uTile Programmable Logic Controller for the Atmega328 microcontroller

Instruction Manual

Revision 0
Date: 2019 July 17

Copyright © 2019 - Francis Lyn

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, architecture and theory of operation. Utile is designed to run on the Microchip Atmega328 microcontroller.

uTile is an acronym for “Micro Threaded Interpretive Language Engine”.

Guidelines are presented for setting up a uTile system and preparing the development environment for creating and testing uTile application programs. A VT100 compatible terminal emulator provides the user interface (UI) to communicate 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 command set is described in detail and illustrated by numerous examples.

1.1 Product description

The uTile PLC consists of a Microchip ATmega328P microcontroller with the uTile executable image m328-nano-utile.hex flashed to the chip’s program memory. This manual describes uTile for the Arduino Nano controller board platform. Utile can be used on other controller boards that use the Atmega328 microcontroller.

1.2 Highlights

- Fully functional, ready to run PLC for small automation projects.
- Fourteen general purpose digital inputs (6) and outputs (8).
- Sixteen one-shot programmable delay timers.
- Sixteen flip-flops.
- Fast and efficient bit stack architecture.

- Complete set of control logic commands, AND, OR, XOR etc.
- Integrated programming editor and controller command screen via VT100 terminal emulator running on host PC.
- Large user file space, 256 program lines.
- Store and Load user programs and delay timer settings to EEPROM.
- Write and Read user applications to and from host PC disc file.
- USB connection to VT100 terminal user interface.
- Full open source code on github repository.
- Directly interfaces to popular Arduino I/O shields, relay modules and custom I/O circuits.
- Autostart stored user program on power-up or board reset.

1.3 Overview

uTile is an application program written in assembly language and designed to run on a Microchip ATmega328P microcontroller. The uTile binary executable image is first flashed to the ATmega328P program memory. Thereafter, when the board is powered-up or reset, the uTile program begins execution and the microcontroller operates as a complete and fully functional high performance PLC.

User access to uTile PLC is via a USB serial port communicating with a VT-100 type terminal emulator running on a host PC. The uTile user interface consists of a main command screen which also doubles as an integrated programming editor for entering the uTile program commands to the ATmega328P chip's internal RAM. Connection of the terminal interface is not required for executing a stored uTile application program.

uTile performs incremental execution of each command immediately after the command is entered, in a similar way Basic and Forth programming language instructions are handled. Each uTile command is first interpreted to a command vector, the command vector is executed. The command vector is stored sequentially in the uTile program space (the User File, or UF) in RAM if the command is an executable command. Command vectors are pointers to assembly language routines, and are always executed at full speed. Commands that affect the interpreter operation only are not stored in the UF. An example of an interpreter only command is the "?" command to display the uTile help screen.

A PLC program consisting of one or more command vectors stored

sequentially in the User File (UF) via the programming editor are executed at full speed. Once the interpreter process identifies and loads a command vector to the UF there is no need to pass through the interpreter again when commands in the UF are executed in the ***run*** mode. The interpreter action during the command entry process essentially 'compiles' the uTile command instructions as command vectors in the UF.

uTile supports permanent non-volatile storage of a UF image in non-volatile EEPROM. Timer settings are also saved in EEPROM when the UF is stored. In addition, the UF and Timer settings can be saved/restored to/from an exteranl disc file on the host PC (currently supported only using Kermit communications program).

1.4 Introduction to uTile

uTile is a special purpose controller designed to replace the hard wired control logic found in typical control systems. A hard wired control system is made from traditional discrete hardware components like relays, counters and timers, panels, sockets, terminal blocks and a lot of interconnection wiring to tie these components together.

The effort to design, build, install, commission and maintain a hard wired control system is not trivial, and the system is costly to implement. Performing maintenance and diagnostic procedures on a hard wired system is also time consuming and difficult. Making modifications or changes to the control logic necessitates changes to the hardware and the physical wiring of the system, a laborious, time consuming and costly process at best.

In contrast to a hard wired control system, uTile uses software defined control logic to implement a control system. Software instructions, consisting of commands taken from the uTile's dictionary of command words, are strung together in a programmed sequence and stored as a user application program, or User File (UF). When this UF is executed by uTile, it performs the prescribed control logic functions, easily replacing a panel full of discrete components and internal wiring.

Using a uTile control system provides many significant benefits to the user. Lower costs, significant reduction in construction and

commissioning time, flexibility, testability, ease of performing maintenance and diagnostic procedures, and most importantly, the ease and speed of making modifications and additions to the control system. uTile provides a wide range of advanced logic functions such as one-shot timers, flip-flops and input signal pre-processing (including input de-bouncing, transition inputs and toggle inputs).

The use of uTile in automation and control applications simplifies the task of creating workable controller solutions for real world applications by removing the burden of writing, testing and debugging code for microcontroller hardware peripheral functions from the ground up. Writing such code demands a good understanding of the controller hardware as well as experience in developing the associated software. Utile allows you to focus on your automation application requirements and avoid getting bogged down in the low level machine details.

1.5 The uTile Application Program

uTile applications are far easier and more efficient to implement compared to the traditional hard-wired logic implementation for several good reasons. uTile uses a powerful descriptive language that describes the logic requirements in a natural manner.

The user can more easily understand and construct logic expressions in the uTile language, compared to using the traditional relay logic symbols and ladder logic schematic diagrams. Documentation effort and readability are improved by use of uTile's more intuitive and efficient descriptive language.

uTile emulates the control logic application entirely in software. Any modifications or changes to the control logic are reduced to changes to the software description (the application program) that defines the logic emulation. Software program changes are much faster and easier to implement compared to physical changes to a hard-wired circuit comprised of discrete components and interconnection wiring.

Several popular PLC architectures allow users to program an application using a variety of schemes, such as ladder logic notation, function block notation, or logic statement notation. Each

architecture has pros and cons, the best choice depends on the size and complexity of the application and on the knowledge, experience and ability of the user.

The uTile instruction set is based on logic notation statements and it is particularly adept at describing boolean logic expressions that are at the core of all control applications. The uTile instruction set consists of English like command words that easily describe logic expressions. Programming uTile is intuitive, easy and efficient.

2 Putting the system together

This section describes the required hardware and software components to assemble a uTile system and shows how to connect the parts together. uTile interfaces to a variety of input and output devices/modules to suit the needs of individual applications.

We will use the Nano controller board throughout the manual as a typical uTile implementation example. Bear in mind that uTile is flexible and the source code allows configuration to run on various members of the Atmega microcontroller family. For example, uTile has been ported to run on the Atmega2560 controller with expanded I/O (16 DI and 16 DO), two UF spaces (UF0 and UF1), and 16 Flip-flops. As another example of the flexibility afforded, uTile has been configured to run on a single Atmega328P device operating on the internal 8 Mhz oscillator, resulting in a true single chip PLC.

The Nano controller board running the m328-uTile program provides direct access to the ATmega328P chip port pins. All input/output devices connected to the Nano port pins must comply with the ATmega328P port pin voltage and current specifications.

Please refer to the ATmega328P device data sheet for the technical specifications and ratings. Because of the low power capability of the device port pins, most applications will require the addition of suitably rated I/O buffer devices to connect to the external sensors and loads on your project.

2.1 Hardware configuration

A typical fully functional uTile system configuration requires the following items:

1. Arduino Nano board.
2. m328-uTile binary application image flashed to ATmega328P program memory.
3. Optional relay module board with on-board relay drivers.

4. Isolated 7 to 12 V dc power supply for powering the Nano controller board.

Notes on uTile system power supply.

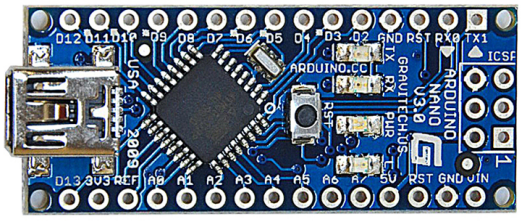
1. The Nano board can be powered in two ways:
   - USB connector 5 V power.
   - X1 connector (7 to 12 V dc).

2. The Nano board has an on-board low dropout voltage regulator that supplies a regulated +5 V dc to the board as well as to external loads connected via the board’s +5 V and GND terminals.

3. External loads can be powered from the Nano board's +5 V and GND connections. It is highly recommended to power exteranl loads from a separate and independent supply to minimise the risk of load induced noise getting through to the controller.

4. External loads connected to the Nano output pins, such as LEDs or relay module inputs, are powered from the Nano's +5 V regulated supply. Please ensure you don't overload the Nano's supply with externally connected loads.

Typical hardware items are shown below.



2.2 Software tools

A VT100 type terminal emulator program running on an external computer provides the user interface to the uTile PLC. The terminal display and keyboard interface is used for entering, testing and modifying the user's application program.

The open source Tera Term software under BSD license, available at http://ttssh2.sourceforge.jp/index.html.en, has the required terminal features for communicating with the Arduino Nano running the m328-nano-uTile PLC program. Tera Term runs under various Windows versions.

Other terminal emulator programs can be used, for example PuTTY, Minicom or C-Kermit programs under a Linux OS. See 6.Software Terminal Emulator Programs for details on Minicom and C-Kermit installation.

2.3 Connecting the pieces together

A basic configuration is described below, without the optional relay module connected. This minimum configuration is used to check the uTile PLC hardware for proper operation. The Nano's on-board LED (silk-screen label **L**), may be used for an output port pin status indicator. You need to connect input devices like push-button switches to the input port pins if you wish to test the inputs.

The basic configuration allows you to program and execute the full complement of uTile PLC commands.

We strongly recommend the wiring-up of a test rig consisting of at least two switches and several LEDs (with current limiting resistors wired in series) and connected to the I/O pins of the Nano board. See Figure 1 for a schematic diagram.

A test rig will allow you to easily write, test and debug your application programs and is highly recommend to have as part of your set of development tools.

2.4 uTile Input/Output Test Rig

The schematic diagram in Figure 1 below shows a test rig consisting of four discrete switches (NO push-button switches) 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 example programs presented later on in the manual use the test rig switches to simulate digital signals to uTile input pins and uses the LEDs to monitor the status of the uTile output pins.

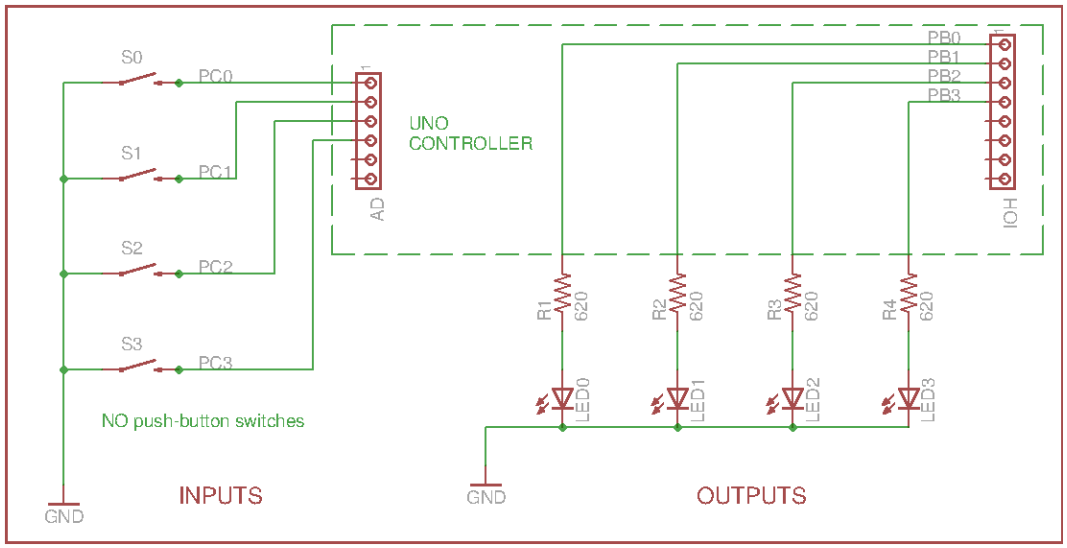


Figure 1 - uTile Test Rig

Update note:
The output port in has been remapped from port PC to port PD. Replace PB0..PB3 by PD2..PD5.

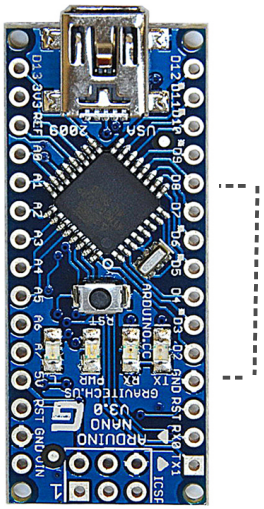
2.4.1 uTile Program Installation

The m328-nano-utile.hex binary program image has to be programmed to the Atmega328P flash memory on the Arduino Nano board in order to run the uTile PLC. The complete source code and documentation for the uTile PLC is on the github repository:
https://github.com/lynf/ATmega328-uTile-PLC

Please refer to the **uTile Program Installation Guide** for details on flashing the m328-nano-utile.hex program image to the Nano board. Once the flashing operation is completed the uTile PLC is ready to run. Connect the Nano board to a host PC as shown below to start running uTile.

USB Cable

VT100 terminal emulator running
On PC provides the user interface
to uTile



Add jumper D8 to GND if
Auto-start of user application
after board power up is required.

Start the VT100 terminal emulator on the PC and set up and save the serial port connection parameters as follows:

- Serial port setup parameters -

Port:	COMx (COM port assigned to Nano). *
Baud rate:	19200
Data:	8 bit
Parity:	none

Stop: 1 bit
Flow control: none

* Windows uses COMx port, Linux uses /dev/ttyUSB0.

The uTile main screen will appear on the terminal emulator if the USB serial communications connection to the Nano is sucessfully established. The terminal user interface gives you complete access to the programming and operation of the m328-uTile PLC.

2.5 First Time uTile Power-up

When power is first applied to the Nano board the following screen appears:

```
                  <<<<  Nano - u T I L E  -  ATmega328  >>>>
--------------------------------------------------------------------------------
          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 ***
```

If the screen appears the basic connections are correct, the Arduino Nano is communicating correctly with the host PC terminal via the USB port and the m328-uTile program is ready to accept user commands at the Enter: prompt.

If you don't see the screen appear it means the uTile PLC is not communicating with the host PC. In this case, you need to check the connections between the Nano board and the host PC and fix any problems.

2.6 The uTile Terminal Screen Elements

There are four main elements shown on the uTile terminal screen, 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 display area and 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 specifying a delay timer setting and for specifying the program line number target for the editor jump operation.

The main display area shows four columns of eight rows of program lines (a display page of 32 program lines) of the UF0 space. There are eight display pages covering the 256 program lines of the UF0 space. The Atmega328 has only one User File space, UF0. Devices with larger RAM space, such as the Atmega256, support two user file spaces, UF0 and UF1. You can scroll around the UF0 space with the cursor arrow keys, or jump to any line in the program as described in the next section.

2.7 uTile Operating Modes

uTile has two operating modes, the EDIT mode and the RUN mode. The EDIT mode is entered when uTile is first powered up, or when uTile is reset. The RUN mode can be entered from the EDIT mode, or from a power up or reset condition if the auto-start input on port pin PB0 (D8) is tied to ground.

2.7.1 uTile EDIT Mode

The EDIT mode in uTile is indicated by message '*** User File 0 ***'

appearing on the status line on the main screen. The 'Enter:' field is active and accepts user command and data input entries. The input line editor supports <BS> and <^X> keys which erases the current line entry.

The Left and Right arrow keys flips between program pages, while the Up and Down arrow keys step backwards or forwards through the page display. Entering a new command at any line number overwrites the existing command with the newly entered command. Entering a number in the range of 0 through 255, then pressing the Home key will move the program line cursor to the program page showing the line number entered.

2.7.2 uTile RUN Mode

A program stored in the UF0 area can be executed at full speed by invoking the ***ex0*** and/or ***run*** commands. These commands start program execution and uTile enters the RUN mode which is indicated by the message '*** Running User File Program ***' appearing on the status line.

2.7.3 Entering commands

uTile is not case sensitive, commands entered in upper and lower case are treated in the same way.

2.7.4 Entering Numerical Data Values

Some operations require entering numerical data. Decimal numbers in the range of 0 through 65556, ending with a <CR>, can be entered at the 'Enter:' prompt. uTile only uses numbers in the range of 0 through 255 for moving the cursor to a specified line number and for entering timer reload values.

3 Basic Programming Examples

This chapter introduces a number of uTile commands that are essential for editing, testing and debugging application programs.

The first programming example will show how to use the uTile editor to enter commands. This manual shows all commands in boldface type characters. Executable commands are shown in upper case while Interpreter commands are shown in lower case.

3.1 Example 1 — fill, NOP, END and ex0 Commands

At the Enter: prompt, type the following command followed by a Carriage Return (<CR>) key. All commands are entered in this way, and the command entry ends with a <CR> key.

Enter: ***fill*<CR>**

The following screen appears.

```
                  <<<<  Nano - u T I L E  -  ATmega328  >>>>
--------------------------------------------------------------------------------
          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 ***fill*** command replaces the five dots on each program line with

NOP words. Initially following first application of power to the Nano board, the User File space (UF0) is filled with random data. uTile does not recognize this random data as valid and displays the 5 dots. The ***fill*** command fills the entire UF0 file space with the NOP word, which is the "NO OPERATION" or do nothing command. Use of the ***fill*** command is not mandatory, but it is recommended good practice to initialize the UF0 with NOP words before starting to write any program.

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

```
                  <<<<  Nano - u T I L E  -  ATmega328  >>>>
--------------------------------------------------------------------------------
          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.

```
                  <<<<  Nano - u T I L E  -  ATmega328  >>>>
--------------------------------------------------------------------------------
           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 the special ***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 last line of the program. The ***END*** command resets uTile's program counter to line 000 and causes execution to continue from line 000 once again.

You can type in the ***END*** command explicitly on the last line of your program. You start the program execution by entering the ***ex0*** command to put uTile into the execution mode. After a running program is stopped for any reason, use the ***ex0*** command to start executing the program again.

Halt the example 1 running program by pressing the Nano board's reset push button.

3.2 Example 2 — Run Command

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

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

```
                  <<<<  Nano - u T I L E  -  ATmega328  >>>>
--------------------------------------------------------------------------------
          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 executed, uTile did two things. It first inserted the ***END*** command at the cursor position where the ***run*** command was entered (line 000), and then it started executing the program commands stored in UF0 beginning at line 000. To see the actual program that was executed, press the reset push button to restart uTile and the page display now shows that an ***END*** command exists at line 000.

3.3 Example 3 — KEY command

In example 2, after the UF0 program is started with the ***run*** command, the only way to interrupt the program is to press the Nano reset switch. A more elegant way to interrupt a running program is with the ***KEY*** command.

The following screen appears after the example 2 program is interrupted by pressing the reset switch.

```
                  <<<<  Nano - u T I L E  -  ATmega328  >>>>
--------------------------------------------------------------------------------
          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 ***
```

Notice the following:

1) The highlighted cursor returned to program line 000. On program line 000, at the ***END*** command, type in ***KEY<CR>*** at the Enter: prompt. The ***KEY*** command is executed, then stored at program line 000 and the cursor advances to line 001. Notice that the ***END*** command is over-written by the ***KEY*** command.

Start the new program by typing ***run<CR>***. The user program begins execution, and the screen looks like this:

2) There is an ***END*** command at line 000, it was automatically

inserted by the ***run*** command when it was entered in example 2.

```
                  <<<<  Nano - u T I L E  -  ATmega328  >>>>
--------------------------------------------------------------------------------
          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 ***
```

Once again, the status line changes to indicate the stored User File 0 is running.

The ***KEY*** command inserted at line 000 allows you to break-out of a running program and return to the uTile main screen by pressing the **/** key. You can always stop a running program by pressing the reset switch on the Nano, but ***KEY*** is more elegant and the reset switch may not be readily accessible.

Press the **/** key is to break out of the running program. The program halts and the screen looks like:

```
                  <<<<  Nano - u T I L E  -  ATmega328  >>>>
--------------------------------------------------------------------------------
          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 uTile main screen.

The ***fill, NOP, END, run, ex0*** and ***KEY*** commands are used extensivley in uTile programming and we have spent a bit of time discussing their usage and behaviour in the above examples.

3.4 Example 4 - The help '?' command

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

```
                  <<<<  Nano - u T I L E  -  ATmega328  >>>>
--------------------------------------------------------------------------------
           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             Interpreter      Timers
  -----   ---------    --------------------------    ------------------   ------
   and       set          In:  Ai. ALi. ATi. ADi.       fill  ex0   run     TQm.
   or        clr         Out: .Yn                       ins   key   end     .Tkm
   xor       dup        Virt:  Un., .Un, Vn., .Vn       del    /    ver     .TRm
    !        drop    (i = 0..5, n = 0..7, m = 0..f)     ?   load  store     ldt
   nop    f., z., !!                                        read  write

  ------- Byte Commands -------      - Flip/Flop -     --- Clock Pulses ---
   PA.  .PY   PU.  .PU  PV. .PV     .Sm  .Rm   Qm.     clka.   clkb.  clkc.
   PPA  PBS   PPU  PPV  PPY                           200 ms  600 ms 1.8 s
  Ground D8 to enable Autostart                  Timers: T0..T3 (100 msec)
  Ground D9 to invert outputs                              T4..Tb (sec)
  Date: 2019/12/09                          Tc..Te (min), Tf (6.66 min)

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

The screen shows a list of all the uTile commands for programming and editing.

3.5 uTile commands — Executable and Interpreter commands

Each time you enter a command, uTile executes the command immediately. uTile then stores the command in the UF0 file space if it is an ***executable*** type of command. Examples of ***executable*** types of commands are the ***NOP*** and ***END*** commands. If it is an ***interpreter*** type of command (e.g. ***fill*** or ***ex0***), it is not stored in the UF0. An ***interpreter*** type command cannot be included in a user program.

The next chapter of the manual covers the architecture of the uTile logic engine and describes the uTile resources available for use in application programs.

4 uTile Architecture

This section provides an introduction to uTile logic engine which is at the heart of the uTile programmable logic controller.

4.1 uTile Logic Engine

The m328-uTile program operates on top of a logic engine called ***uTile***, an acronym for “Micro Threaded Interpretive Language Engine”. Throughout the manual, the uTile logic engine will be referred to simply as uTile for brevity. Although programming uTile is straightforward and intuitive, a basic understanding of how the engine works is necessary for creating successful user application programs.

4.2 A brief description of the uTile features

The uTile logic engine is a fully functional programmable logic controller with a user file (UF0) space of 256 command words. uTile uses a command line entry and display screen (uTile main screen) for entering commands and numerical settings.

uTile operates in two modes, the EDIT mode and the RUN mode. The EDIT mode allows a new user program to be entered, or an existing program to be modified. Entered programs can be stored to non-volatile EEPROM for later recall or for automatic execution following a reset. An entered command is executed once by the interpreter and the command vector is stored in UF0 space if the command is an executable type. Non-executable type commands, or interpreter commands, are executed when entered, but are not stored in the UF0 space.

In the RUN mode the user's program stored in the UF0 space is run continuously at full execution speed. The ***ex0*** command invokes the Run mode from the EDIT mode.

The next topic of discussion introduces the uTile architecture and the concept of the uTile bit stack. The use of bit and port labels to

refer to the uTile devices such as input/output ports and pins, timers, flip-flops and virtual storage are covered.

4.3 uTile Architecture — the Bit Stack

The central feature of the uTile architecture is the ***bit stack BSTK,*** an 8 bit data structure with a last-in-first-out storage mechanism. Logic data bits, such as the six input bits, Flip/Flop output bits, timer output bits and virtual byte bits can be pushed, or written, to the bit stack.

Logic operations can be performed on data stored on the bit stack and the results of the operations stored back on the bit stack. Data from the bit stack can be popped, or read, from the bit stack and sent to another destination such as an output port pin or to a virtual byte bit.

All data transfers operations use the bit stack as the source or destination operand.

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 (LIFO) data structure.

Bit Stack	Bit label	Comments
0	[TOS]	← Top-of-stack (TOS)
1	[TOS+1]	
2	[TOS+2]	
3	[TOS+3]	
4	[TOS+4]	
5	[TOS+5]	
6	[TOS+6]	
7	[TOS+7]	← Bottom-of-stack (BOS)

Data is pushed onto the top of the stack (TOS) by a logic operation that fetches a data bit. The push operation moves all bits in the stack down by one bit position; bit 0 moves to bit 1, bit 1 moves to bit 2, and so on. Bit 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 of the data bit is specified by the operation that invoked the push. If you push more that eight data bits on the stack before you pop data bits off, you will lose data beyond the latest eight bits stored.

Data is popped off the stack from the TOS. The pop operation moves all bits in the stack up by one bit position; bit 0 is stored in the destination specified by the command that invoked the pop. Bit 1 moves to bit 0, bit 2 moves to bit 1, and so on. A zero moves into bit position 7.

You can view the bit stack operations as commands are executed in the EDIT mode by observing the bit stack display.

Logic operations can be performed on the data bits that are pushed on the stack. When a logic operation is performed on two operands on the stack, the operands are popped from the stack, the logic operation is performed, and the results is pushed back on the stack. The operations are performed in “Reverse Polish” notation (RPN) as opposed to the more commonly known but less intuitive “Algebraic” notation. In RPN notation, the operands are first pushed on the stack, then the operation to be performed is specified. The result of the specified operation is then pushed on the stack.

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

Logic commands such as ***AND***, ***OR*** and ***DUP*** uses two operands from the bit stack (the ***TOS*** and [***TOS+1***] bits), performs the logical operation on the two bits, and pushes the single result bit back to the TOS. The two source bits are consumed from the stack by these commands.

4.3.1 uTile Ports and Bit Labels

uTile has access to devices such as input and output ports, flip-flops, delay timers, virtual storage and internal clock signals. Port and bit labels are used to refer to the various devices.

The term “port” refers to a uTile device, such as the input port ***PA***, the output port ***PY***, or the virtual storage location ***PU***. A full device port has eight bits, but in some cases less than eight bits may be in use.

The port label consists of one or more letters, for example ***PY, PA, PU, TK, S, R, CLKA***, etc.

uTile operates on binary data at the bit level. To facilitate access at the bit level, a set of unique bit labels are assigned to each bit of every device port. A bit label consists of a portion of the port label plus a number between 0 and 7 specifying the particular bit in the port byte. For example, the bit labels associated with port ***PY*** are ***Y7, Y6,...,Y0***.

uTile can operate on byte sized data as well in some cases. uTile commands use the port labels for byte level operations. 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***
Delay Timers: ***T*** Trigger input ***TK*** Reset input ***TR*** Timer output ***TQ***	 ***TKf,...,TK0*** ***TRf,...,TR0*** ***TQf,...,TQ0***
Flip-Flops: Set input ***S*** Reset input ***R*** Output ***Q***	 ***Sf,...,S0*** ***Rf,...,R0*** ***Qf,...,Q0***
Clock signals	***CLKA, CLKB, CLKC***

Table 1 - uTile Port & Bit Labels

4.3.2 uTile '.' Command Notation

uTile bit stack operations involve the bi-directional transfer of data between the bit stack ***BSTK*** and a uTile device. The bit stack is implicit in every data transfer operation, meaning that the bit stack is always acting either as a source or as a destination operand.

uTile commands use a '.' symbol as a prefix or postfix to a bit or byte label to signify if the bit stack is acting as a source or destination operand.

A bit label with a postfix '**.**' indicates the bit stack is the destination operand of the data transfer. For example, the command ***A0.*** reads the port bit ***A0*** state and moves it to the TOS.

A prefix '**.**' and a bit label indicates the bit stack is the source operand of the data transfer. For example, the command ***.Y0*** writes the TOS to the ***Y0*** port pin.

4.4 uTile Input/Output Port Pins

uTile has six digital inputs and eight digital outputs. The input/output (I/O) pins are directly connected to the ATmega328P port pins via the connectors on the Nano board.

Input port pins are scanned regularly and pre-processed before they are used in the logic processing stage. 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 signal, such as leading edge and trailing edge transitions, are captured and saved by the input pre-processor and stored in appropriate bit labels for access by the user program as needed.

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

Output port pins are updated regularly from information stored in the output pin bit label storage area. The output port pin drive characteristics and ratings are specified in the ATmega data sheet in the section on Pin Driver Strength.

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

4.4.1 uTile Input Port PA

uTile supports one input port ***PA***, with six input port pins ***A0, A1, A2, A3, A4,*** and ***A5***.

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.

An. ***[TOS]*** ← ***An***, where n = 5,...,0

uTile'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:

ALn. ***[TOS]*** ← ***ALn***, where n = 5,...,0 (Leading edge)

ATn. ***[TOS]*** ← ***ATn***, where n = 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:

ADn. ***[TOS]*** ← ***ADn***, where n = 5,...,0 (Double action)

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

PA. ***BSTK*** ← ***PA***

Since ***PA*** is only six bits wide, only six bits of the ***PA*** are transferred to ***BSTK***.

The programming examples section of the manual shows how to read the input port byte and bits.

4.4.2 uTile Output Port PY

uTile supports one output port ***PY*** with eight output port pins ***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 pin.

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

The uTile command to write the ***BSTK*** to the six output pins 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.

4.4.3 uTile Virtual Port PU and PV

uTile supports two virtual ports ***PU*** and ***PV***. Since both ports are identical in operation, the following description is for ***PU*** only, but it applies equally as well to ***PV***.

A virtual port ***PU*** is provided for storing intermediate results. The virtual bits of this port byte are individually addressable and support read and write operations. The virtual bit labels are ***Un***, where n = 0,..,7 (the bit number). The bi-directional transfer between the port byte and the bit stack is also 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 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 ***PU*** are as follows:

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

The uTile command to write the ***BSTK*** to ***PU*** is:

.PU ***PU ← BSTK***

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

4.4.4 uTile Delay Timers

Sixteen delay (one-shot) timers are available. The first four timers count in 100 milliseconds, the next eight timers count in seconds, the following three timers count in minutes, and the last timer counts in increments of 6.66 minutes. 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 Section 3.3.1 above, Table 1, uTile Port & Bit Labels. The timer label subscript ***m*** ranges from 0,1,...,d,e,f.

The delay 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 resets at the end of the timing cycle.

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.

4.4.5 uTile Flip-Flops

Sixteen set-reset flip-flops (F/F) are used for storing momentary 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 standard logic elements, the flip-flops are easy to understand and use.

The bit labels associated with the flip-flops are listed in Table1 of Section 3.3.1 above.

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

Sm	Rm	Qm
0	0	*
0	1	0
1	0	1
1	1	0

* no change in output state

5 uTile 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 Figure 1, uTile Test Rig is a recommended way for providing the input stimulus signals and for monitoring the resulting output signals.

The program code shown in the examples are by no means the only solution to problem, there may well be several ways to achieving the same result.

5.1 Example 5 – Reading/writing uTile I/Os.

An extensive set of commands for reading input port pins are provided to allow extraction of additional signal characteristics (leading and trailing edge detection and toggling data) that are useful in control applications. Sections uTile Input Port PA and uTile Output Port PY describe the set of input/output commands while the following program examples illustrate the use of these commands.

Enter and ***run*** the following program in the UF0 space:

```
                  <<<<  Nano - u T I L E  -  ATmega328  >>>>
-------------------------------------------------------------------------------
          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 1 1              00000
-------------------------------------------------------------------------------

      000  A0.          008  KEY          016  NOP          024  NOP
      001  .Y0          009  NOP          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
```

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

Don't forget to type in the ***run*** command when the cursor is at line 009 to insert an ***END*** command and start executing the program.

Line 000 reads in the state of S0 switch and stores the result on the ***TOS***. The normally open (NO) S0 switch pulls the ***A0*** input low when the switch is activated.

Line 001 transfers the ***TOS*** bit to output pin ***Y0***, so LED0 reflects the state of input switch S0. Similarly, the states of the other switches are read and the results are transferred to the associated LEDs. Observe the LEDs while you press any combination of the switches. Notice the LEDs turn off on when the switches are activated.

Break out of the running program by pressing the **/** key and clear the UF0 space using the ***fill*** 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 bit on the TOS before it is written to the output as follows:

```
                   <<<<  Nano - u T I L E  -  ATmega328  >>>>
--------------------------------------------------------------------------------
          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 1 1              00000
--------------------------------------------------------------------------------

      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




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

Now when the input switch is activated, the associated LED goes on. See if you can rewrite the program to change the input to output assignment 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 available byte operations.

Enter and ***run*** the following program:

```
                <<<<  Nano - u T I L E  -  ATmega328  >>>>
-------------------------------------------------------------------------------
          Bit Stack                                          Port Byte
        7 6 5 4 3 2 1 0              Data                7 6 5 4 3 2 1 0
        ===============              =====               ===============
        1 1 0 0 0 0 0 0              00000               0 0 1 1 1 1 1 1
-------------------------------------------------------------------------------

       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  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 ***
```

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

The ***PPA*** command is optional and mostly used during program development and debugging 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 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 uTile Input Port PA for the specific bit labels.

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

```
000  AT0.        008  NOP         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  Q0.         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 program uses the trailing edge transition bits on switches S0 and S1 to set and reset a flip-flop, while the flip-flop's output is monitored on LED0.

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

Line	Command	Action
000	AT0.	S0 trailing edge, high when S0 pressed
001	.S0	F/F0 set when S0 pressed
002	AT1.	S1 trailing edge, high when S1 pressed
003	.R0	F/F0 reset reset when S1 pressed
004	Q0.	F/F0 output to TOS
005	.Y0	TOS to LED0
006	KEY	Program break-out

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 S0 or

S1 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 once for every two changes 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 all possible truth tables into logic expressions.

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

5.3.1 Example 8 – Logic AND command

This example illustrates the logic ***AND*** operator. Enter and ***run*** 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  NOP         015  NOP         023  NOP         031  NOP
```

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 writes the result on the ***TOS*** to output pin ***Y0***. Pressing S0 and S1 turns on the LED0 indicator.

5.3.2 Example 7 — Logic OR command

This example illustrates the logic ***OR*** operator. Enter and ***run*** 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 ***!*** commands in lines 001 and 003 inverts the signals read in from S0 and S1 so that the values on the ***BSTK*** are 1 when the switches are pressed. The switches are NO types, so the inputs read 0 when the switches are pressed.

Line	Command	Action
000	A0.	Read S0 to TOS
001	!	Complement TOS
002	A1.	Read S1 to TOS
003	!	Complement TOS
004	OR	TOS <-- TOS OR TOS+1
005	.Y0	Write TOS to Y0, LED0
006	KEY	Program break-out

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 writes the result on the ***TOS*** to output pin ***Y0***. Pressing either S0 or S1 turns on the LED0 indicator. Pressing both S0 and S1 at the same time also turns on the LED0 indicator.

5.3.3 Example 9 — Logic XOR command

This example illustrates the logic ***XOR*** operator. Enter and ***run*** 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  NOP          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 writes the result on the ***TOS*** to output pin ***Y0***. Pressing S0 or S1 turns on the LED0 indicator. Pressing both S0 and S1 at the same time turns off the LED0 indicator. It is this last behaviour that distinguishes the ***XOR*** from the ***OR*** operator.

5.4 Example 10 — Bit Stack commands

Sevearl bit stack commands are provided for programming, diagnostics and trouble-shooting. Many of the commands are useful for testing the m328-uTile outputs. Look at the help screen under Bit Stack you see the following commands listed:

Bit Stack	Operation
set	Set top of stack (***TOS***) bit
clr	Clear ***TOS*** bit
dup	Duplicates ***TOS*** bit and push to ***BSTK***
drop	Drops ***TOS*** bit from ***BSTK***
pbs	Print ***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 ***SET*** command. The ***TOS*** bit is now 1.
Enter the ***DUP*** command. The ***TOS*** and next bit are now 1's.
Enter the ***DROP*** command. The ***TOS*** bit is popped off the ***BSTK***.
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 except for the ***fill*** command, which is an editor only command.

The next example illustrates how to manually control an output pin while in the editor mode, using uTile's incremental execution mode. Output pin ***Y7*** also drives the Nano on-board LED "**L**", which serves as a convenient output indicator.

First enter a ***Z.*** command to clear the ***BSTK*** followed by a ***fill*** command to clear the UF0 space. Essentially you wish to send a logic 1 bit from the ***TOS*** to the output ***Y7***. Enter the following commands in the sequence shown and observe the bit stack display and LED "**L**":

Line	Command	Action
000	SET	***TOS*** <-- 1
001	.Y7	***Y7*** <-- ***TOS***

To turn off output pin ***Y7*** you need to send a logic 0 bit from the ***TOS*** to the output ***Y7***. The ***TOS*** contains 0 after the line 001 command is entered and executed (data popped from ***BSTK***).

Turn off LED "**L**" with the following command:

002	.Y7	***Y7*** <-- ***TOS***

5.5 Example 11 — Virtual Byte commands

The virtual storage byte ***PU*** is a general purpose storage buffer with associated byte and bit commands for read and write access. The ***PU*** byte is used for storing data that is used later on in an application program. Enter and ***run*** 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  NOP          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 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***.

5.6 Internal Test Clock Signals

Three general purpose internal clock signals are provided for convenience in alarm signal generation and test applications.

The three clock signals are accessed by commands ***CLKA.***, ***CLKB.*** and ***CLKC.*** The clock signals are square wave outputs with 100 ms, 600 ms and 1800 ms pulse width durations. The clock period is twice as long as the clock pulse.

5.6.1 Example 12 — Clock Test Signals

These examples illustrates the use of the clock signals in alarm signal generation and for trouble-shooting the outputs. Enter and ***run*** 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  NOP          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 reads in the ***CLKA***, ***CLKB*** and ***CLKC*** signals to the ***BSTK*** then writes them to outputs ***Y0***, ***Y1*** and ***Y2*** so you can observe the signals on the test rig LEDs.

Program lines 006 through 009 reads in the ***CLKA*** and ***CLKC*** signals, logical ***AND*** the signals and writes the result to output ***Y3*** so you can observe combined signal on LED3. This technique is useful for providing a more pleasant alarm signal for driving external audio/visual alarm annunciators.

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

5.7 Setting the Timers – ldt command

The sixteen programmable delay (one-shot) timers are set up with the ldt command. See uTile Time Delay Timers for timer details. Before using the timers in your application, they must be loaded with the desired delay time values. The first 4 timers (0,1,2,3) are 100 milliseconds, the next 8 timers (4,5,..b) are seconds, the following 3 timers (c,d,e) are minutes timers, and the last timer (f) counts in units of 6.66 minutes.

Following a cold start power-up, the timers contain random data and need to be initialized to their desired values before being used. You can clear all the timers by entering ‘Z’ key at any time.

Type the ***ldt*** command at the Enter: prompt to load the delay timer settings. The following screen appears:

```
                  <<<<  Nano - u T I L E  -  ATmega328  >>>>
--------------------------------------------------------------------------------
           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. a **<CR>** key only.

After a warm start power-up or 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 and later restored using the ***read*** and ***write*** uTile commands are also preserved.

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.

5.8 User File storage on disc file

uTile supports storage and retrieval of a UF image and timer settings to a disc file on a host PC. The ***read*** and ***write*** commands handle the file transfer from uTile. The disc file transfer procedure is handled by two scripts that are invoked from the communications program providing the user interface functions for uTile.

In order to make use of disc file storage feature, the communications program must be able to support scripting for controlling the disc file transfer operations. The C-Kermit program has the requisite script capability and is suitable for use with uTile. Please refer to the section Installing C-Kermit for a description of the file transfer scripts.

5.9 Displaying the current program version

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

6 Software Terminal Emulator Programs

Several Open-source terminal emulator programs are available for free that can be used for uTile's UI to provide the screen display and keyboard input for uTile's program editor. uTile communicates with a DEC VT100 type terminal emulator via the USB serial interface.

If you are running a *nix (Unix-like) OS like Linux, FreeBSD etc, the Open-source Minicom communications program works well with uTile and is usually available from the distribution's software repository.

The C-Kermit 9.0 communications program works very well as a UI for uTile and the powerful command and script language offers the added advantage of automating the file transfer procedure that allows uTile to store and retrieve User File application programs to and from the host PC. For this reason, C-Kermit is the preferred communications program for uTile's UI.

The Kermit 95 Windows OS equivalent version of C-Kermit is unfortunately not Open-source and therefore it is not available for free.

If you aren't able to use C-Kermit for whatever reason you can use any of the other recommended terminal emulator programs and get a fully functioning uTile UI except for the file transfer functions. The file transfer functions are nice-to-have but are not essential for using uTile.

6.1 Installing C-Kermit

C-Kermit is available under an Open-source license and runs under *nix operating systems. You can usually get the application from your distribution's package repository.

The following description is for a Debian 8 Linux distribution. The instructions presented here assume you are familiar with the procedure for installing software packages on your system. Install the C-Kermit program package. Download the three files .mykermrc, download.cmd, upload.cmd from the web site http://www.plcchips.com/home.html. and copy them to your /home/(your-

user-name)/ directory.

The file .mykermrc is an initialization file to configure kermit for use with uTile. This file sets up the USB device port to /dev/ttyUSB0, baud rate to 19200, 8N1 and other housekeeping chores, then opens the connection to the terminal interface to uTile.

The download.cmd file is a kermit script file that automates the procedure for getting a User's application program from uTile and storing an image of the UF0 contents on the host PC.

The upload.cmd file is a kermit script file that automates the procedure for sending a User's application program image stored on the host PC as a disc file back to uTile's UF0 program area.

Plug a USB cable into the PC and Nano board USB socket and start kermit by opening a terminal screen on the PC and invoking the command 'kermit' followed by a <CR> on the PC. If kermit is installed and operating properly you should see the uTile sign-on screen appear as described in Section 2 above.

Please visit the Kermit Project web site http://www.columbia.edu/kermit/index.html for further information on C-Kermit.

6.1.1 Basic commands for running Kermit

Kermit is a comprehensive communications package with a lot of commands but as we are only using a very small sub-set of the features we only need to know a few basic commands. When you connect the Nano controller to the PC with the USB cable and start kermit, the .mykermrc initiliazation file opens the user interface terminal to uTile and puts you at the main editor screen.

To quit the uTile main screen and return to kermit's command line prompt type the following:

'^\c'

That is, press the 'CTL' key and '\' key at the same time, release

both keys then press the 'c' key. Kermit closes the terminal interface and drops you back at the kermit command line prompt:

(/home/[your-user-name]) C-Kermit>

You can invoke script files from the command line and enter whatever other command you wish to invoke. To open the terminal interface to uTile from the kermit prompt, type the following command:

connect

uTile's main editor screen appears. You can just type 'c' instead of 'connect' for this command.

Whenever you wish to quit kermit from the kermit prompt, type 'q' followed by a <CR>. Kermit closes the USB port and exits back to the terminal screen from which it was started.

6.1.2 Using the DOWNLOAD.CMD script file

This script file is invoked at the kermit command line prompt to get uTile to send to the host PC an image of the contents of the user program that is currently in UF0 file space. The image is stored as a disc file on the PC for later sending back to uTile.

uTile should be at the main screen and your application program and delay timer settings should already be entered. Escape from uTile back to the kermit command line by entering the '^\c' command. When you are at the kermit command line prompt, enter the following command:

'download myfile.hex' followed by a carriage return <CR>.

The 'myfile.hex' is any name you wish to put on the incoming file. The file is saved in Intel Hex file format. If you don't provide the script a filename, a default name of utile.hex is used. The hex file is an ascii readable file so you can view the file with a text editor, but you won't see the uTile instructions as they appear in the editor's UF0 page display. Instead what you will see is the binary data stored in the UF0 space in ascii hexadecimal format.

6.1.3 Using the UPLOAD.CMD script file

This script file is invoked at the kermit command line prompt to get the host PC to upload to uTile a hex file that was previously saved by the download.cmd script. The uploaded file is loaded to uTile's UF0 file space and the saved timer data is restored.

uTile should be at the main screen. Escape from uTile back to the kermit command line by entering the '^\c' command. When you are at the kermit command line prompt, enter the following command:

'upload myfile.hex' followed by a carriage return <CR>.

The 'myfile.hex' is the name of the file you wish to upload. Use the name of whatever file you wish to upload. If you don't provide the script a filename, a default name of utile.hex is used. A 'File not found!' error message is shown if the file does not exist, then the script will exit back to the kermit prompt.

If the file exists, it will be uploaded to uTile. When the file transfer is completer the message 'File read completed - '/' for menu -->' is shown. Enter a '/' and uTile's main screen will appear.

6.1.4 Kermit File Permissions

*nix (Unix-like) operating systems require users to belong to a file's group before access to the file is granted.

Communications programs like kermit and minicom require users to join the 'dialout' group before they are granted access to the serial device being used by the communications program, i.e. /dev/ttyUSB0, /dev/ttyACM0, /dev/ttyrfcomm0 etc.

Check if you belong to the 'dialout' group by looking at the output of the 'groups' command typed in at the command line in a terminal session. If you don't see 'dialout' in the list of groups you need to add yourself to this group by issuing the following command:

sudo usermod -a -G dialout your_user_name

Issue the 'groups' command again to confirm 'dialout' appears in the list of groups you belong to.

You may need to change the permissions for ***/dev/ttyUSB0*** with the command ***sudo chmod a+rw /dev/ttyUSB0***, and do the same for ***/dev/ttyACM0*** if you need to access this device for your selected controller board. You only need to do this once for each device.

6.2 Installing Minicom

Minicom is one of several communications programs that run under Linux that can be used with uTile.

The following description is for a Debian Linux distribution. The instructions presented here assume you are familiar with the procedure for installing software packages on your system. Install the minicom program package from the distribution's package repository.

Install the minicom package on your system using the ***#apt-get install minicom*** command or by using Debian's Synaptic package manager. If you are using another flavor of Linux, use their package management tools to install minicom.

6.2.1 Minicom configuration

To run Minicom as a normal user it needs to be configured first. Minicom is runs in a terminal as it does not have a GUI. To run Minicom you must first open up a terminal screen. The configuration instructions are done with root priviliges, then the configuration is saved. You can then run Minicom with normal user priviliges.

Root commands issued in the command line below are indicated by the # prefix while user commands are prefixed by ***$***.

Plug in the 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/ttyACM0
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
crw-rw---T 1 root dialout   4, 67 Sep 27 04:04 /dev/ttyS3
```

In a root terminal, log in to minicom with the following command: **#*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/ttyACM0. Enter E to set the Comm Parameters and use the A or B choices to adjust the baud rate to 19200. You should select 8-N-1. Turn off hardware and software flow control. Save the setup as a .dfl file, then exit minicom.

You should run minicom as a normal user and not as root. Next, type **#*id -Gn <username>*** and see if you belong to the dialout group or not. If not, add yourself to the dialout group with the command **#*usermod -a -G dialout <username>***. Logout and login again to effect the changes, and issue the command **#*id -Gn <username>*** again to confirm you now belong to the dialout group.

You now need to change the permissions on /usr/bin/minicom to allow you to execute minicom as a normal user by issuing the command **#*chmod g+s /usr/bin/minicom***.

Change the permissions for ***/dev/ttyUSB0*** with the command **#*chmod a+rw /dev/ttyUSB0***, and do the same for ***/dev/ttyACM0*** if you need to access this device for your selected controller board. Make sure the Nano board is connected to one of the PC's USB ports and start minicom by typing ***$minicom***. The minicom terminal should start up and the uTile main screen appear.

A nice feature of minicom is the ability to save several different user configuration files for different setups in the home directory, giving each configuration file a meaningful name.

For example if you are using another Arduino board which uses a different USB port, say /dev/ttyACM0 instead of the default /dev/ttyUSB0, and you wish to save this configuration as .minirc.acm you can do this by copying the system-wide configuration to your home partition, edit it to change the line from:

pu   port   /dev/ttyUSB0

to:

pu   port   /dev/ttyACM0

Change the file ownership to your username and give it 644 permissions.

If you wish now to start minicom using the modified configuration settings, just enter ***$minicom acm***.

6.2.2 Basic commands for running Minicom

If minicom fails to connect to uTile for whatever reason and you need to change the configuration setup, start minicom in the setup mode by typing ***$minicom -- setup*** in a root terminal. Any corrections or changes to the system-wide configuration file can then be saved.

To quit the uTile main screen and return to the terminal screen from which minicom was started, 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.

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

7 uTile Nano pinouts

7.1 Port PA mapping

uTile's input port ***PA*** is mapped to the Nano controller 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

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

7.2 Port PY mapping

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

Outputs	Port pin	Nano pin
Y0	PD2	Digital 2
Y1	PD3	Digital 3
Y2	PD4	Digital 4
Y3	PD5	Digital 5
Y4	PD6	Digital 6
Y5	PD7	Digital 7
Y6	PB4	Digital 12

Outputs	Port pin	Nano pin
Y7	PB5	Digital 13

uTile has two special purpose dedicated control 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.

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

Any UF0 program stored in internal EEPROM with the ***store*** command can be started automatically after a power-up or reset condition by grounding the auto-start input line D8. By using this feature, uTile will start executing the UF0 application without the need for a terminal interface and manual start with the ***ex0*** command.

7.3 The autostart input on D8

Any UF0 program stored in internal EEPROM with the ***store*** command can be started automatically after a power-up or reset condition by grounding the auto-start input line D8. By using this feature, uTile will start executing the UF0 application without the need for a terminal interface and manual start with the ***ex0*** command.

7.4 The out_sense input on D9

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 condition. 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.

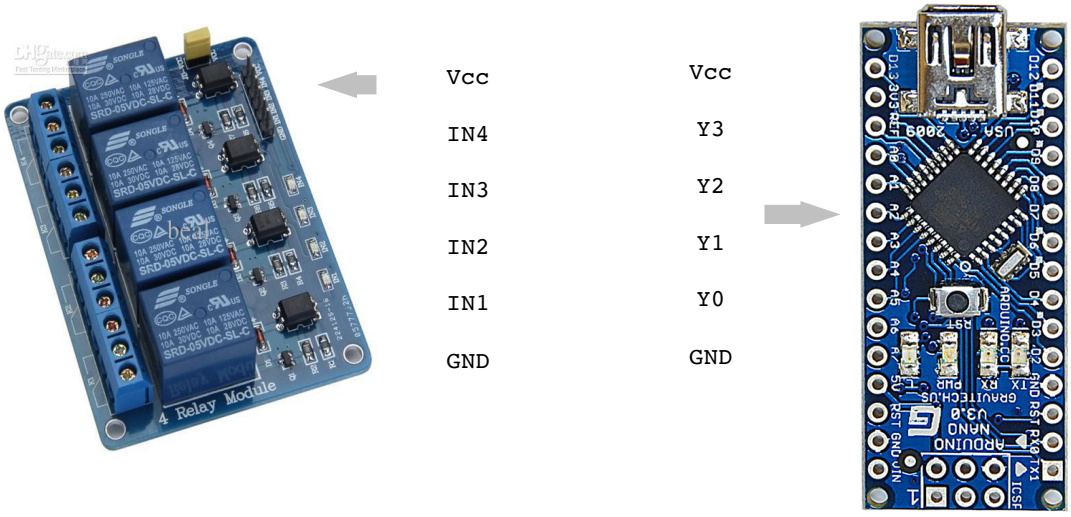
Note that this method is superior to inverting the ***PY*** bits using uTile program commands as signal inversion takes place only after the UF0 program begins executing and not before.

8 Adding a Relay Module to the Nano Board

uTile applications that control high power loads such as motors, inductive loads like solenoids or motor starters require a suitably rated buffer device between the Nano and the load. Relay modules offer a rugged, reliable and economical buffer solution.

Relay modules with opt-isolators on the input are recommended. See Hardware configuration for a typical relay module of this type. Specifying relays with 5 V coils can simplify power supply requirements.

The inputs to the relay module board are designed to be directly compatible to the digital output port of the Nano. Here is a typical interconnection diagram for the Nano and relay module board:



Connect the ground terminals on the Nano and relay module boards together. We recommend providing an adequately rated power supply with decent bypass capacitors. Separate power wiring from the power supply to the relay module board helps to minimize coupling any voltage spikes from switching relays back to the Nano.

The relay modules come with either active-low or active-high type inputs. An active-low input requires a low signal to switch on the relay, while an active-high input requires a high signal to switch on the relay. Since by default the m328-uTile application drives the

output to a high state when it is switched on, active-high input relay modules are simpler to use and are preferred over the active-low type of modules. If you must use an active-low type module, a few additional commands can invert the output signals from uTile to meet the input requirements of the module.

8.1 Using a Relay Module with Active Low Inputs

When a relay module with active-low inputs is connected to the Nano board the relays are energized when the output pins are in the off state (logic 0).

A uTile output pin is at logic 0 when the output is in the off state, so in order to drive active-low relay inputs, the output signal needs to be inverted before sending it to the relay input.

uTile can perform this signal inversion on a byte basis with the ***!!*** command, or on a bit basis with the ***!*** command.

8.2 Example 13 — Driving Active-low Relay Module

Enter and ***run*** the following code snippet:

```
000  PA.          008  NOP          016  NOP          024  NOP
001  .py          009  NOP          017  NOP          025  NOP
002  KEY          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
```

Line	Command	Action
000	PA.	***BSTK*** <-- ***PA***
001	.PY	***PY*** <-- ***BSTK***, output to relays
002	KEY	Program break-out

Line 000 reads in ***PA*** to the ***BSTK***. As the inputs are from NO pusk-button switches, the inputs are low when the switches are pressed so

the switch values read in are of the correct polarity for driving active-low input relays. Pressing a switch turns off the associated indicator LED and energizes the associated relay. To invert the relay drive signals, enter and ***run*** the following code snippet:

```
000  PA.          008  NOP          016  NOP          024  NOP
001  !!           009  NOP          017  NOP          025  NOP
002  .PY          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
```

The ***!!*** command in line 001 inverts the ***BSTK*** contents before writing it to the output port ***PY***.

Enter and ***run*** the following code snippet:

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

The ***!*** command in line 001 inverts the A0 input on the ***TOS*** before writing it to the output ***Y0***. In contrast, the ***A1*** input is not inverted before it is written to the output ***Y1***. These examples show how easy it is to change a signal's polarity using uTile commands.

9 Command Summary

The m328-uTile command set is summarized below. Commands are case insensitive (commands may be entered in upper or lower case).

Interpreter Commands	
?	Print a help screen.
/	Redraw the main screen in EDIT mode .
INS	Insert a 'NOP' command at the current program line position. The command on line 255 drops off the end of the user file 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.
RUN	Inserts an END command and begins execution of user program in UF0 by invoking the EX0 command. Short cut to typing END and EX0 commands.
EX0	Execute UF0 program.
FILL	Fill current UF with NOP commands.
VER	Display current version number
LDT	Load delay timer settings
Special program control commands	
KEY	Provides break-out from running program and returns to the EDIT mode. Enter a **/<CR>** to invoke the program break-out.
END	Marks the end of the user's program in UF0. An END command must be placed at the end of the user's program.
Logic Commands	
AND	Logic AND of two bit variables. The bit variable operands are popped from the TOS and [TOS+1]. The logic ***and*** of the two bits is computed and the result is pushed back to the TOS. TOS ← TOS & [TOS+1]

OR	Logic OR of two bit variables. The bit variable operands are popped from the TOS and [TOS+1]. The result is pushed back to the TOS. TOS <-- TOS V [TOS+1]
XOR	Logic exclusive OR of two bit variables. The bit variable operands are popped from the TOS and [TOS+1]. The result is pushed back to the TOS.
!	Logic complement. The bit variable operand is popped from the TOS, complemented, and pushed back to the TOS. TOS <-- ~TOS
NOP	No operation. Execution continues at the next program line.
Bit Stack Commands	
SET	Logic 1 bit is pushed to the TOS. TOS <-- 1
CLR	Logic 0 bit is pushed to the TOS. TOS <-- 0
DUP	Duplicate the TOS variable. Copy the TOS bit and push it to the bit stack.
DROP	Pop TOS bit and discard the data.
F.	Set all eight bits in bit stack to logic 1.
Z.	Clear all eight bits in bit stack to logic 0.
!!	Logic complement bit stack. All eight bits of the bit stack are complemented.
I/O Commands (Bit Commands)	
An.	Read the state of the input pin An, n = 0,...,7, and push the value to the bit stack.
ALn.	Read leading edge transition bit, push to TOS.
ATn.	Read trailing edge transition bit, push to TOS.
ADn.	Read double action (toggle) bit, push to TOS.
.Yn	Pop the TOS bit and write to output pin Yn, n = 0,...,7
Un.	Read the state of the virtual bit Un, n = 0,...,7, and push the value to the bit stack.

.Un	Pop the TOS bit and write to the virtual bit Un, n = 0,...,7
Byte Commands	
PA.	Read the state of input Port PA pins to the bit stack.
PPA	Show the Port PA bits in the Port Byte display area.
.PY	Write the bit stack byte to the output Port PY pins.
PPY	Show the Port PY bits in the Port Byte display area.
PU.	Read the virtual Port PU byte to the bit stack byte.
.PU	Write the bit stack byte to the virtual Port PU.
PPU	Show the Port PU bits in the Port Byte display area.
PBS	Show the bit stack bits in the bit stack display area.
Clock Signal Commands	
CLKA.	Square wave clock signal, 0.4 s period, general purpose timing bit. This command reads an internally generated clock signal that changes state every 100 ms and places it on the TOS.
CLKB.	Square wave clock signal, 1.2 s period, general purpose timing bit. This command reads an internally generated clock signal that changes state every 600 ms and places it on the TOS.
CLKC.	Square wave clock signal, 3.6 s period, general purpose timing bit. This command reads an internally generated clock signal that changes state every 1.8 s and places it on the TOS.
Flip/Flop commands	
.Sm	Pop the TOS bit and write to the F/F set bit Sm, m = 0,...,f
.Rm	Pop the TOS bit and write to the F/F reset bit Rm, m = 0,...,7
Qm.	Push the F/F output bit Qm, m = 0,...,f, to the TOS.
Delay Timers commands	
.TKm	Pop the TOS bit and write to the timer trigger bit TKm, m = 0,...,f

.TRm	Pop the TOS bit and write to the timer reset bit TRm, m = 0,...,f
TQm.	Push the timer output bit TQm, m = 0,...,f, to the TOS.
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 Read and Write Commands	
WRITE	Writes the UF0 file image to the host PC as a disc file. Timer values are saved along with the user file. The WRITE command is invoked by the kermit script DOWNLOAD.CMD
READ	Reads a previously stored UF0 file image from the disc file stored on the host PC and sends the file up to uTile. The READ command is invoked by the kermit script UPLOAD.CMD

10 Frequently Asked Questions and Answers

Q1. Is the m328-uTile program only available for the ATMega328P chip?

A1. The source code allows porting the m328-uTile program to the Atmel Atmega2560 chip.

Q2. Tera Term does not connect to the Nano, why?

A2. Make sure you have selected in Tera Term the correct serial port settings and COM port number that is mapped to the USB serial connection used by the Nano. You may need to remove and re-apply power to the Nano board, or to close and re-start Tera Term to have it rescan the serial connection. After you establish a successful connection, you should save the setup configuration.

Q3 Tera Term says “Cannot open COM1” (default port) when connected to the Nano, yet no other COM ports appear when setup is entered, serial port is selected and another Port selection is tried. Looks like Tera Term cannot access the serial ports. What's wrong?

A3. Make sure you installed the Arduino USB driver from the Arduino IDE package, available from Arduino's web site. Please follow the driver installation instructions for your operating system.

Q4. The relays are all energized when the relay module board is initially powered-up. The relays must all be de-energized when the system is powered-up. How do I do this?

A4a. The easy and recommended way is to replace the relay module board with one that has active-high inputs (a high input signal turns on the relay).

A4b. If the relay module cannot be changed to an active-high input type, an inverting buffer device can be wired between the outputs from the m328-uTile and the relay module. A suitable inverting buffer is the 74HCT540 octal inverting buffer. This buffer is rated for 5 V logic supply.

A4c. Use the programmed approach by inverting the state of the result bits before sending them to the output pins. See Example 13 – Driving Active-low Relay Module above for further details.

A4d. Use the out_sense approach by grounding input port pin PB1 (D9) to logically invert all uTile digital output signals.

Q5. I use Linux and wonder if there are any suitable communications programs with VT100 compatible terminal programs that can be used with the Nano board.

A5. Two programs that work well with uTile are Minicom and C-Kermit. Minicom is smaller than C-Kermit, very configurable and easy to use. C-Kermit is highly configurable and its programming and scripting language makes it an excellent choice for use with uTile. C-Kermit supports the disc file storage and transfer feature of uTile.

The PuTTY SSH Client program that runs on Windows and Linux. Please refer to is small and easy to configure to run with uTile. Please refer to the uTile-demo-putty-manual.pdf document for details on configuring PuTTY.

11 A Collection of Application Programs

A selection of programming examples illustrating the versatility, efficiency and simplicity of using the uTile PLC is presented in this section. These examples are based on use of the test rig for simulating uTile inputs/outputs. You can modify the programs if you use a different I/O configuration/hardware.

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

Update notice:
uTile timers T0..T3 count rate have been changed from 1 second to 100 milliseconds to cover a larger timer range. In the following examples please change change T0..T3 to any of the T4..Tb timers to maintain the original seconds count rate.

11.1 Push-button switch and timer

Description: Press switch to start timer. Press again before timeout to stop timer.

Resources:
1 flip/flop
1 delay timer

A0 = Push-button switch, S0
Y0 = Output, LED0
T0 = 5 s time delay

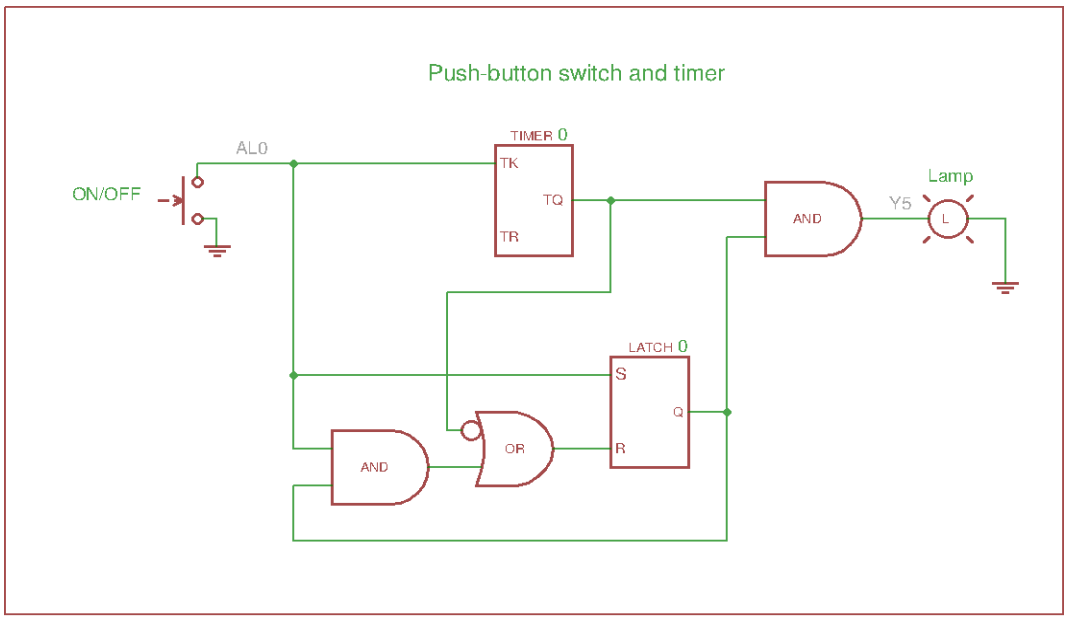


Figure 2 - PB Switch and Timer

```
000  AL0.          008  !             016  END           024  NOP
 001  DUP           009  OR            017  NOP           025  NOP
 002  DUP           010  .R0           018  NOP           026  NOP
 003  .TK0          011  TQ0.          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
```

11.2 '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

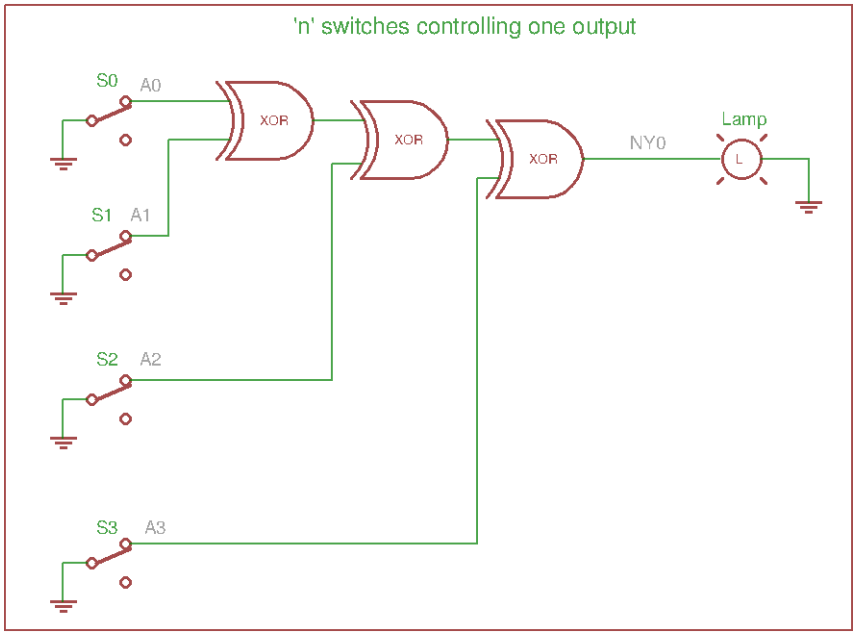


Figure 3 - 'n' switch XOR

```
000  A0.          008  KEY          016  NOP          024  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
```

11.3 Push-button lighting control.

Description: Push-button switch S0 with local indicator LED3 toggles output LED0 on/off each time S0 is pressed. LED3 flashes slowly when output is off. LED3 on full when output LED0 is on. Useful for lighting control when light is not visible from switch location.

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
```

11.1 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).

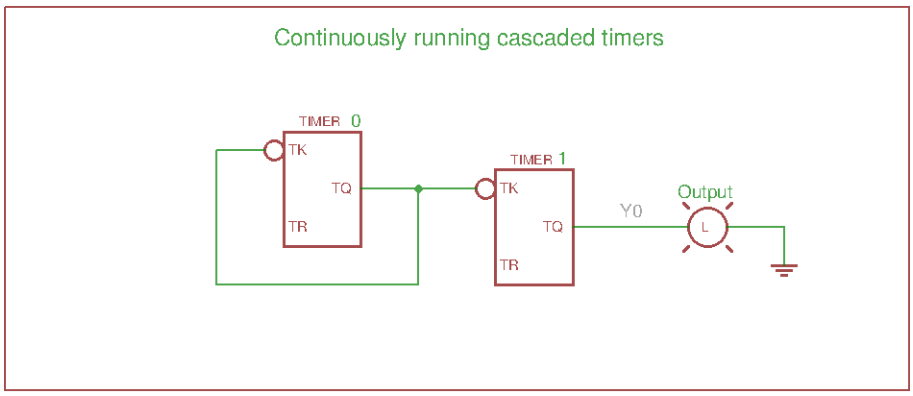


Figure 4 - Continuously running timer

```
000  TQ0.       008  KEY        016  NOP        024  NOP
001  DUP        009  END        017  NOP        025  NOP
002  !          010  NOP        018  NOP        026  NOP
003  .TK0       011  NOP        019  NOP        027  NOP
004  !          012  NOP        020  NOP        028  NOP
005  .TK1       013  NOP        021  NOP        029  NOP
006  TQ1.       014  NOP        022  NOP        030  NOP
007  .Y0        015  NOP        023  NOP        031  NOP
```

Timers:
T0 = 4, Pulse generator repitition rate
T1 = 1, Pulse ON time

11.2 Intrusion alarm system.

Description: Press S0 to arm alarm. T0 time delay before arming alarm

to allow egress from protected area. LED0 indicator (Y0) flashes to indicate hold-off active, turns on fully at end of T0 to indicate alarm is now armed.

If field contact string is broken by an intrusion, T1 alarm hold-off delay starts to allow time for user to reset alarm system by pressing S1. LED1 (Y1) flashes during alarm hold-off period. If S1 is not pressed by end of T1 time delay, alarm is activated and alarm output goes on for T2 minus T1 time. Alarm turns off at end of T2 time. Alarm can be reset at any time by pressing S1.

A0 = push-button switch S0, Arm
A1 = push-button switch S1, 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:
T0 = 5, Alarm arm delay
T1 = 3, Alarm hold-off delay
T2 = 30, Alarm timer

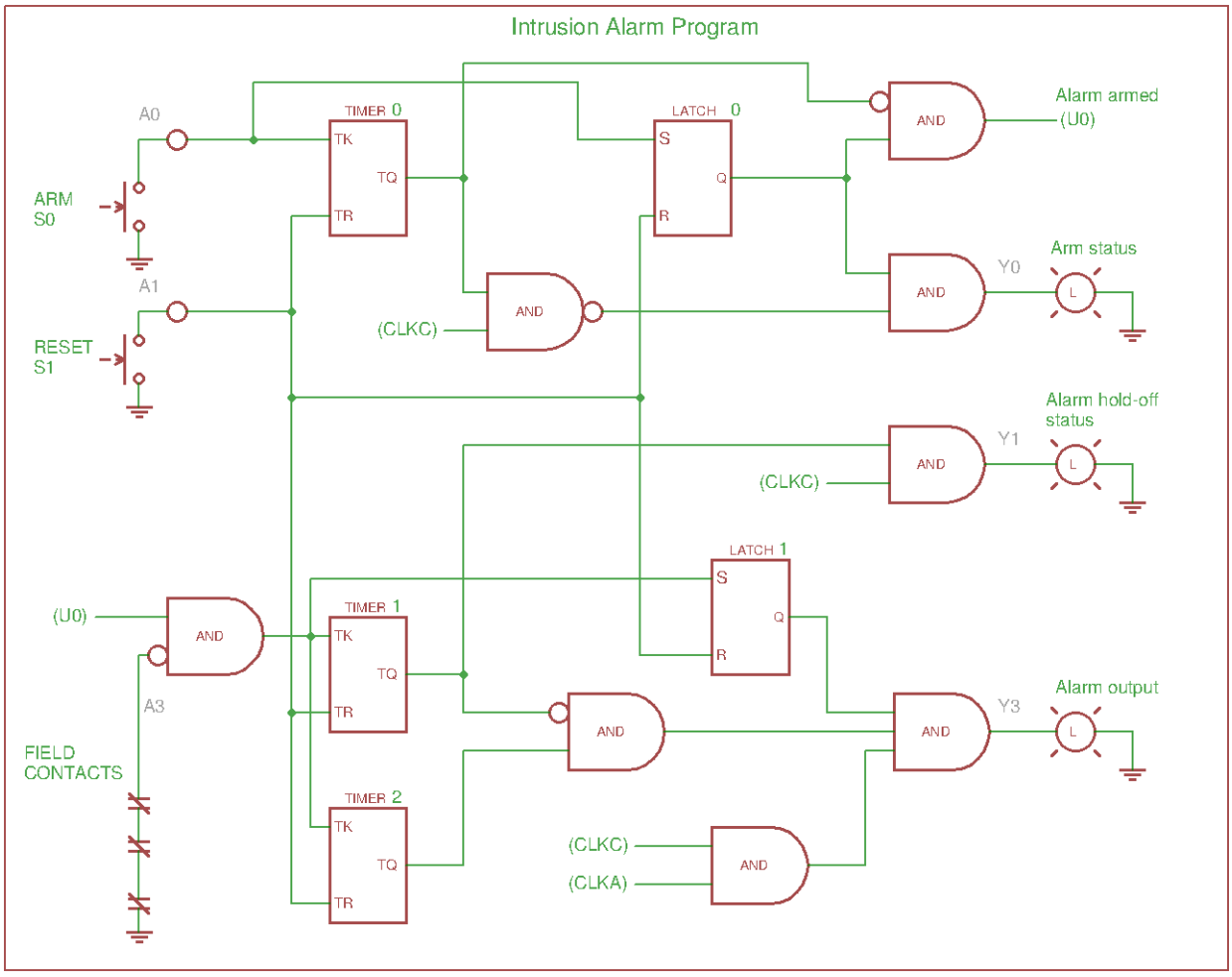


Figure 5 - Intrusion alarm

```
000  A0.          008  DUP          016  CLKB.        024  !
001  !            009  DUP          017  TQ0.         025  Q0.
002  DUP          010  DUP          018  AND          026  AND
003  .S0          011  .TR0         019  !            027  .U0
004  .TK0         012  .TR1         020  Q0.          028  A3.
005  A1.          013  .TR2         021  AND          029  !
006  !            014  .R0          022  .Y0          030  U0.
007  DUP          015  .R1          023  TQ0.         031  AND

032  DUP          040  AND          048  AND          056  NOP
033  DUP          041  U0.          049  .Y3          057  NOP
034  .TK1         042  AND          050  TQ1.         058  NOP
035  .TK2         043  CLKA.        051  CLKB.        059  NOP
036  .S1          044  AND          052  AND          060  NOP
037  TQ1.         045  CLKC.        053  .Y1          061  NOP
038  !            046  AND          054  KEY          062  NOP
039  TQ2.         047  Q1.          055  END          063  NOP
```

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

Description: Classic motor control circuit. Press S0 to start motor, Y0 local RUN indicator and Y5 motor load turn ON. Press S1 to stop motor. If O/L (overload) opens due a fault condition, Y1 O/L indicator turns on and if motor is running it is tripped off. If motor is not running, it is locked out and prevented from starting as long as O/L is tripped.

To reset the circuit to normal operation, O/L must be cleared and S1 pressed to reset motor control logic. Only then can motor be started.

A0 = Start switch, N0
A1 = Stop switch, NO
A3 = O/L switch, NC
Y0 = Run, indicator
Y1 = O/L indicator
Y3 = Motor load

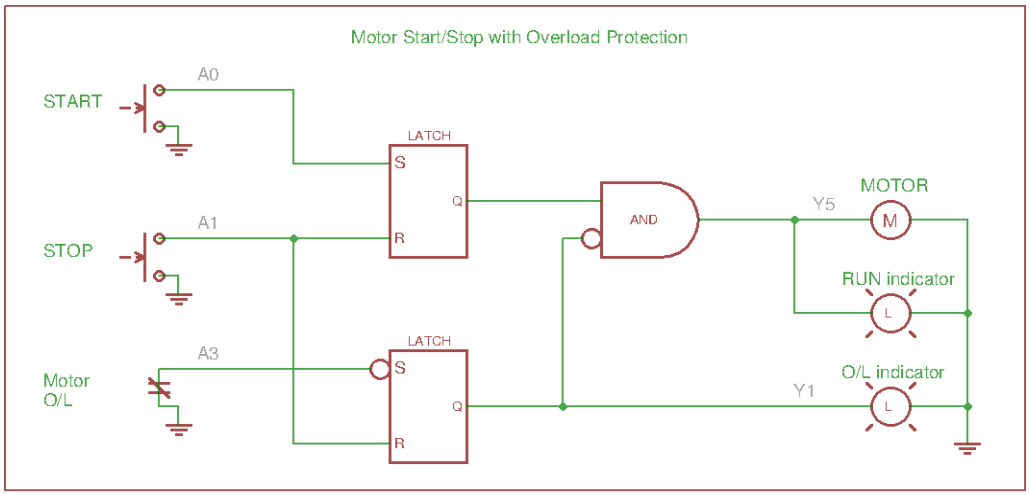


Figure 6 - Motor Starter

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

11.4 Lead/Lag Sump Pump Control with LSHH alarm

Description: Two pumps used to drain a sump (well) 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 the 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 further above the high high level (when there is a pump failure, for example).

Input/output connections:

A1 = Low level switch, LSL, NO switch
A0 = 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

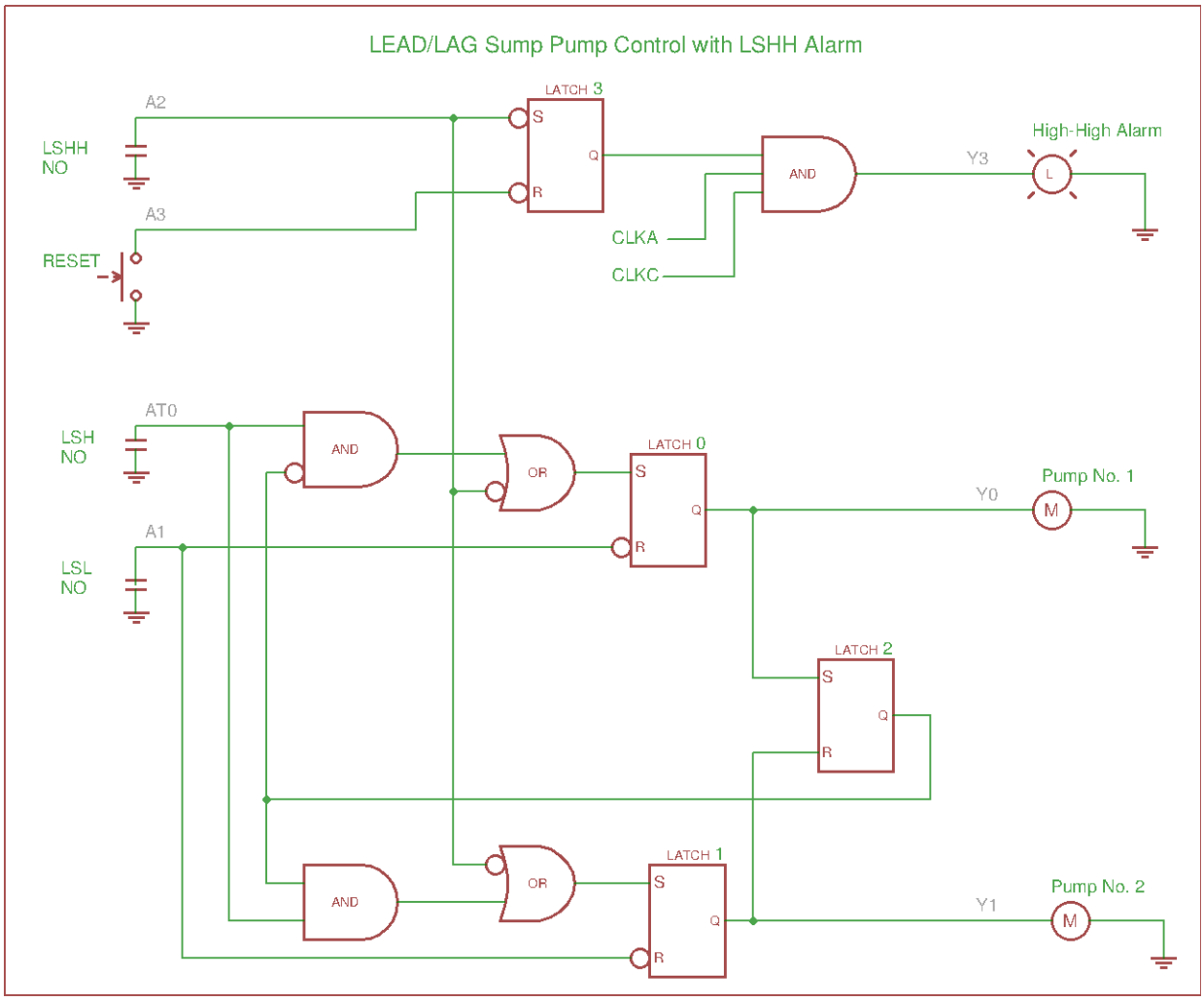


Figure 7 - Lead/Lag sump pump control

```
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  AT0.         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  A1.          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. Sump starts to fill up with water and level rises slowly. Level continues to rise until level passes LSH switch. At this point pump 1 starts, draining water from sump, and level of sump starts to fall.

Level keeps falling as pump 1 contines to drain the sump, until sump 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 operating.

The fill/drain cycles repeat with pump 1 and pump 2 operating alternatively. Each of the pumps operate for equal times, so that neither pump operates continuously and both pumps share operating duty.

Pump failure:

In the case where the pump that is supposed to run fails to start, the water level in the sump keeps rising past the LSH switch until the level passes the LSHH switch. At this point a water level high-high alarm is given (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.

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.

MIT License

Copyright (c) 2016 Francis Lyn

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.