output port pins. The uTile Test Rig described in Section 7 shows a simple circuit that provides 4 input stimulus signals and 4 output port signal status monitors using externally connected switches and LEDs.

The UF0 application program snippets shown in the examples are by no means the only way to write the programs, as there usually other ways of achieving the same result.

5.1 Example 5 – Reading/writing I/Os

An extensive set of commands for accessing I/O ports are provided as describer in Section 3.6 The following program examples illustrate the use of these commands.

Enter the following program snippet in the UF0 space:

```
000  A0.          008  KEY          016  NOP          024  NOP
001  .Y0          009  END          017  NOP          025  NOP
002  A1.          010  NOP          018  NOP          026  NOP
003  .Y1          011  NOP          019  NOP          027  NOP
004  A2.          012  NOP          020  NOP          028  NOP
005  .Y2          013  NOP          021  NOP          029  NOP
006  A3.          014  NOP          022  NOP          030  NOP
007  .Y3          015  NOP          023  NOP          031  NOP
```

Start executing the program by entering the ***ex0*** command.

Line 000 command reads in the state of ***A0*** input connected to switch SW0 of the test rig and pushes the result to the B***STK***. The normally open (NO) SW0 switch pulls the ***A0*** input low when the switch is activated.

Line 001 command pops the ***BSTK*** and writes the bit to ***Y0*** output that is connected to LED0 of the test rig. The LED reflects the state of input switch SW0.

Lines 000 and 001 of the program reads the state ***A0*** input and transfers the result via the ***BSTK*** to the ***Y0*** bit of output port ***PY***.

Similarly lines 003 through 007 of the program transfer the states read at inputs ***A1***, ***A2*** and ***A3*** to output bits ***Y1***, ***Y2***, and ***Y3*** of ***PY***.

Observe all four LEDs are on initially when all four switches are not bactivated. Press each switch in turn and see the associated LED turns off.

Break out of the running program by pressing the '***/***' key and clear the UF0 space using the ***F.*** command. If you wish to invert the input signals before they are sent to the outputs, you can use the bit complement command ***!*** to invert the TOS bit before it is written to the output as follows:

```
000  A0.          008  .Y2          016  NOP          024  NOP
001  !            009  A3.          017  NOP          025  NOP
002  .Y0          010  !            018  NOP          026  NOP
003  A1.          011  .Y3          019  NOP          027  NOP
004  !            012  KEY          020  NOP          028  NOP
005  .Y1          013  END          021  NOP          029  NOP
006  A2.          014  NOP          022  NOP          030  NOP
007  !            015  NOP          023  NOP          031  NOP
```

Now when the input switch is activated (pressed), the associated LED goes on. See if you can rewrite the program to change the input to output assignments to a different ordering, such as ***A0*** to ***Y3***, ***A1*** to ***Y2***, ***A2*** to ***Y1*** and ***A3*** to ***Y0***.

Sometimes it is more efficient for an application to handle byte sized data instead of individual data bits. The byte commands are provided for this purpose. The next example illustrates some of the byte available operations.

Enter ***ex0*** command to run the following program snippet:

```
000  PA.          008  NOP          016  NOP          024  NOP
001  PPA          009  NOP          017  NOP          025  NOP
002  !!           010  NOP          018  NOP          026  NOP
003  .PY          011  NOP          019  NOP          027  NOP
004  KEY          012  NOP          020  NOP          028  NOP
005  END          013  NOP          021  NOP          029  NOP
006  NOP          014  NOP          022  NOP          030  NOP
007  NOP          015  NOP          023  NOP          031  NOP
```

Athe list below describes the action of each line of the program:

Line	Command	Action
000	PA.	Read all port ***PA*** bits to ***BSTK***
001	PPA	Display ***PA*** byte on Port Byte display
002	!!	Complement ***BSTK*** byte
003	.PY	Write ***BSTK*** to Port ***PY***
004	KEY	Program break-out
005	END	

The ***PPA*** command is mostly used during a program's development and debugging stage to provide a real time display of input conditions. The ***!!*** command complements the ***BSTK*** before it is written to the output port ***PY***.

5.2 Example 6 – Transition and toggle bits

The uTile input pre-processor extracts useful characteristics of the input signals and stores this information in special input status bits. The input ***ALn*** leading edge and ***ATn*** trailing edge transition bits store leading and trailing edge information for each of the scanned input pins. See Section 3.6.1 for the bit labels.

Transition bits are useful for triggering timers and flip-flops. Enter and run the following code snippet:

```
000 AT0.    008 END     016 NOP     024 NOP
001 .S0     009 NOP     017 NOP     025 NOP
002 AT1.    010 NOP     018 NOP     026 NOP
003 .R0     011 NOP     019 NOP     027 NOP
004 NOP     012 NOP     020 NOP     028 NOP
005 Q0.     013 NOP     021 NOP     029 NOP
006 .Y0     014 NOP     022 NOP     030 NOP
007 KEY     015 NOP     023 NOP     031 NOP
```

The program uses the trailing edge transition bit ***AT0*** for switch SW0 to set ***S0*** and transition bit ***AT1*** for SW1 to set R0 of flip-flop 0. The F/F Q0 output is written to ***Y0*** output that is connected to LED0.

Notice that as the switches are NO types, the signals present on the input port pins ***A0*** and ***A1*** are normally high and go low when the switches are pressed. The program reads the trailing edge bits that go high when the switches are pressed.

```
Line   Command     Action

000    AT0.        SW0 trailing edge, high when SW0 pressed
001    .S0         F/F0 set when SW0 pressed
002    AT1.        SW1 trailing edge, high when SW1 pressed
003    .R0         F/F0 reset when SW1 pressed
004    Q0.         F/F0 output to TOS
005    .Y0         Pop BSTK and write to Y0
006    KEY         Program break-out
005    END
```

Modify the program by changing line 000 to ***AL0.*** and line 002 to ***AL1.*** and run the program. Notice that the F/F now responds when the SW0 or SW1 switch is released and not when pressed.

We now introduce the input toggle (or double action) bit. The toggle bit can be viewed as a divide-by-two function of the input signal that changes state for every rising edge change in input states. Enter and run the following program snippet:

```
000 AD0.    008 KEY     016 NOP     024 NOP
001 .Y0     009 NOP     017 NOP     025 NOP
002 AD1.    010 NOP     018 NOP     026 NOP
003 .Y1     011 NOP     019 NOP     027 NOP
004 AD2.    012 NOP     020 NOP     028 NOP
005 .Y2     013 NOP     021 NOP     029 NOP
006 AD3.    014 NOP     022 NOP     030 NOP
007 .Y3     015 NOP     023 NOP     031 NOP
```

The program uses the toggle bits on the four input switches and maps them to the four indicator LEDs. The LEDs change state each time a corresponding switch is pressed and released.

5.3 Logic operators

The logic operators ***AND, OR*** and ***!*** form a functionally complete set of Boolean operators for describing every Boolean function.

The next set of examples introduces the usage of these important operators.

5.3.1 Example 8 – AND command

This example illustrates the ***AND*** operator. Enter and ***ex0*** the following program snippet:

```
000  A0.          008  NOP          016  NOP          024  NOP
001  !            009  NOP          017  NOP          025  NOP
002  A1.          010  NOP          018  NOP          026  NOP
003  !            011  NOP          019  NOP          027  NOP
004  AND          012  NOP          020  NOP          028  NOP
005  .Y0          013  NOP          021  NOP          029  NOP
006  KEY          014  NOP          022  NOP          030  NOP
007  END          015  NOP          023  NOP          031  NOP
```

The ***!*** commands in lines 001 and 003 inverts the signals read in from SW0 and SW1 so that the values on the ***BSTK*** are 1 when the switches are pressed. The ***AND*** operator in line 004 pops the first two bits from the ***BSTK***, forms the logical ***AND*** of the two bits, then pushes the result of the operation back to the ***TOS***. Line 005 pops the result from the ***BSTK*** writes the bit to output pin ***Y0***.

Pressing SW0 and SW1 turns on the LED0 indicator.

5.3.2 Example 7 – OR command

This example illustrates the ***OR*** operator. Enter and ***ex0*** the following program snippet:

```
000  A0.          008  NOP          016  NOP          024  NOP
001  !            009  NOP          017  NOP          025  NOP
002  A1.          010  NOP          018  NOP          026  NOP
003  !            011  NOP          019  NOP          027  NOP
004  OR           012  NOP          020  NOP          028  NOP
005  .Y0          013  NOP          021  NOP          029  NOP
006  KEY          014  NOP          022  NOP          030  NOP
007  NOP          015  NOP          023  NOP          031  NOP
```

The ***OR*** operator in line 004 pops the first two bits from the ***BSTK***, forms the logical ***OR*** of the two bits, then pushes the result of the operation back to the ***TOS***. Line 005 pops the result from the ***BSTK*** writes the bit to output pin ***Y0***.

Pressing SW0 or SW1 turns on the LED0 indicator.

Pressing both SW0 and SW1 at the same time turns off the LED0 indicator.

5.3.3 Example 9 – XOR command

This example illustrates the ***XOR*** operator. Enter and ***ex0*** the following program snippet:

```
000  A0.          008  NOP          016  NOP          024  NOP
001  !            009  NOP          017  NOP          025  NOP
002  A1.          010  NOP          018  NOP          026  NOP
003  !            011  NOP          019  NOP          027  NOP
004  XOR          012  NOP          020  NOP          028  NOP
005  .Y0          013  NOP          021  NOP          029  NOP
006  KEY          014  NOP          022  NOP          030  NOP
007  END          015  NOP          023  NOP          031  NOP
```

The ***XOR*** operator in line 004 pops the first two bits from the ***BSTK***, forms the logical ***XOR*** of the two bits, then pushes the result of the operation back to the ***TOS***. Line 005 pops the result from the ***BSTK*** writes the bit to output pin ***Y0***.

Pressing SW0 or SW1 turns on the LED0 indicator.

Pressing both SW0 and SW1 at the same time turns off the LED0 indicator. It is this last behaviour that distinguishes the ***XOR*** from the ***OR*** operator.

5.3.4 Example 10 – Bit Stack commands

Several bit stack commands are provided for initialising and juggling ***BSTK*** bits. The help screen under Bit Stack lists these commands:

Bit Stack	Operation
1.	Push 1 to ***BSTK***

0.	Push 0 to ***BSTK***
dup	Copies ***TOS*** bit and push to ***BSTK***
2dup	Copies ***TOS, TOS+1*** bits and push to ***BSTK***
drop	Pops ***BSTK***. ***TOS*** bit is discarded
swap	Swaps ***TOS, TOS+1*** bits on ***BSTK***
f., z.	Fill ***BSTK*** with 1's, 0's
!	Complement ***TOS*** bit
!!	Complement ***BSTK*** byte

Try out the each of the commands and observe the bit stack display to see the effect of the commands on the bit stack contents as you enter and execute each command in turn.

Enter the ***fill*** command to load UF0 with ***NOP*** commands.
Enter the ***Z.*** command. The ***BSTK*** is filled with 0 bits.
Enter the ***1.*** command. The ***TOS*** bit is now 1.
Enter the 0***.*** command. The ***TOS*** bit is now 0.

The ***BSTK*** display shows (0 0 0 0 0 0 1 0).

Enter the ***2DUP*** command. ***BSTK*** display: (0 0 0 0 1 0 1 0).
Enter the ***DUP*** command. ***BSTK*** display: (0 0 0 1 0 1 0 0).
Enter the ***DROP*** command. ***BSTK*** display: (0 0 0 0 1 0 1 0).
Enter the ***F.*** command. The ***BSTK*** is filled with 1 bits.
Enter the ***!*** command. The ***TOS*** bit is cleared to 0.
Enter the ***!!*** command. The ***BSTK*** byte is complemented.

All the bit stack commands are executable (can be compiled in the UF0 space) except for the ***fill*** command, which is an EDIT mode command.

5.3.5 Example 11 – Virtual byte commands

The virtual storage bytes ***PU*** and ***PV*** are general purpose storage bytes with associated byte and bit commands for read and write access. The description below for ***PU*** applies equally to ***PV***.

The ***PU*** byte is used for storing data temporarily. Enter and ***ex0*** the

following program snippet:

```
000  PA.          008  .Y2          016  NOP          024  NOP
001  !!           009  A3.          017  NOP          025  NOP
002  .PU          010  .Y3          018  NOP          026  NOP
003  U0.          011  KEY          019  NOP          027  NOP
004  .Y0          012  END          020  NOP          028  NOP
005  U1.          013  NOP          021  NOP          029  NOP
006  .Y1          014  NOP          022  NOP          030  NOP
007  A2.          015  NOP          023  NOP          031  NOP
```

The program reads input port ***PA*** to ***BSTK***, complements ***BSTK***, then writes ***BSTK*** to port ***PU***. ***PU*** now contains an inverted copy of the inputs read from ***PA***. Bits ***U0*** and ***U1*** are written to outputs ***Y0*** and ***Y1***, while bits ***A2*** and ***A3*** are written to outputs ***Y2*** and ***Y3***.

5.3.6 Internal CLKA, CLKB, CLKC bits

Three general purpose internal clock bits are provided for use in UF0 applications.

The three clock bits are accessed by commands ***CLKA., CLKB.*** and ***CLKC.*** The clock bits are square wave signals with 200 ms, 600 ms and 1800 ms pulse widths. The clock period is twice as long as the clock pulse width.

5.3.7 Example 12 – Clock signal bits

This example illustrates the typical use of the internal clock signal bits in a program application.

Enter and ***ex0*** the following program snippet:

```
000  CLKA.        008  AND          016  NOP          024  NOP
001  CLKB.        009  .Y3          017  NOP          025  NOP
002  CLKC.        010  KEY          018  NOP          026  NOP
003  .Y0          011  END          019  NOP          027  NOP
004  .Y1          012  NOP          020  NOP          028  NOP
005  .Y2          013  NOP          021  NOP          029  NOP
006  CLKA.        014  NOP          022  NOP          030  NOP
007  CLKC.        015  NOP          023  NOP          031  NOP
```

The program lines 000 through 005 push the ***CLKA***, ***CLKB*** and ***CLKC*** bits to the ***BSTK***, then pops the ***BSTK*** and writes to outputs ***Y0***, ***Y1*** and ***Y2.*** Each clock bit drives LED0, LED1 and LED2 on the test rig. Notice the assignment of clock bits to outputs, eg. ***CLKA*** is written to ***Y2***. This is due to the last-in-first-out nature of ***BSTK*** operations.

Program lines 006 through 009 reads in the ***CLKA*** and ***CLKC*** bits, performa a logical ***AND***, and writes the result to output ***Y3.*** You can observe combined signal on LED3.

Try various combinations of clock signals, substitute the ***AND*** with the ***OR*** command, and observe the differences.

5.3.8 ldt command – Load timers

The sixteen programmable delay (one-shot) timers are set up with the ldt command. See Section 3.6.4 for timer details. Before using the timers in your application, they must be loaded with the desired delay time values.

Timers T0..T3 are 0.1 s resolution, timers T3..Tb are 1 s resolution. timer Tc..Te are 1 min resolution, and the remaining timer Tf is 10 min resolution.

Enter the ***ldt*** command to bring up the delay timer settings screen:

```
             <<<<  m328-uTile Programmable Logic Controller  >>>>
-------------------------------------------------------------------------------
          Bit Stack                                        Port Byte
       7 6 5 4 3 2 1 0              Data              7 6 5 4 3 2 1 0
       ===============              =====             ===============
       0 0 0 0 0 0 0 0              00000
-------------------------------------------------------------------------------

             T0     T1     T2     T3     T4     T5     T6     T7
             ================================================
             000    000    000    000    000    000    000    000

             T8     T9     TA     TB     TC     TD     TE     TF
             ================================================
             000    000    000    000    000    000    000    000
```

```
     Enter:
--------------------------------------------------------------------------------
 *** Load timers, Z clear all, <CR> exit ***
```

Enter the desired setting for each timer. Return to the Editor screen by entering a blank line, i.e. press the '**CR**' key.

After a warm start power-up reset, the timer settings remain undisturbed. When a user's application program in UF0 is saved to EEPROM with the ***store*** command, the timer settings are saved as well. When the ***load*** command is used to re-load a saved UF0 program, the saved timer settings are also re-loaded.

Timer settings saved as part of a UF0 application to a host PC disc file with the ***read*** command are restored with the ***write*** command.

The valid range for timer settings is from 000 to 255. If you enter a number greater than 255, for example 9999, the entry is capped to a maximum value of 255. Complete the number entry by pressing the '**CR**' key. Use the up and down arrow keys to move from one timer to the next.

5.3.9 Example 13 - *CNT* command

The ***CNT*** command is used to count internally generated 1 s ticks in a gated counter. The counter is enabled by a high gate signal and the accumulated counts at the end of the gate can be used as a numerical value to load a delay timer.

Enter and ***ex0*** the following program snippet:

```
000  A0.          008  NOP          016  NOP          024  NOP
001  !            009  NOP          017  NOP          025  NOP
002  .CNT         010  NOP          018  NOP          026  NOP
003  KEY          011  NOP          019  NOP          027  NOP
004  END          012  NOP          020  NOP          028  NOP
005  NOP          013  NOP          021  NOP          029  NOP
006  NOP          014  NOP          022  NOP          030  NOP
007  NOP          015  NOP          023  NOP          031  NOP
```

Ground input A0 and observe the Data display incrementing from 0 counts every 1 s until A0 is opened again. The Data display holds the last accumulated count value. If you ground A0 input again, the Data display starts incrementing again from 0 counts.

The ***CNT*** command is used as an alternative to the ***ldt*** command for entering a numerical value in an application that is run in a stand-alone environment and is not connected to Minicom user interface.

5.3.10 Example 14 – LDTm commands

The ***LDTm*** commands (m = 0..f) fetches a numerical value generated by the ***CNT*** command and loads the value into the specified timer m.

Initialise UF0 space with ***NOP*** commands using the ***fill*** command. Use the ***ldt*** command to zero all the timers (press 'Z'). UF0 is now filled with NOPs and all the timers are cleared to 0. Press 'CR" to return to the edit screen.

Enter and ***ex0*** the following program snippet:

```
000  A0.          008  NOP          016  NOP          024  NOP
001  !            009  NOP          017  NOP          025  NOP
002  .CNT         010  NOP          018  NOP          026  NOP
003  LDT8         011  NOP          019  NOP          027  NOP
004  KEY          012  NOP          020  NOP          028  NOP
005  END          013  NOP          021  NOP          029  NOP
006  NOP          014  NOP          022  NOP          030  NOP
007  NOP          015  NOP          023  NOP          031  NOP
```

Ground input A0 and open A0 when the Data display indicates any arbitrary value, say 9. Enter '/' to break out of the running program. Enter the ***ldt*** command and verify that the T8 setting = 9, the value captured by ***CNT***.

5.3.11 Example 15 - TGn and TGQn commands

The eight .***TGn*** and ***TGQn.*** (n = 0..7) commands are the input and output of each ***TGn*** toggle function. Each toggle function performs a divide-by-two on the rising edge of the input bit. The toggle function operates in a way similar to the input double-action bits ***ADi*** (i = 0..5). The difference being ***TGn*** inputs are can read all internal uTile bits.

Enter and ***ex0*** the following program snippet:

```
000  CLKB.        008  NOP          016  NOP          024  NOP
001  DUP          009  NOP          017  NOP          025  NOP
002  .Y0          010  NOP          018  NOP          026  NOP
003  .TG0         011  NOP          019  NOP          027  NOP
004  TGQ0.        012  NOP          020  NOP          028  NOP
005  .Y3          013  NOP          021  NOP          029  NOP
006  KEY          014  NOP          022  NOP          030  NOP
007  END          015  NOP          023  NOP          031  NOP
```

The CLKB signal drives TG8 toggle and also Y0 (LED0). The TGQ0 output drives Y3. With the program running on the uTile test rig, you can see the toggle output (LED3) is flashing at one-half the rate of the CLKB signal.

5.4 Byte commands

Several byte commands are provided for handling byte wide operands. They are used mainly during the development stage of an application to display the contents of ports and data bytes, and for performing data transfers at the byte level.

5.4.1 Example 16 – PA, PY, PU, PV data transfers

The ***PU.***, ***PV.*** and ***PA.*** byte read commands transfers the specified port bytes directly to the ***BSTK*** while the ***.PU***, ***.PV*** and ***.PY*** byte write commands transfer ***BSTK*** data directly to the specified bytes.

The ***PPA***, ***PPU***, ***PPV*** and ***PPY*** commands displays the contents of the specified port/data byte on the Port Byte display. The PBS command displays the ***BSTK*** data on the Bit Stack display.

Note that during RUN mode the port byte and bit stack data displays are updated at a 100 ms rate.

Enter and ***ex0*** the following program snippet:

```
000  PPA     Show PA byte on Port Byte display
001  PA.     Move PA byte to BSTK
002  !!      Invert BSTK
003  .PY     Move BSTK to PY
004  KEY
```

```
005  END
```

Observe on the test rig the LEDs turn on in response to pressing the PB switches. Notice the bits shown under the Port Byte display are a copy of the ***PA*** byte. Bits 0..3 change states as the corresponding ***A0*** .. ***A3*** inputs change states.

Modify the program by changing line 000 command as below and ***ex0*** the program again.

```
000  PBS    Update and show BSTK under Bit Stack display
```

Observe the Bit Stack display shows what is in the ***BSTK*** while the program is running. Note the ***BSTK*** data is inverted (line 002) before it is written to ***PY*** (line 003).

Enter and ***ex0*** the following program snippet:

```
000  PA.    Move PA byte to BSTK
001  .PU    Move BSTK to PU
002  PU.    Move PU byte to BSTK
003  !!     Invert  BSTK
004  .PY    Move BSTK to PY
005  KEY
006  END
```

Lines 001 & 002 show store data to a virtual byte then retrieve the stored data later on.

6 Housekeeping functions

Housekeeping commands are EDIT mode type that are not executable in a UF0 program. These commands are used for creating a UF0 program listing, and a UF0 program space and delay timer settings binary image in Intel hex file format.

Both the listing and image data are encoded as ascii text data which can then be written to disc files on the host PC for offline permanent storage.

A UF0 program space and delay timer settings binary image stored on a disc file can be uploaded to uTile from the host PC again using the ***read*** command.

6.1 List command

The ***list*** command opens a menu screen and displays the message:

list .... 'CR' to send .... '/' to abort

Type ‘/’ to abort the command and return to the editor screen. Type ‘CR’ to list on screen the contents of UF0 program space and the timer settings. When the listing is completed, type ‘CR’ again to return to the editor screen.

Minicom can save the listing to a disc file on the host PC by opening a log session before starting the listing, then closing the log session when the listing is fnished. This is done as follows:

a) Open a log session:

Enter Minicom ^A L command. A message appears in a text box:

Capture to which file?

> minicom.cap

Type in a file name with a .lst extension. Press ‘CR’ to start the session.

b) Close a log session:

Enter Minicom ^A L command. A message appears in a text box:

Capture file

Close Pause Exit

Press ‘CR’ to close the log session and the capture file.

Type a ‘CR’ to return to the editor screen.

Note that If you use the ***list*** command again and use the same log file name, Minicom appends the new listing to the old log file instead of overwriting the old

log file.

6.2 Write command

The ***write*** command is similar to the ***list*** command operation. The ***write*** command opens a menu screen and displays the message:

File write .... 'CR' to send .... '/' to abort

Follow the procedure for the ***list*** command for saving the dumped hex file to disc, but use a .hex instead of .lst file extension.

6.3 Read command

The ***read*** command reads in a UF0 image hex file sent from ta disc file stored on the host PC. The ***read*** command clears the edit screen’s program area and showss the message:

Reading file ..... '/' to abort

uTile waits for the incoming transmission from Minicom. Start the Minicom file upload by entering ^A S to start the file upload. An Upload text box menu appears. Scroll down to ‘ascii’ and press ‘CR’ to upload the ascii hex file. A new text box opens and shows the files in the upload directory. Scroll down to the required hex file and tag the file. Press ‘CR’ to start the upload.

A text box appears and shows the file transfer information. At the end of the transfer a message sappears on the screen’s status line:

File read completed - 'CR' for menu -->

Press ‘CR’ to return to the EDIT mode.

6.4 Ver command

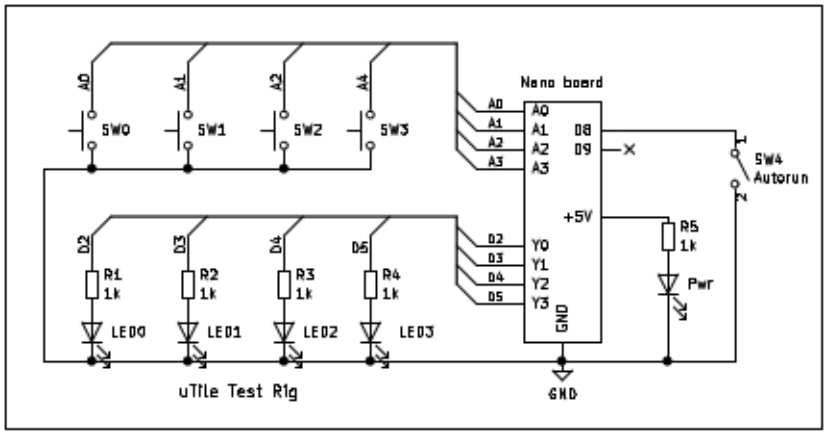
Enter the ***VER*** command to show the current version of the m328-uTile controller program in the status line of the main screen.

7 uTile Test Rig

The schematic diagram in Figure 1 below shows a test rig consisting of four discrete switches (NO push-button switches) as input devices and four LEDs as output indicators. The switches provide digital input signals to the uTile input pins while the LEDs provide visual indication on the status of the output pins.

The SW4 switch is connected to the Nano board's D8 pin and activates the Autostart feature when the switch is closed.

The Pwr LED is optional and is useful when the Nano board is inside and enclosure and the on-board ON LED is not visible.



The example programs presented Section 5 use the test rig switches oo simulate digital signals. The LEDs are used to display digital output status.

8 uTile Nano board pinouts

The uTile input and output ports are mapped to the Nano board ports as shown in the tables below. Column 1 lists the uTils port bit lables, column 2 shows the ATmega328P pin labels and column 3 shows the Nano board connection labels.

8.1 Port PA mapping

uTile's input port ***PA*** is mapped to the Nanocontroller board pins as shown below:

Inputs	Port pin	Nano pin
A0	PC0	A0
A1	PC1	A1
A2	PC2	A2
A3	PC3	A3
A4	PC4	A4
A5	PC5	A5

Open inputs are pulled high by internal pull-up resistors.

8.2 Port PY mapping

The output port ***PY*** is mapped to the Nano controller board pins as shown below:

Outputs	Port pin	Uno pin
Y0	PD2	Digital 2
Y1	PD3	Digital 3
Y2	PD4	Digital 4
Y3	PD5	Digital 5
Y4	PD6	Digital 6

Outputs	Port pin	Uno pin
Y5	PD7	Digital 7
Y6	PB4	Digital 12
Y7	PB5	Digital 13

uTile has two special purpose dedicated inputs, the auto-start input on D8 and the out_sense input on D9. These inputs are activated by adding a jumper between the input line and ground.

8.3 D8 and D9 control inputs

Two Nano board port pins are configured as inputs for enabling auto-start and output sense control inputs. The pin functions are disabled when the pins are not grounded. The pin functions are described below.

Inputs	Port pin	Nano pin
Auto-start	PB0	Digital 8
Out_sense	PB1	Digital 9

a) Auto-start input

Grounding the auto-start input line D8 enables the autostart function which causes uTile to load any UF0 program stored in internal EEPROM back to the UF0 program space and start executing it immediately after a power-up or reset event.

Autostart alows uTile run headless (no user interface connected) and without manual intervention for unattended operation.

b) Output sense control

Grounding the out_sense input line D9 causes uTile to invert all output port ***PY*** bits before writing them to the output lines.

The default state of the port ***PY*** bits is 0 (logic low) following a power-up or reset event. If ***PY*** is interfacing to a relay module with active-low inputs,

the relays are driven to the ON state under a power-up or reset condition, which results in an un-safe condition. To avoid this un-safe operating condition, activating the out_sense function will keep the relays in the OFF state following a power-up or reset condition.

Use of output sense feature is recommended over inverting the ***PY*** bits using uTile program commands because inversion takes place only after the UF0 program begins executing and not before.

9 Command summary

The m328-uTile command words are summarized below. Commands are case insensitive and may be entered in upper or lower case.

Editor (Interpreter) Commands	
?	Print a help screen.
/	Redraw the main screen in EDIT mode.
INS	Insert a 'NOP' command at the current program line position. All commands at current line are pushed down by 1 line and the command last command on line 255 drops off from UF0 and is lost.
DEL	Delete the command at the current program line position. All commands below the current program line move up by one line position. A 'NOP' command is inserted on program line 255.
NOP	No operation. Execution continues at the next program line.
RUN	Inserts an END command and begins execution of user program in UF0.
EX0	Execute UF0 program. And END command must be placed at the end of the user's program.
END	Mark the end of the user's program in UF0. Reset uTile program counter to line 0 and continue execution.
FILL	Fill current UF0 with NOP commands.
KEY	Break-out from running program and return to the EDIT mode. Enter a '***/***' to invoke the program break-out.
VER	Display current version number.
Logic Commands	
AND	Pop TOS and [TOS+1] from BSTK. perform logic ***AND*** of the two bits and push result to BSTK. TOS ← TOS & [TOS+1]
OR	Pop TOS and [TOS+1] from BSTK. perform logic ***OR*** of the two bits and push result to BSTK. TOS ← TOS & [TOS+1]

	TOS <-- TOS V [TOS+1]
XOR	Pop TOS and [TOS+1] from BSTK. perform logic ***XOR*** of the two bits and push result to BSTK. TOS ← TOS ExOR [TOS+1]
!	Pop BSTK, complement and push to BSTK. TOS <-- ~TOS
!!	Complement 8 bits of BSTK. BSTK ← ~BSTK
Bit Stack Commands	
1.	Push 1 to BSTK. TOS <-- 1
0.	Push 0 to BSTK. TOS <-- 0
DUP	Duplicate the TOS bit. Copy the TOS bit and push to BSTK.
2DUP	Duplicate the TOS and TOS+1 bits and push them to BSTK.
DROP	Pop BSTK and discard the data.
F.	Set all eight BSTK bits to logic 1.
Z.	Clear all eight BSTK bits to logic 0.
!!	Complement all eight BSTK bits.
I/O Commands	
Ai.	Read input pin An (n = 0..5) and push to BSTK.
ALi.	Read ALi (n = 0..5) leading edge bit and push to BSTK.
ATi.	Read ATi (n = 0..5) trailing edge bit and push to BSTK.
ADi.	Read ADi (n = 0..5) double action bit and push to BSTK.
.Yn	Pop BSTK and write to output pin Yn (n = 0..7)
Un.	Read virtual bit Un (n = 0..7) and push to BSTK.
.Un	Pop BSTK and write to the virtual bit Un (n = 0..7)
Vn.	Read virtual bit Vn (n = 0..7) and push to BSTK.
.Vn	Pop BSTK and write to the virtual bit Vn (n = 0..7)

Flip-flop commands	
.Sm	Pop BSTK and write to the Sm (m = 0..f) F/F set input.
.Rm	Pop BSTK and write to the Rm (m = 0..f) F/F reset input.
Qm.	Read Qm (n = 0,,5) F/F output bit and push to BSTK.
Timer commands	
.TKm	Pop BSTK and write to the TKm (m = 0..f) timer trigger input.
.TRm	Pop BSTK and write to the Rm (m = 0..f) timer reset input.
Tqm.	Read TQm (n = 0..5) timer output bit and push to BSTK.
ldt	Display and edit the sixteen timer reload settings on screen. Commands: • ‘Z’ clears all timers • Cursor left and right keys selects timer • Enter and load new timer settings. • ‘CR’ returns to edit screen.
LDTm	Load accumulated counts from gated CNT counter to Tm (m = 0...f) timer buffer if valid CNT data ready, update Tm buffer byte image in EEPROM. If CNT has no valid data ready, do nothing.
Gated counter	
.CNT	Pop BSTK and write to the CNT gate input. On a high transition, enable counting of internal 1 s ticks. Stop counting when low transition detected, set data ready flag. Accumulated counts saved in buffer and retrieved by LDTm..
Toggle function	
.TGn	Pop BSTK and write to .TGn (n = 0..7) toggle input
TGQn.	Read TGQm (n = 0..) toggle output and push to BSTK.
Byte Commands	
PA.	Read input Port PA pins to BSTK.
PPA	Print Port PA bits in the Port Byte display area.

.PY	Write BSTK to the output Port PY pins.
PPY	Print Port PY bits in the Port Byte display area.
PU.	Read the virtual Port PU byte to BSTK.
.PU	Write the BSTK to the virtual Port PU.
PPU	Print Port PU bits in the Port Byte display area.
PV.	Read the virtual Port PV byte to BSTK.
PPV	Print Port PV bits in the Port Byte display area.
PBS	Print the BSTK in the BSTK display area. Use in RUN mode for program debugging.
Clock Signal Commands	
CLKA.	Square wave clock signal, 0.2 s pulse width, general purpose timing bit.
CLKB.	Square wave clock signal, 0.6 s pulse width, general purpose timing bit.
CLKC.	Square wave clock signal, 1.8 s pulse width, general purpose timing bit.
User File Load and Store Commands	
STORE	Writes the UF0 file image to the controller's internal EEPROM non-volatile memory. Timer values are saved along with the user file.
LOAD	Re-loads a previously stored UF0 file image from the controller's internal EEPROM non-volatile memory. Load also restores saved timer values.
User File List, Read and Write Commands	
LIST	Dump the contents of the UF0 space and timer settings buffer in RAM to the terminal screen in ascii readable format. Use Minicom log session (^AL) function to capture screen dump to a disc file on host PC.
WRITE	Dump the contents of the UF0 space and timer settings buffer in RAM to the terminal screen in Intel hex file format. Use Minicom log session (^AL) function to capture the screen dump to a disc file on host PC.
READ	Read incoming Intel hex file transmission of stored disc file from host PC. Use Minicom ascii file send (^AS) function to upload the disc file.

10 Application program examples

A collection of application program examples are presented to illustrate one method of writing application programs for uTile.

A logic diagram is created to define the requirements of the application program. The logic diagram makes it much easier to follow the control logic and it makes the task of designing, testing and debugging the application much easier.

The coding of the application using uTile commands to implement the control logic diagram is straightforward as the command words are designed to naturally describe the.logic functions used in the diagram.

The examples are based on use of the uTile test rig for simulating I/Os.

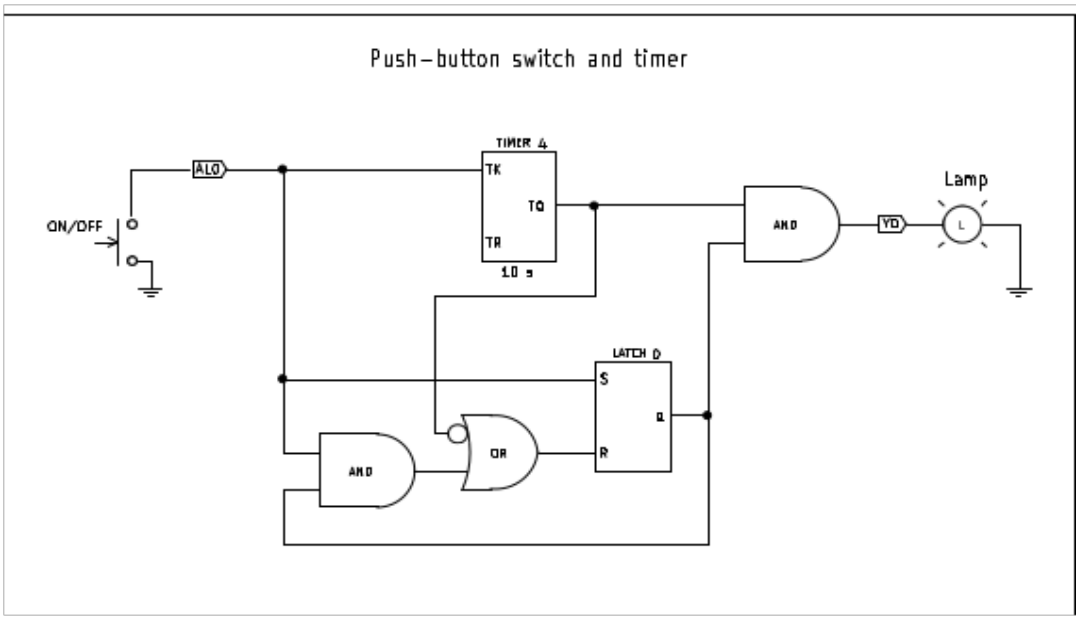
All of the examples include a logic diagram to illustrate the conceptual logic design of the application.

10.1 Push-button switch and timer

Description: Press switch to start timer. Press again before timeout to stop timer. When the switch is pressed the output turns on until the end of the timeout is reached, or if the switch is pressed again before the timeout ends.

Resources:
1 flip/flop
1 delay timer

A0 = Push-button switch, SW0
Y0 = Output, LED0
T4 = 5 s time delay



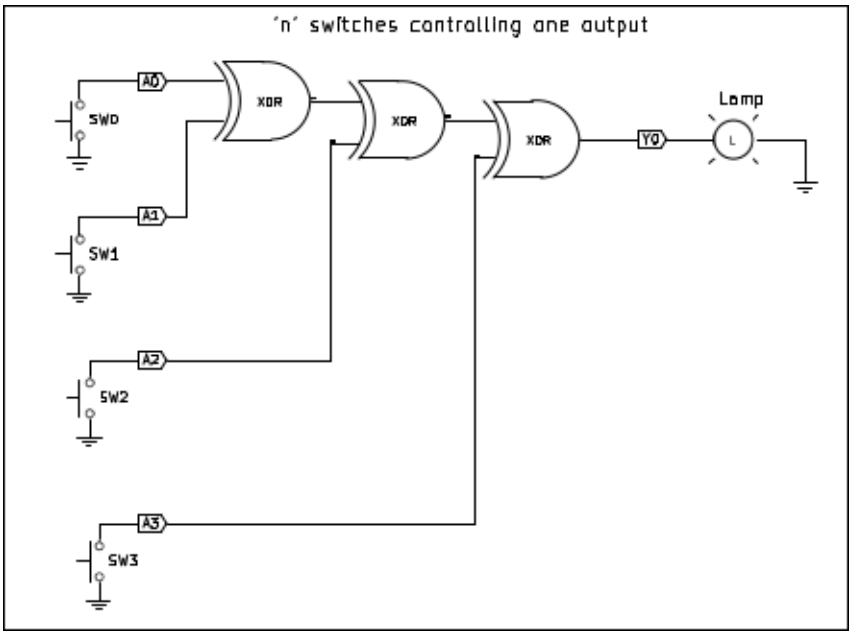

```
000  AL0.          008  !             016  END           024  NOP
001  DUP           009  OR            017  NOP           025  NOP
002  DUP           010  .R0           018  NOP           026  NOP
003  .TK4          011  TQ4.          019  NOP           027  NOP
004  .S0           012  Q0.           020  NOP           028  NOP
005  Q0.           013  AND           021  NOP           029  NOP
006  AND           014  .Y0           022  NOP           030  NOP
007  TQ0.          015  KEY           023  NOP           031  NOP
```

10.2 'n' switches controlling one output

Description: Changing the state of any input switch causes the output Y0 to change state. Useful for lighting control applications where multiple switches in different locations control one light.

A0 = spdt switch, S0
A1 = spdt switch, S1
A2 = spdt switch, S2
A3 = spdt switch, S3
Y0 = Output, LED0

The three cascaded XOR gates are the functional equivalent of a single 4 input XOR gate. This idea applies equally to AND and OR gates as well.




```
001  A0.          009  KEY          017  NOP          025  NOP
001  A1.          009  END          017  NOP          025  NOP
002  XOR          010  NOP          018  NOP          026  NOP
003  A2.          011  NOP          019  NOP          027  NOP
004  XOR          012  NOP          020  NOP          028  NOP
005  A3.          013  NOP          021  NOP          029  NOP
006  XOR          014  NOP          022  NOP          030  NOP
007  .Y0          015  NOP          023  NOP          031  NOP
```

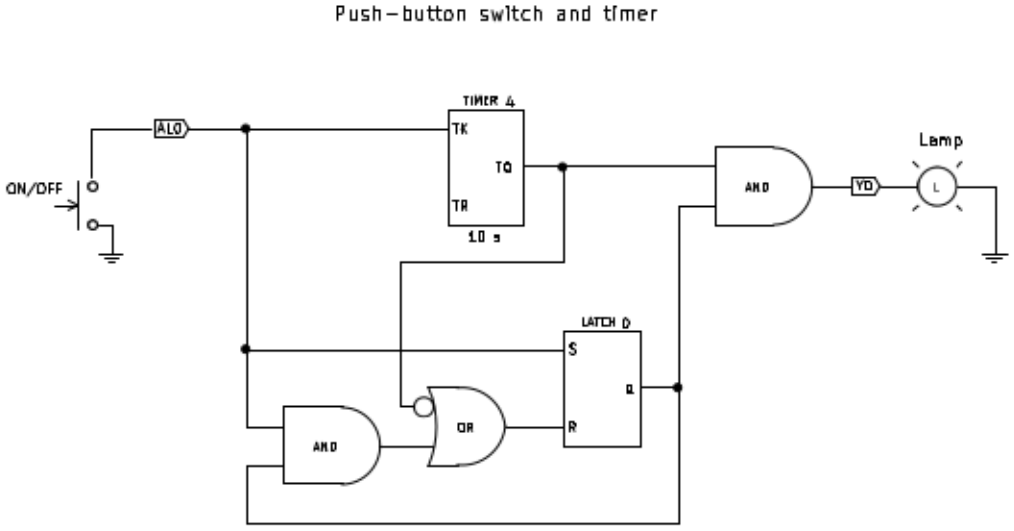
If the program is run on the test rig the PB switch has to be pressed down continuously to simulate the input from a closed toggle switch.

10.3 Push-button lighting control

Description: Push-button switch SW0 with local indicator LED3 toggles output LED0 on/off each time SW0 is pressed. LED3 flashes slowly when output is off. LED3 on full when output LED0 is on.

Useful for lighting control application where the light LED0 is remotely located and cannot be seen from the controlling SW0 its local indicator LED3.

A0 = push-button switch, S0
Y0 = Output, LED0
Y3 = Output, LED3



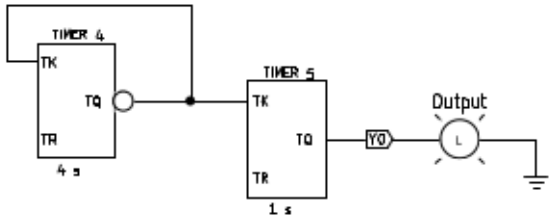

```
000  AD0.          008  .Y3           016  NOP           024  NOP
001  DUP           009  KEY           017  NOP           025  NOP
002  DUP           010  END           018  NOP           026  NOP
003  .Y0           011  NOP           019  NOP           027  NOP
004  !             012  NOP           020  NOP           028  NOP
005  CLKB.         013  NOP           021  NOP           029  NOP
006  AND           014  NOP           022  NOP           030  NOP
007  OR            015  NOP           023  NOP           031  NOP
```

10.4 Continuously running timers

Description: Run two cascaded timers continuously as a pulse generator. The first timer re-triggers itself and sets the pulse repitition rate. The second timer sets the output pulse duty cycle (controls the output pulse ON time).

```
Timers:
T4 = 4, Pulse generator repitition rate
T5 = 1, Pulse ON time
```

Continuously running cascaded timers




```
000  TQ4.          008  KEY           016  NOP           024  NOP
001  !             009  END           017  NOP           025  NOP
002  DUP           010  NOP           018  NOP           026  NOP
003  .TK4          012  NOP           020  NOP           028  NOP
005  .TK5          013  NOP           021  NOP           029  NOP
006  TQ5.          014  NOP           022  NOP           030  NOP
007  .Y0           015  NOP           023  NOP           031  NOP
```

10.5 Intrusion alarm system

Description: Press SW0 to arm alarm. T4 time delay before arming alarm to allow egress from protected area. LED0 indicator (Y0) flashes to indicate arm hold-off active, turns on fully at end of T4 to indicate alarm is now armed. During hold-off time opening of the field string does not trigger the alarm.

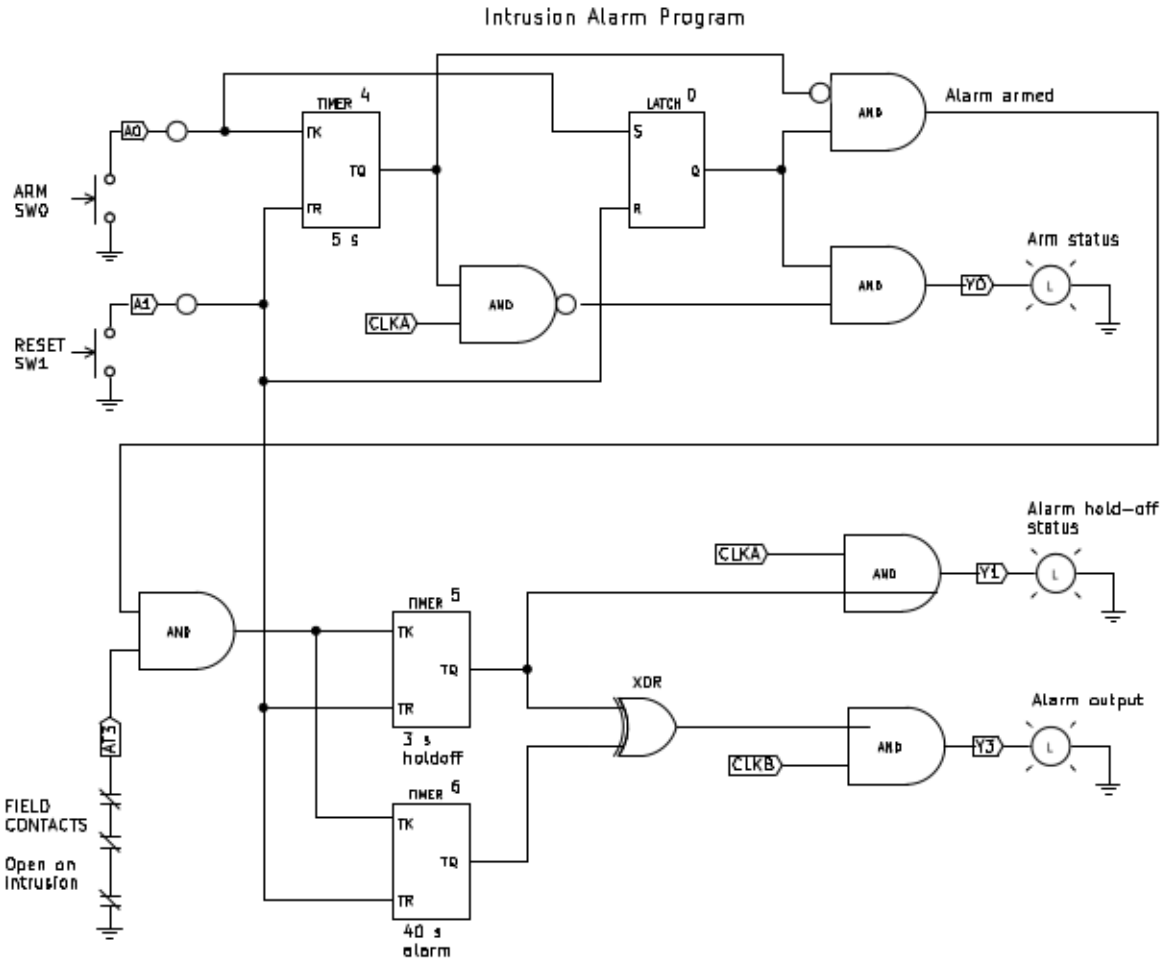
If field contact string is broken when the alarm is armed, T5 alarm hold-off delay starts to allow time for user to reset alarm system by pressing SW1. LED1 (Y1) flashes during alarm hold-off period. If SW1 reset is not pressed by end of T5 time delay, alarm is activated and alarm output goes on for T6 minus T5 time. Alarm turns off at end of T6 time.

Alarm system can be reset at any time by pressing SW1.

```
A0 = push-button switch SW0, Arm
A1 = push-button switch SW1, Reset
A3 = NC switches, series wired, open on alarm, field contact(s)
Y0 = Output, LED0, Arm status
Y1 = Output, LED1, Alarm hold-off status
Y3 = Output, LED3, Alarm output
```

Timers:

T4 = 5, Alarm arm delay
T5 = 3, Alarm hold-off delay
T6 = 40, Alarm timer



UF0 listing:

```
000  A0.     008  DUP     016  AND     024  AND
001  !       009  DUP     017  !       025  AT3.
002  DUP     010  .TR4    018  Q0.     026  AND
003  .S0     011  .TR5    019  AND     027  DUP
004  .TK4    012  .TR6    020  .Y0     028  .TK5
005  A1.     013  .R0     021  TQ4.    029  .TK6
006  !       014  CLKA.   022  !       030  TQ5.
007  DUP     015  TQ4.    023  Q0.     031  TQ6.
```

```
032  XOR     040  KEY      048  NOP     056  NOP
033  CLKB.   041  END      049  NOP     057  NOP
034  AND     042  NOP      050  NOP     058  NOP
035  .Y3     043  NOP      051  NOP     059  NOP
036  TQ5.    044  NOP      052  NOP     060  NOP
037  CLKA.   045  NOP      053  NOP     061  NOP
038  AND     046  NOP      054  NOP     062  NOP
039  .Y2     047  NOP      055  NOP     063  NOP
```

Timer settings:

T0	T1	T2	T3	T4	T5	T6	T7
00	000	000	000	005	003	040	000

T8	T9	TA	TB	TC	TD	TE	TF
000	000	000	000	000	000	000	000

10.6 Motor Start/Stop with O/L trip and lock-out

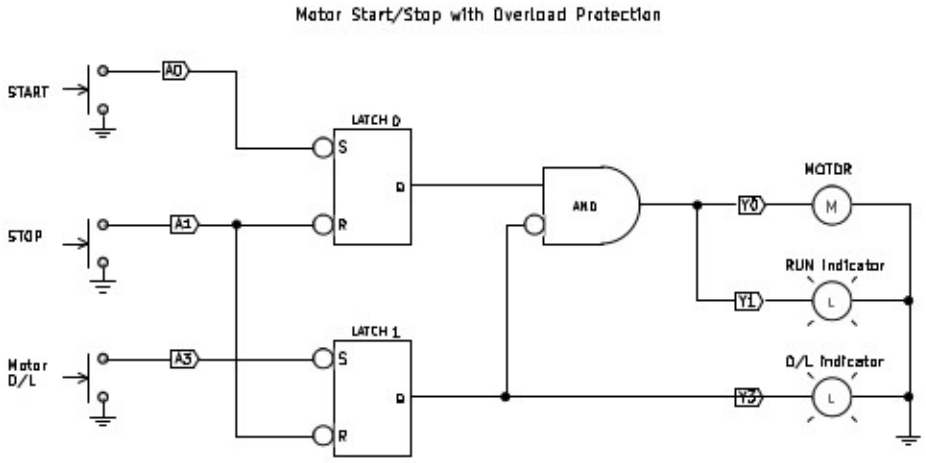
Description: Classic motor control circuit. Press SW0 to start motor, the Y0 local RUN indicator and Y1 motor load turn ON. Press SW1 to stop motor.

If the overload (O/L) input closes due a fault condition, the Y3 O/L indicator turns on and the motor is tripped and locked iout if it is running. The motor is prevented from starting as long as O/L is tripped.

The O/L is cleared by pressing SW1. Then SW0 can start the motor again.

A0 = Start switch, NO
A1 = Stop switch, NO
A3 = O/L switch, NO. In practice this O/L contact should be NC
Y0 = Run, indicator
Y1 = Motor load
Y3 = O/L indicator

The logic diagram for the motor starter circuit for simulation on the test rig is shown below. Note that in practice the O/L contact is NC for safety reasons. The inverter in the set input S1 om LATCH 1 would be removed if a NC O/L a3.switch contact is used.




```
000  A3.          008  A0.          016  .Y0          024  NOP
001  !            009  !            017  .Y1          025  NOP
002  .S1          010  .S0          018  Q1.          026  NOP
003  A1.          011  Q0.          019  .Y3          027  NOP
004  !            012  Q1.          020  KEY          028  NOP
005  DUP          013  !            021  END          029  NOP
006  .R0          014  AND          022  NOP          030  NOP
007  .R1          015  DUP          023  NOP          031  NOP
```

10.7 Lead/Lag sump pump control with LSHH alarm

Description: Two pumps used to drain a sump when the water level rises to a high level. Only one pump, the lead pump 1, operates the first time the sump level rises. The other pump is called the lag pump 2, and it operates the second time the level rises. The lead and lag pumps operate alternately, first the lead pump, then the lag pump, then the lead pump, etc.

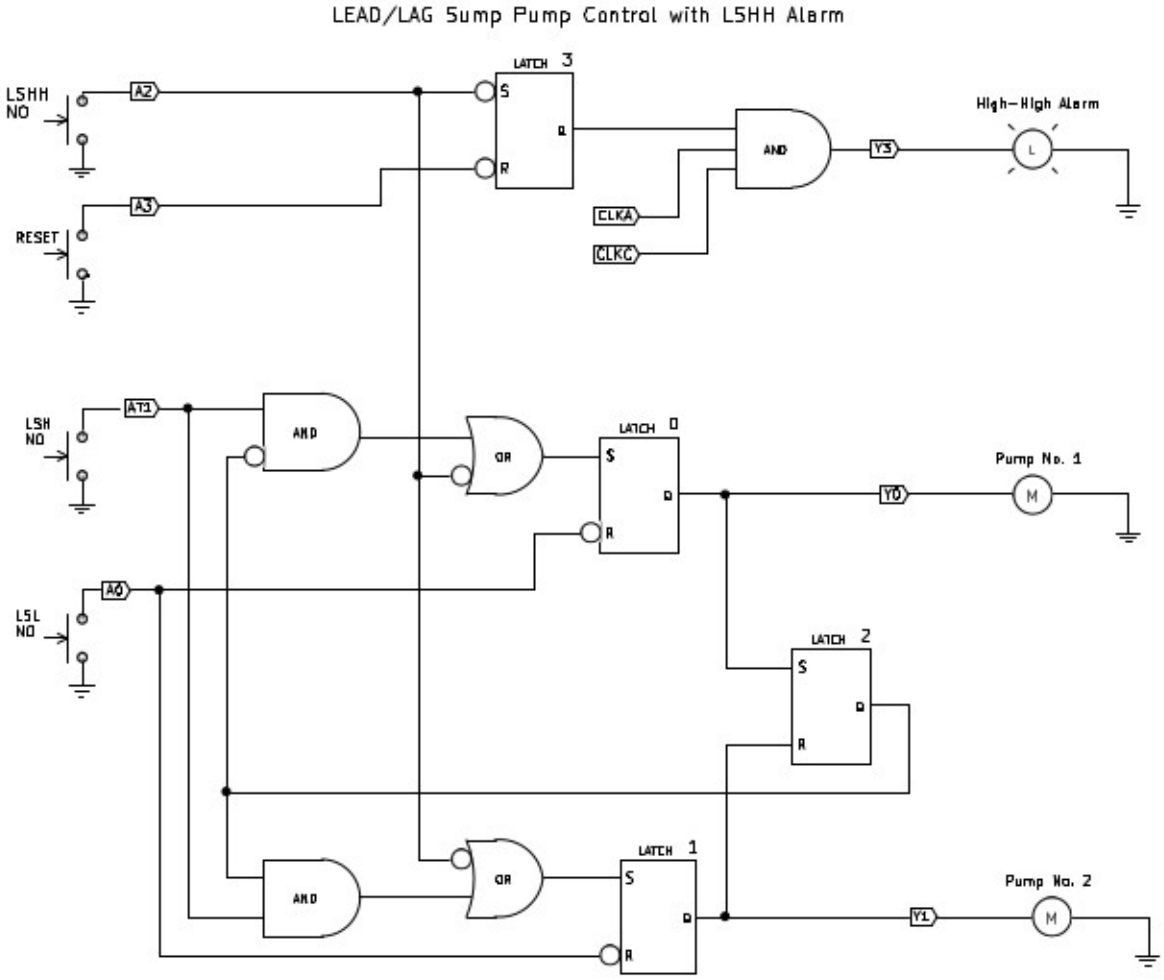
There are three control level switches that sense three levels in the sump. The low level switch activates when the sump level falls below the low level. The high level switch activates when the sump level rises above the high level. The high-high level switch activates when the sump level rises above the high-high level (when there is a pump failure, for example).

Input/output connections:

A0 = Low level switch, LSL, NO switch

A1 = high level switch, LSH, NO switch
A2 = High-high level switch, LSHH, NO switch
A3 = Alarm reset switch, NO switch

Y0 = Pump 1
Y1 = Pump 2
Y3 = Alarm indicator




```
000  A2.          008  AND          016  AND          024  !
001  !            009  Q3.          017  A2.          025  OR
002  .S3          010  AND          018  !            026  .S1
003  A3.          011  .Y3          019  OR           027  A3.
004  !            012  AT1.         020  .S0          028  !
005  .R3          013  DUP          021  Q2.          029  .R3
006  CLKA.        014  Q2.          022  AND          030  Q0.
007  CLKC.        015  !            023  A2.          031  DUP

032  .Y0          040  DUP          048  NOP          056  NOP
```

```
033  .S2          041  .R0          049  NOP          057  NOP
034  Q1.          042  .R1          050  NOP          058  NOP
035  DUP          043  KEY          051  NOP          059  NOP
036  .Y1          044  END          052  NOP          060  NOP
037  .R2          045  NOP          053  NOP          061  NOP
038  A0.          046  NOP          054  NOP          062  NOP
039  !            047  NOP          055  NOP          063  NOP
```

Operation:

System is turned on for the first time, sump level is below the LSL switch. As the sump starts to fill up with water the Level continues to rise until it passes the LSH switch. At this point pump 1 starts, draining water from sump, the sump level to fall.

The level keeps falling as pump 1 continues draining the sump, until level falls below LSL switch. At this point pump 1 is turned off.

Sump starts to fill up with water again, and level rises until it passes LSH setting again. At this point pump 2 starts and drains sump again. Sump level falls until it drops below LSL setting, which causes pump 2 to stop running.

The fill/drain cycles repeat with pump 1 and pump 2 operating alternatively. Each of the pumps operate for equal times so that pumps share operating duty.

Pump failure:

In the case where the pump that is supposed to run fails to start, the sump water level keeps rising past the LSH switch until the level passes the LSHH switch. At this point a water level high-high alarm is generated (Y3 is activated) and the alternate pump is started to drain the sump. The sump level falls until it passes below the LSL level switch, which stops the running pump. The system continues to operate in this failure mode but the alarm indicator remains actived until the alarm reset switch A2 is pressed.

Program simulation:

Start the program and begin <u>normal</u> cycle simulation:

a) Sump is empty, all LEDs are off. LED0 is Pump 1, LED1 is Pump 2
b) Sump level rises and activates LSH on A1. Press SW1 to simulate high level. LED0 switches on.
c) Sump drains down until LSL aon A0 ctivates. Press SW0 to simulate low level. LED0 switches off.

d) Sump level rises and activates LSH on A1. Press SW1 to simulate high level. LED1 switches on.
e) Sump drains down until LSL aon A0 ctivates. Press SW0 to simulate low level. LED1 switches off. Notice toggling between LED0 and LED1 in steps c) and d).

Start the program and begin abnormal cycle simulation:
a) Sump is empty, all LEDs are off. LED0 is Pump 1, LED1 is Pump 2
b) Sump level rises and activates LSH on A1. Press SW1 to simulate high level. LED0 (Pump 1) switches on. Assume Pump 1 fails to run and sump level keeps rising and activates LSHH on A2. Press SW2 to simulate high-high level. High-high alarm activates, LED3 starts flashing. LED2 (Pump 1) switches.
c) Sump drains down until LSL on A0 ctivates. Press SW0 to simulate low level. LED0 and LED1 both switch off.
d) Sump control program continues to operate with failure of either pumps. The high-high alarm is latched until it is reset by pressing SW3 to clear the alarm.

11 Minicom program

Minicom communications program provides the VT102 compatible terminal emulation for use as the uTile user interface.

11.1 Minicom installation and setup

Minicom is one of several communications programs that run under Linux.

The following description is for a Fedora Linux distribution but should apply for other distributions as well. The instructions presented here assume you are familiar with the basic procedures for installing software packages on your system.

Install the minicom package on your system using the ***#dnf install minicom*** command. If you are using another flavor of Linux, use the package management tools that comes with the distribution to install minicom.

11.1.1 Minicom setup

To run Minicom as a normal user it needs to be setup first. Minicom runs in a terminal and it does not have a GUI. The setup is done with root priviliges then the system wide configuration is saved. After saving the configuration you should run Minicom as a normal user.

Plug in a USB cable to connect the Nano board to one of the PC's USB ports and type in the command ***ls -l /dev/tty* | grep dialout*** to show a list of serial ports belonging to the group dialout***.*** You should see something like this:

```
ls -l /dev/tty* | grep dialout
crw-rw---T 1 root dialout 166,  0 Sep 27 08:53 /dev/ttyUSB0
crw-rw---T 1 root dialout   4, 64 Sep 27 04:04 /dev/ttyS0
crw-rw---T 1 root dialout   4, 65 Sep 27 04:04 /dev/ttyS1
crw-rw---T 1 root dialout   4, 66 Sep 27 04:04 /dev/ttyS2
```

The first line of the listing shows that /dev/ttyUSB0 is assigned to the Nano board serial connection and the device belongs to the dialout group.

Run minicom with root priviliges with the following command:

sudo minicom -s

This drops you into the minicom configuration screen. Select Serial port setup. Enter A to select the Serial Device and change it to /dev/ttyUSB0. Enter E to set the Comm Parameters and use the A or B choices to adjust the baud rate to 19200. Select 8-N-1. Turn off hardware and software flow control.

Setup the ascii file send parameters by typing ^AZ to call up the Command Summary, then type O to select the configuration menu, then select the File transfer protocols. In the menu screen showing the protocols, confirm there is line for the ascii transfer write protocol:

```
I  ascii    /usr/bin/ascii-xfr -sv      Y    U    N       Y       N
```

Press J to add a line for the ascii transfer read protocol and enter the following:

```
J  ascii    /usr/bin/ascii-xfr -rv      Y    D    N       Y       N
```

Exit the menu, and save the setup as a .dfl file, then exit minicom.

Type ***id -Gn <username>*** command and see if you belong to the dialout group or not. If not, add yourself to the dialout group with the command ***sudo usermod -a -G dialout <username>***. Logout and login again to effect the changes. Issue the command ***id -Gn <username>*** again and confirm you now belong to the dialout group. Minicom is now setup to run with uTile.

11.1.2 Basic commands for running Minicom

To start a minicom session in a terminal, enter ***minicom*** on the command line. The minicom terminal should start up and display uTile main screen.

To quit and exit a minicom session type the following:

'^AX'

That is, press the 'CTL' key and 'A' key at the same time, release both keys then press the 'X' key. A message pops up asking if you wish to leave minicom. Select YES to exit.

To call up minicom's help screen, type '^AZ' to get a Command Summary list.

To start and close a Minicom log session, type ‘^AL’.

You can view the minicom manual page by typing ***$man minicom*** in a terminal screen.

12 WARRANTY

The m328-uTile program is not intended for use in commercial, industrial, medical or safety-related applications. No liability is assumed for use of the m328-uTile program.

THERE IS NO WARRANTY FOR THE m328-uTile PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. THE PROGRAM IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL WE BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM, EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.