

# Contents

<i>Overview.....</i>	<i>1</i>
<i>Noise Output Waveform.....</i>	<i>1</i>
<i>Timing Trace.....</i>	<i>2</i>
<i>MyForth Program.....</i>	<i>3</i>



# 32 Bit Pseudo Random Noise Generator

## Overview

The following describes a pseudo random sequence generator program for a Silicon Laboratories C8051F120 microprocessor (the Target) running at 98 MHz. The generator was programmed in MyForth by Charles Shattuck.

## Noise Output Waveform

Figure 1 below was captured from a Tektronix TDS 2014 oscilloscope. It shows the pseudo random output of the generator on Channel 1 and the loop timing signal on Channel 3. The next section contains a better picture of the Channel 3 waveform.

The Channel 1 trace shows the voltage appearing on bit 6 of port 1. The Channel 3 trace shows the voltage on bit 7 of port 1.

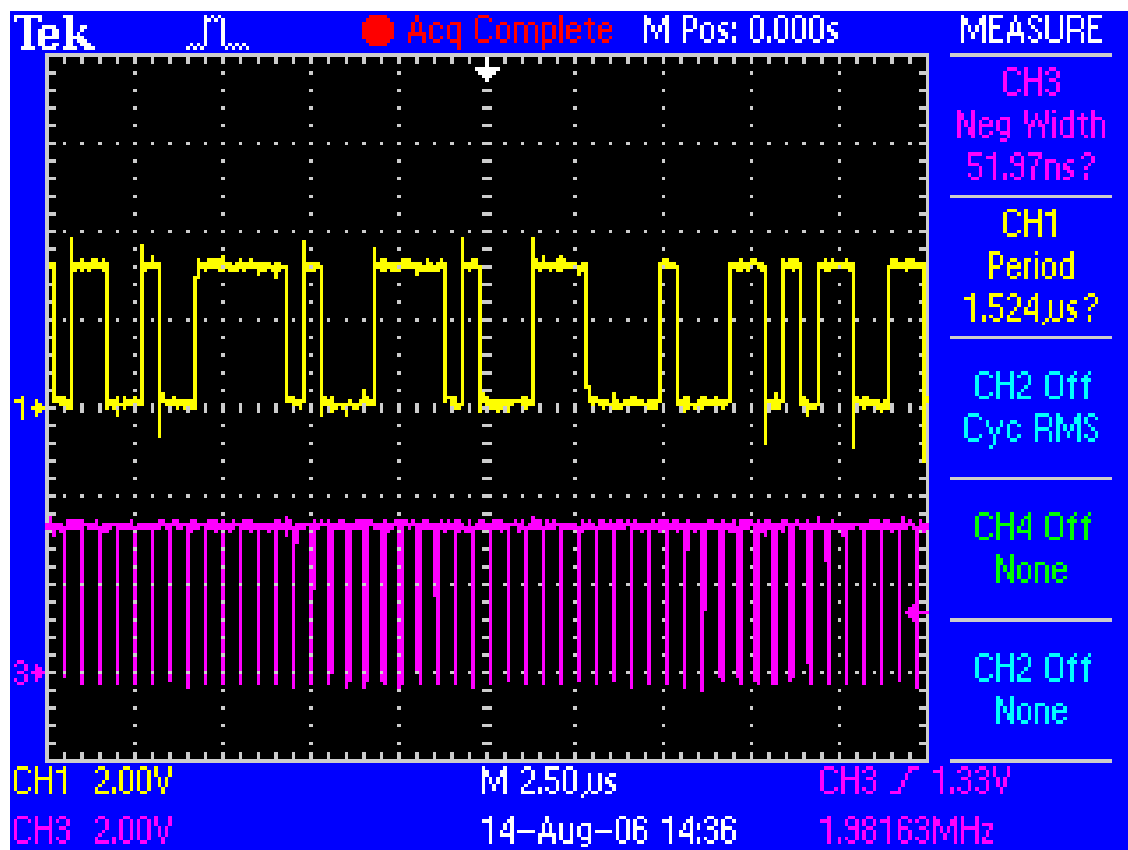


Figure 1. Pseudo Random Output (Channel 1)

# 32 Bit Pseudo Random Noise Generator

## Timing Trace

Figure 2 below shows the timing trace in more detail than Figure 1. Note that the loop return and the toggling of the timing bit consume approximately 51 nanoseconds. A disassembly was checked against the instruction timing chart for the C8051F120 and it was verified that the code used 5 cycles. At a clock rate of 98 MHz, the timing is as expected.

The trace also shows that the instructions to implement the 32 bit shift register and feedback takes approximately 454 nanoseconds to execute. Thus, noise is output at 1.98 MHz.

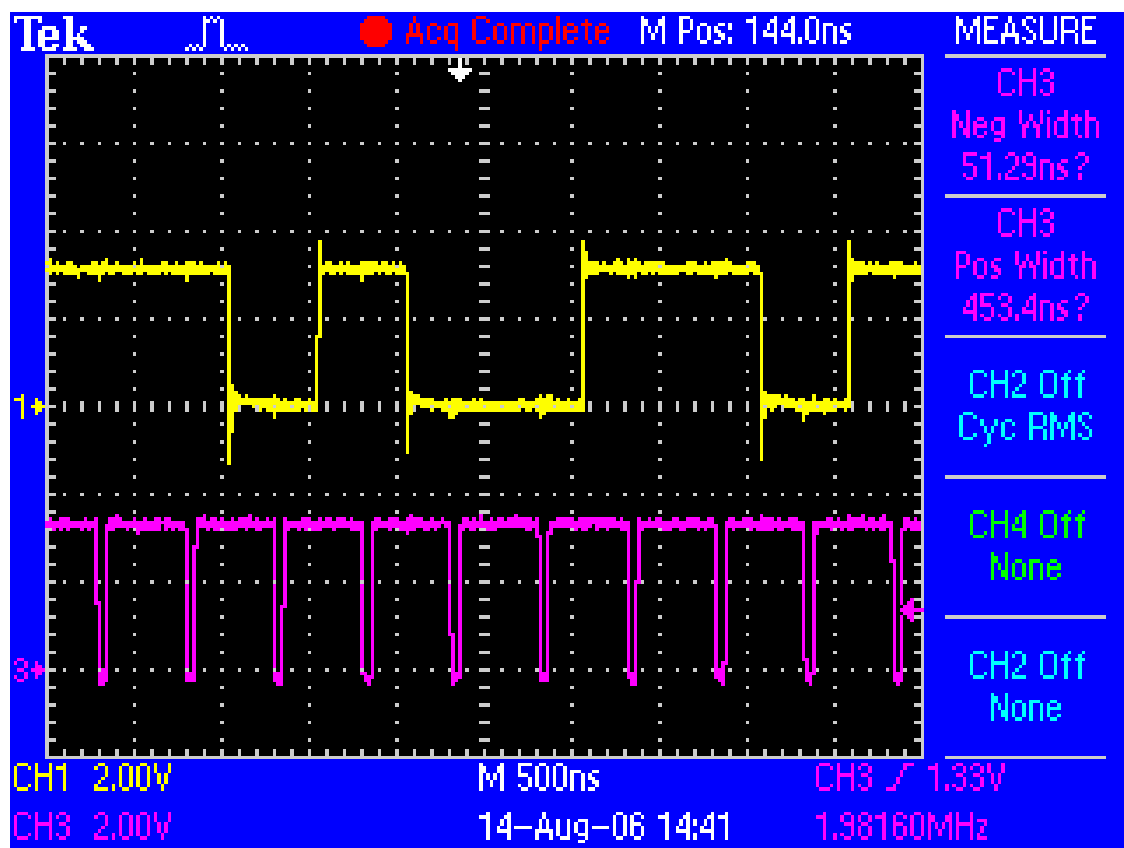


Figure 2. Timing Trace Detail (Channel 3)

# 32 Bit Pseudo Random Noise Generator

---

## MyForth Program

The following is a listing of the MyForth program, contained in files **io.fs** and **main.fs**, to produce the waveforms shown in the previous sections. Note the following:

1. MyForth uses color to improve readability. Comments are blue. Forth definitions and macros (:m) are in red. Definitions in black use Gforth (the Host Forth) to calculate direct cell addresses. There are only two vocabularies, Forth and Target..The [ Word searches Forth first, Target second; the ] Word searches Target first, Forth second.
2. Some crossbar settings in **init-xbr** are for another application (e.g., CS).
3. The **SFR-f** and **SFR-0** Words set the 120's page registers.
4. Special function register (e.g., XBR0 and P0MDOUT) are defined elsewhere, but you can get their addresses from the C8051F120 data sheet.
5. Numbers to be put on the Target's stack must be followed by the # Word
6. The **#!** and **#@** Words store and fetch values to/from a register or direct cell
7. **T** is the top of stack, **a** is the indirect address register (R1) – R0 is the data stack pointer
8. **2\*'** performs a multiply by two with carry (rlc)

A more detailed explanation of the PSR application is contained in the **MyForth Reference Manual**, distributed with MyForth in the DOCS directory.

# 32 Bit Pseudo Random Noise Generator

---

\ io.fs

: init-xbr

SFR-f

\$07 # XBR0 #! \ i2c, SPI, UART0.

\$80 # XBR1 #! \ /SYSCLK

\$44 # XBR2 #! \ UART1, Weak Pullups, XBR enabled.

\$14 # P0MDOUT #! \ Push/Pull SCLK,MOSI

\$ff # P1MDOUT #! \ Push/Pull /SYSCLK, all P1 are output.

\$08 # P2MDOUT #! \ CS on P2.3

SFR-0 ;

\ main.fs

:m outbit 6 .P1 m; \ LED

:m clue 7 .P1 m; \ For timing

:m +clue clue set m; :m -clue clue clr m;

\ :m +clue m; :m -clue m; \ disappear.

cpuHERE constant sequence 4 cpuALLOT

\ XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX

\ ^ bit 31 ^ bit 18

\ ^ sequence ^ sequence+2

\ If all bits are clear, reseed with \$aaaaaaaa.

: ?seed

sequence # a! \$aa #

dup !+ dup !+ dup !+ ! ;

# 32 Bit Pseudo Random Noise Generator

---

\ Shift once with feedback from bits 18 and 31.

```
:m psr+clue
[ sequence 1 + ] #@ \ Get bit 18.
[ 2 .T movbc 7 .T movcb ] \ Move it to bit 7 of TOS.
sequence #@ xor \ xor bits 31 and 18.
2*' \ Move xored bit into carry.
outbit movcb
\ Shift xored bit into sequence.
[ sequence 3 + ] (#@) 2*' [ sequence 3 + ] (#!)
[ sequence 2 + ] (#@) 2*' [ sequence 2 + ] (#!)
[ sequence 1 + ] (#@) 2*' [ sequence 1 + ] (#!)
[ sequence 0 + ] (#@) 2*' [ sequence 0 + ] (#!)
drop -clue m;
```

\ View current shift register in hex.

```
:.psr cr
[ sequence 0 + ] #@ h.
[ sequence 1 + ] #@ h.
[ sequence 2 + ] #@ h.
[ sequence 3 + ] #@ h.
space
[ sequence 0 + ] #@ u.
[ sequence 1 + ] #@ u.
[ sequence 2 + ] #@ u.
[ sequence 3 + ] #@ u.
space
[ sequence 1 + ] #@
[ sequence 2 + ] #@ d.
;
```

\ Load a seed value in the shift register.

```
: psr! ( n1 n2 n3 n4 - ) sequence # a! !+ !+ !+ ! ;
```

\ Note that #@ and #! push and pop the data stack, but

\ (#@) and (#!) assume the top of stack is already free to be used, so

\ don't need to push or pop.

## 32 Bit Pseudo Random Noise Generator

---

```
: 0psr    0 # dup dup dup psr! ?seed ;  
: init    0psr 0 # 7 #for psr 7 #next ;  
: t       13 # 7 #for psr 7 #next .psr ;  
  
: go      init-xbr 0psr begin psr again  
  
: ~P1.6   6 .P1 toggle ;  
: ~P1.7   7 .P1 toggle ;
```