Simple Sequencer Software

Project Name: Simple Sequencer

Project Manager: Jarmo Verho

Prepared by: Minh Nguyen, Minh Do, Thong Nguyen and Jarmo Verho

Issue Date: 9.5.2021

Version: 1.1

Distribution: Public

Synopsis: Description of the Simple Sequencer software.

Version history:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Version #** | **Implemented**  **By** | **Revision**  **Date** | **Approved**  **By** | **Approval**  **Date** | **Reason** |
| 0.1 | JV | 6.4.2021 |  |  | Initial version |
| 1.0 | JV | 7.4.2021 | JV | 7.4.2021 | First public version |
| 1.1 | JV | 9.5.2021 | JV | 9.5.2021 | Update on programming. |
|  |  |  |  |  |  |

Introduction

This document is intended to give an overview of the Simple Sequencer firmware, so that the reader can see the general structure and the main points of the code easier than by going through the source files themselves. The detailed discussion is, for the most parts, omitted to keep this overview reasonably short. Thus, for detailed information, the reader should refer to the source code itself, which is extensively commented. For details about the processor and its peripherals, the reader should refer to ATmega328P’s documentation [[1]](#footnote-1).

The software is written in plain C language. The recommended programming environment is the Microchip Studio [[2]](#footnote-2). It is also possible to use the Arduino programming environment. The Arduino environment can also be used for uploading the compiled software to Arduino [[3]](#footnote-3), although (if there’s no other use for the Arduino IDE), it is probably more convenient to use tools like XLoader [[4]](#footnote-4) for this. See the building instructions for a more detailed discussion about the software tools.

Basic operation

From software point of view, Simple Sequencer is a counter, with direction, modulo and reset inputs. Two moduli are possible, 8 and 4. With modulus of 8 and forward direction, the counter runs the sequence 0-1-2-3-4-5-6-7 and with modulus of 4 and backward direction, the sequence is 3-2-1-0.

The counter advances when a low-to-high transition is detected in the dedicated clock input pin or when the internal (software) clock generator completes one cycle. The software checks the clock source input pin for current operating mode.

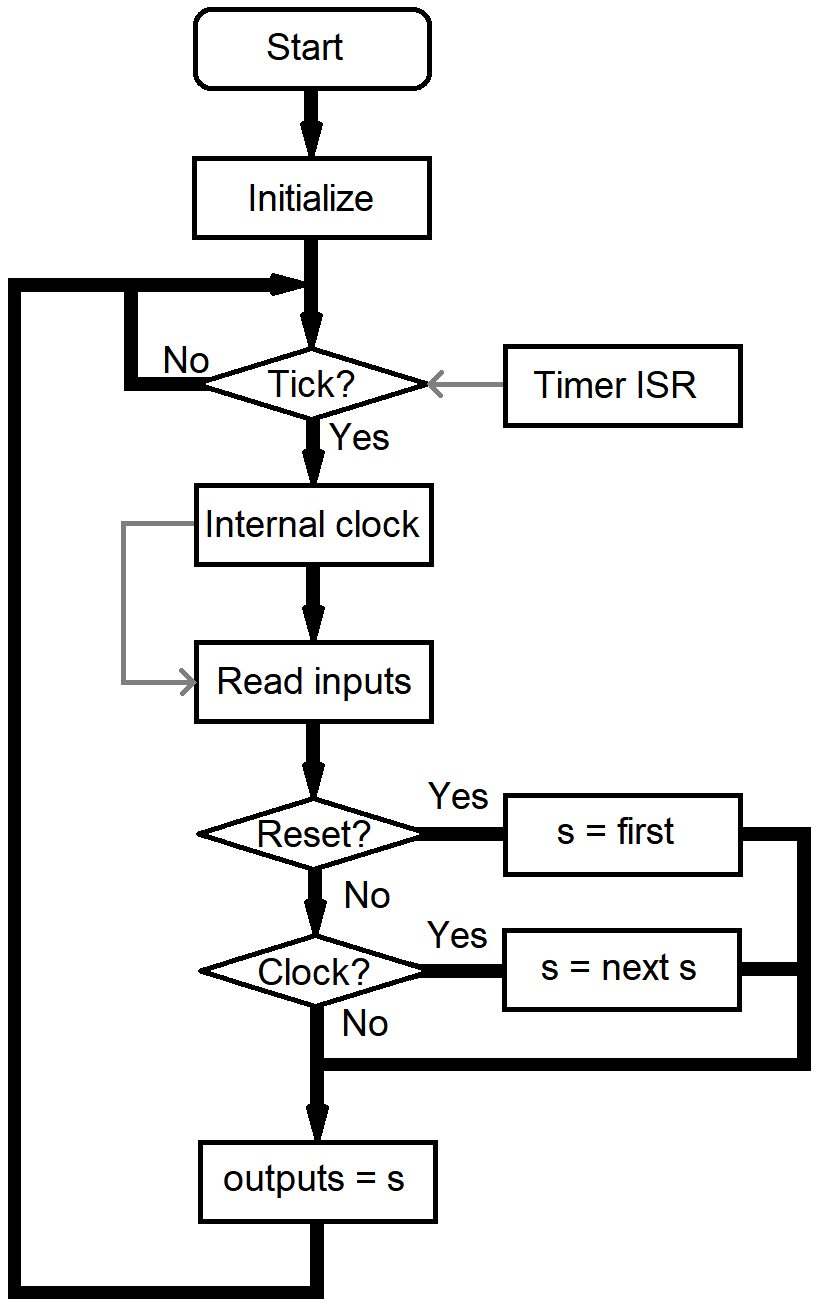
The hardware requires two sets of outputs related to the counter state: it needs a 3-bit binary code, reflecting the counter’s state and an 8-bit one-hot style output. That is, one output of the eight needs to be high, while the other 7 are low. For example, counter state 0 is outputted as 00000001b.

The internal clock generator reads the speed potentiometer’s current state and scales that to beats per minute in range 40…200. This, in turn, is converted to milliseconds between clock pulses. The clock generator then counts milliseconds, producing a clock pulse whenever the specified number of milliseconds has passed.

The inputs, like clock and reset, require filtering to remove noise and contact bounce. This filter is implemented in software. It takes samples of the input signals at fixed rate and when the inputs remain stable long enough, passes that state to the counter logic.

Both the internal clock generator and the filter require a relatively stable time base. This is produced by the processor’s timer module, which triggers an interrupt once per 1ms.

Figure 1 represents the software operation in very general manner. It omits several details, for example the clock source selection logic. The variable ***s*** represents the sequencer’s current state and the operation of the sequence length and direction selection switches is represented by “first” and “next s”. Of these, *first* is determined by both sequence length and direction settings: when direction is forwards, the first state is always 0 and when the direction is backwards, the first state is 7 or 3, depending on sequence length. “Next s” is here a pseudo operation, producing the next s value. This is affected by the sequence direction switch and the sequence length switch.



**Figure 1.** The main program loop.

Another simplification is how the internal/external clock selection is handled. The diagram suggests that the internal clock oscillator notifies the input reading step about a clock edge. However, in actuality both the external clock pulses and the clock pulses produced by the internal clock oscillator are detected, but the clock source selection switch determines which one is treated as the clock signal later. It is also noteworthy that the clock generator step is quite complex in comparison to the main loop, even though it is presented here as just single block.

Functions and function-like # defines

The software has 10 actual functions, defined in SimpleSeq.c:

* main (), containing all the primary program logic; reacts to inputs, produces outputs. The basic functionality of main () is presented in figure 1.
* InitializeHW (), which initializes the processor’s peripherals so that they can be used. This includes setting the processor I/O pin directions (see the hardware documentation for pin list), initializing the processor’s internal analog-to-digital converter and configuring Timer 1 to produce ticks at 1000Hz rate. This function is called just once, before everything else.
* GetTicks (), which returns the number of 1ms (1000Hz) ticks elapsed since the last call. Thus, if called repeatedly in a loop taking less than 1ms per iteration, the function will return just 0 or 1. This is the case with current software; while longer time between calls does not really break the software, it leads to timing inaccuracies.
* ISR (TIMER1\_COMPA\_vect) is the timer 1 interrupt service routine. It gets called 1000 timer per second. This maintains the 1ms timer used by GetTicks (). The processing done in this function should be kept at its minimum. The interrupt is triggered by Timer 1’s output compare functionality.
* GetFilteredInputs () reads the states of all the mode setting switches and the clock and reset inputs. It removes noise from these signals, so that the returned signals are free of spurious transitions. *Note that the clock input state returned by this function actually indicates a rising edge on clock input, not its real state like the others.*
* FilterInputs () is a helper function for GetFilteredInputs (). This function performs filtering for one boolean input. This is done by remembering the values of the last N calls (the default is 4) and comparing them. If they are all true or all false, the function indicates this as a certain true or certain false, otherwise indicating an uncertain result.
* GetIntClockSpeedAsTicks () reads the value (voltage) of the speed setting potentiometer for the internal clock generator and converts this to milliseconds (ticks) between beats.
* GetADCValue () reads the voltage of the specified ADC channel. Currently only one input (speed potentiometer) is read as an analog voltage.
* InternalClockOscillator () is the internal clock oscillator’s state machine. It is meant to use GetTicks () as its time base and GetIntClockSpeedAsTicks () for its speed setting. Basically it maintains a variable containing the time since the last produced clock pulse and adds GetTicks () ticks to it. When the tick limit is reached, the function resets its internal counter and indicates a clock pulse to the caller.
* SetSequencerOutputs () takes the current step number and drives the 8 one-hot and the 3 binary outputs accordingly.

In addition to these, there are 7 function-like macros, defined in the SimpleSeq.h:

* GetSequenceDirection () reads the current (unfiltered) direction switch state (false = forward, true = backwards).
* GetSequenceLength () reads the length switch state (false = 8 steps, true = 4 steps).
* GetClockSource () reads the clock source selection switch state (false = external, true = internal)
* GetClockInput () reads the clock input’s state (false = input low, true = input high).
* GetResetInput () reads the reset input’s state (false = normal operation, true = reset)
* SetGateOutputs () sets the 8 digital outputs driving the gate switches.
* SetPotMultiplexer () sets the 3 digital outputs driving the control voltage potentiometer multiplexers.

Suggestions for quick modifications

Some rather useful modifications are very easy to do and might be worth trying even if the reader is otherwise happy with the default functionality:

* The default range for the internal clock oscillator is 40…200 beats per minute. It can be adjusted by changing the # defines BPM\_MINIMUM and BPM\_MAXIMUM.
* The default software offers two sequence lengths: 4 and 8 steps. These limits are set by # defines MAX\_FULL\_STEP and MAX\_HALF\_STEP.
* If the switches or the clocking seem to be prone to noise and spurious operation, the filtering can be increased by changing the # define FILTERLENGTH. The maximum filter length (in samples, taken once per 1ms) is 8.

1. ATmega328P datasheet: <https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061B.pdf> [↑](#footnote-ref-1)
2. Install package and documentation: <https://www.microchip.com/en-us/development-tools-tools-and-software/microchip-studio-for-avr-and-sam-devices> [↑](#footnote-ref-2)
3. Arduino IDE install package and documentation: <https://www.arduino.cc/en/software> [↑](#footnote-ref-3)
4. XLoader: <https://www.hobbytronics.co.uk/arduino-xloader> [↑](#footnote-ref-4)