LFSR and ASG Pseudorandom Number Generators
Embedded System Design, Lab 5

Ben Lorenzetti

October 22, 2015

Contents

1 Objectives and Problem Descriptions

1.1 8-Bit Linear Feedback Shift Register (LFSR)

Implement an 8–bit linear feedback shift register (LFSR) on the PIC16F877 44–Pin Demo Board. The LFSR should be designed with suitable taps such that 255 unique numbers are produced in a pseudorandom sequence before the cycle repeats.

The 8–bits should be displayed on the 8 LEDs connected to Port D on the demo board. The LFSR should cycle (change state) once per second.

1.2 Alternating Step Generator (ASG)

Implement an alternating step generator (ASG) on the PIC16F877 44–pin demo board. The ASG should be comprised of three LFSRs with 14–bit, 15–bit, and 16–bit lengths. The pseudorandom bit sequence produced by the ASG should be displayed on the 8 LEDs on board the 44–pin demo board. Eight new bits should be displayed every second.

The ASG should be implemented with suitable taps such that each LFSR produces the maximum length sequence for it's bit size. One LFSR should be cycled each iteration, and the output bit from this LFSR should be used to select one of the other two. From the other two LFSRs, the register which is cycled produces the output bit of the ASG for that cycle.

2 Procedure

2.1 Reading Assignments

The lab procedure and the blackboard post had a cornucopia of reading assignments, including the user manuals for MPLAB X IDE, the PIC Assembler, the 44–pin demo board, and online getting started guides.

There was way to much to actually read, but some of it was helpful. Particularly the first sections of the online getting started with MPLAB guide and the Assembler Users Manual. The PIC16F877 datasheet was also useful since the PDF is searchable.

2.2 LFSRs and ASGs Theory

A linear–feedback shift register (LFSR) is a shift register whose input is a linear, real time function of its current state.

Usually the linear function is a cascade of XOR gates that "tap" some of the upper order bits on the LFSR. The tapped bits are know as the feedback polynomial for the LFSR. If the feedback function is constructed as a cascade of XOR gates, the LFSR is known as a Fibonacci LFSR. An example of a 14–bit Fibonacci LFSR is shown in figure 1.

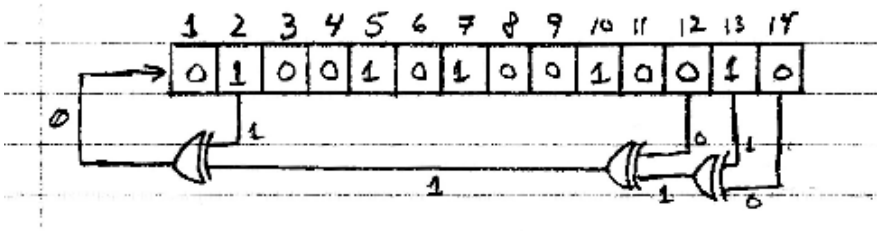

Figure 1: A 14-bit Fibonacci LFSR with feedback polynomial $F(x) = x^{14} + x^{13} + x^{12} + x^2 + 1$

Table 1: Feedback Polynomials with Maximum Length Sequences

Size of LFSR	Feedback Polynomial	Sequence Length	TAP_MASK
3–bits	$x^3 + x^2 + 1$	7	0x0006
4–bits	$x^4 + x^3 + 1$	15	0x000C
5–bits	$x^5 + x^3 + 1$	31	0x0014
6–bits	$x^6 + x^5 + 1$	63	0x0030
7–bits	$x^7 + x^6 + 1$	127	0x0060
8–bits	$x^8 + x^6 + x^5 + x^4 + 1$	255	0x00D8
14–bits	$x^{14} + x^{13} + x^{12} + x^2 + 1$	16383	0x3802
15–bits	$x^{15} + x^{14} + 1$	32767	0x6000
16–bits	$x^{16} + x^{14} + x^{13} + x^{11} + 1$	65535	0xB400

LFSRs are often used as pseudorandom number generators and the initial value is know as the "seed" value. For Fibonacci LFSRs, a seed value of zero will cause the LFSR to be forever stuck in the zero state.

If the feedback polynomial is chosen correctly, the LFSR can produce $2^n - 1$ (not including 0) pseudorandom numbers before the sequence begins to repeat. It turns out that all of the LFSRs of bit length n with a sequence $2^n - 1$ long have some similarities. They have an even number of taps and all of the tap locations, taken together, must be relatively prime. For example, an LFSR tapped at bits 2 and 4 or 2 and 8 would certainly not have the maximum sequence length. Some feedback polynomials that are know to produce maximum length sequences are listed in table 1.

2.3 Implementation Design

Bitwise manipulation is always difficult in any programming language because computers and microcontrollers are designed to use data in groups of bytes—or whatever the word size of the machine may be.

Luckily, the PIC instruction set has rotate left and rotate right instructions which can shift bits in and out of a byte like a queue. Nine bits are involved, eight bits from the byte being rotated and a ninth bit–the carry flag from the status register. The carry flag is set by many ALU operations but, in the case of rotate left and rotart right, the value in the Carry Flag is shifted into the byte and is replaced by the value shifting out the other side of the byte queue. The mnemonics for rotate left and rotate right are

```
RLF f, d
RRF f, d
```

where `f` is the register address and `d=0` indicates store in the accumulator `W` and `d=1` indicates replace at the `f` address.

My design for a generic bit-length LFSR takes takes advantage of the rotate instructions. A hardware concept is shown in figure 2. This arrangement of hardware is better for implementable in software because there are fewer conditional statements; the XOR cascade can be implemented as a loop instead of a unique–per–feedback–polynomial sequence of logic statements. Also, instead of trying to arrange bits to overlap for `XOR` ALU operations, it uses the ALU to add 1s Digit signals with an effect equivalent to XORing if you ignore the higher order bits.

Since LFSRs are easily visuallized as hardware, a flowchart for the XOR function implementation is similarly drawn in figure 3. This flowchart shows the value of the PIC architecture's Carry Flag for implementing an LFSR.

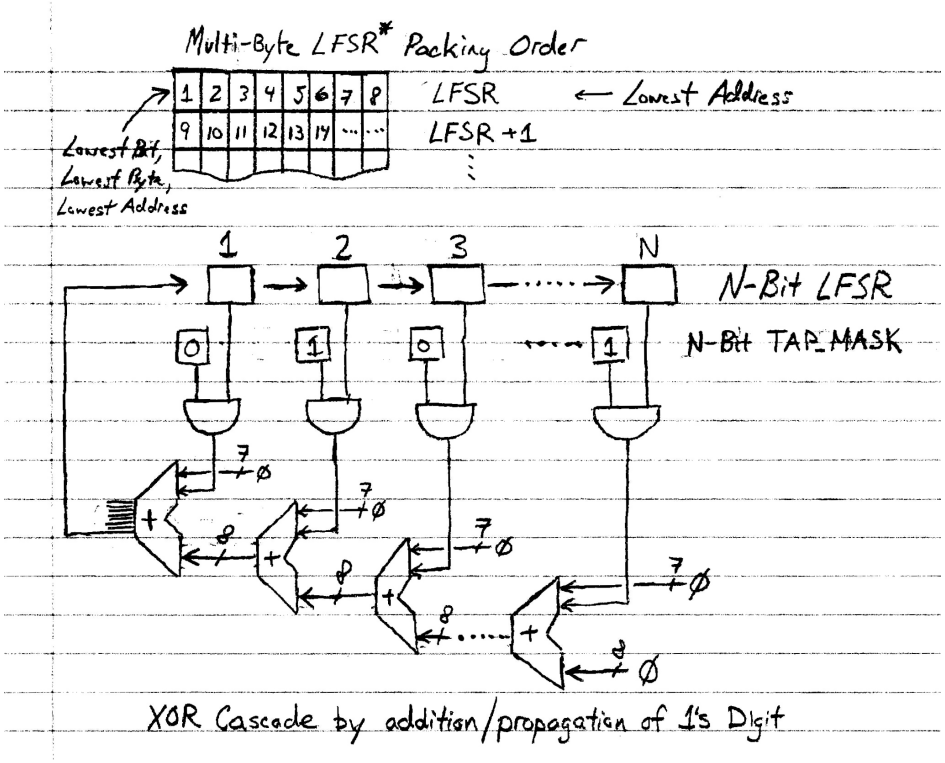

Figure 2: Equivalent Circuit Implementation of an Arbitrary Length Fibonacci LFSR

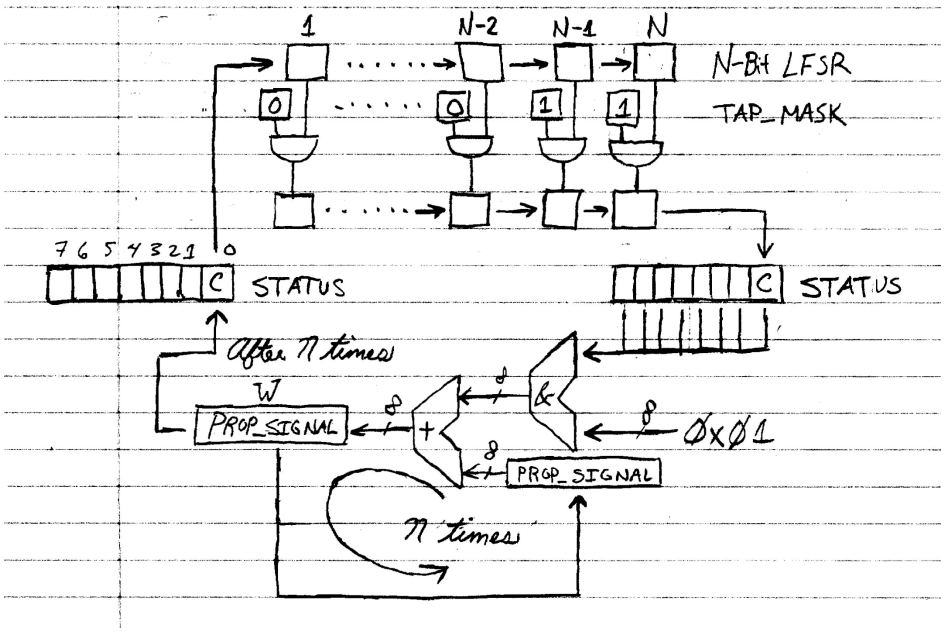

Figure 3: Implementation of XOR Propagation

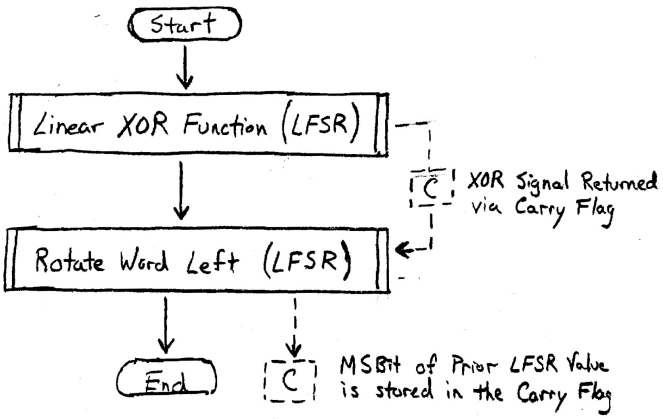

Figure 4: Cycle LFSR Function Implementation Flowchart

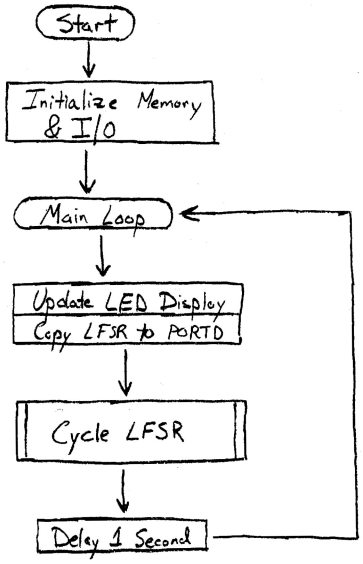

Figure 5: 8-Bit LFSR Implementation Flowchart

The full process of advancing to the next state in the LFSR sequence involves calculating the result of the linear XOR function and then shifting the register by 1–bit with the XOR result as input. This is shown in the implementation flowchart in figure 4.

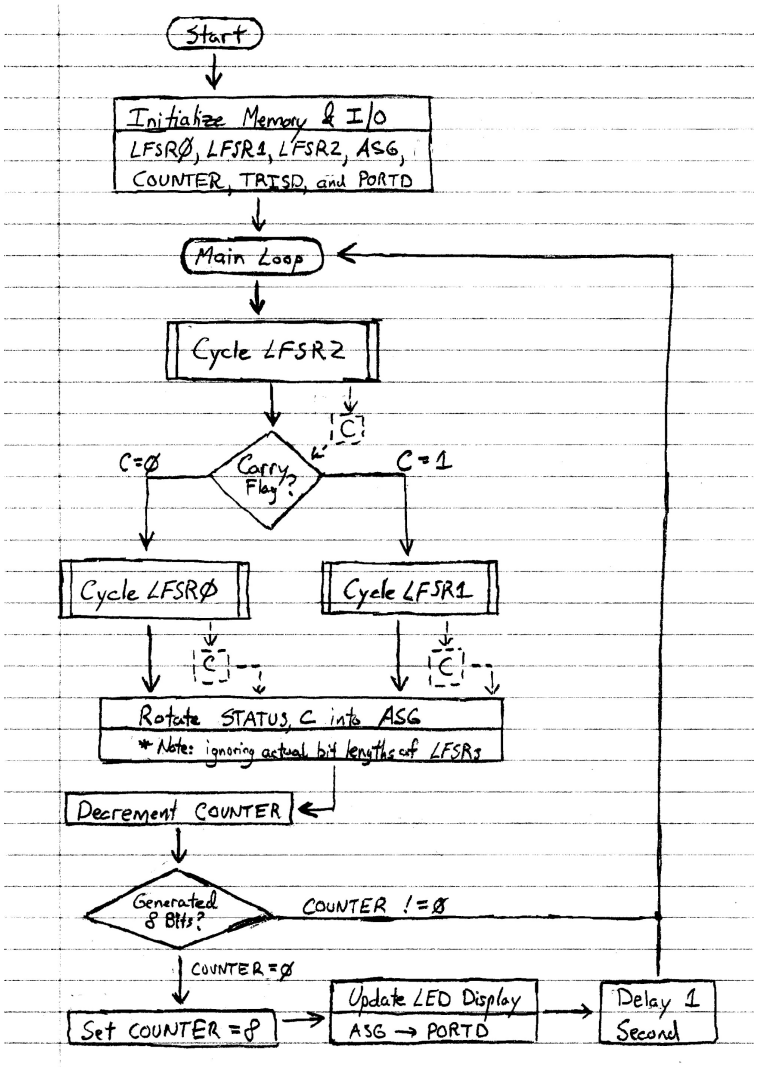

Figure 6: Alternating Step Generator Implementation

8 Implementation Code

8.1 8-Bit LFSR

```
; lfsr.asm
; 8-Bit Linear Feedback Shift Register on PIC Development Board
; Ben Lorenzetti
; Embedded Systems Design, Fall 2015

#include <p16f887.inc>
        __CONFIG        _CONFIG1, _LVP_OFF & _FCMEN_OFF & _IESO_OFF &
           _BOR_OFF & _CPD_OFF & _CP_OFF & _MCLRE_OFF & _PWRTE_ON &
           _WDT_OFF & _INTRC_OSC_NOCLKOUT
        __CONFIG        _CONFIG2, _WRT_OFF & _BOR21V

;----------LFSR Bit/Byte Sizes, Tap Locations, and Initial Values
   ----------;
#define LFSR_SIZE 1             ; LFSR byte size
#define LED_MASK        0xFF    ; 0x07 for 3 bit LFSR, 0x0F for 4-bit
   LFSR, 0xFF for 8-bit LFSR
#define LFSR1_TAP_MASK  0xB8    ; 0x06 for 3-bit LFSR, 0x0C for 4-bit
   LFSR, 0xB8 for 8-bit LFSR
#define INITIAL_VALUE   1

;-------------------------Organize Program Memory-------------------------;
Reset_Vector
        ORG 0
        GOTO Initialize

Interupt_Vector
        ORG 4

;---------------------Allocate Static Variables-------------------------;
        cblock  0x20
        OUTER_DELAY
        MIDDLE_DELAY
        INNER_DELAY
        REMAINDER_DELAY
        PROP_SIGNAL
        BIT_INDEX
        LOCAL_LFSR: LFSR_SIZE
        LFSR1: LFSR_SIZE
        endc

;-------- void rotate_word_left (word*, word_size); ------------------;
RLF_Word        MACRO   Word_Addr, Word_Size
        local i = 0
        while i < Word_Size
```

```
            RLF     (Word_Addr + i), 1
            i += 1
          endw
          ENDM

#define OUTER_MAX_PLUS_1 60
#define MIDDLE_MAX_PLUS_1 40
#define INNER_MAX_PLUS_1 40
#define REMAINDER 1
;---------------------- void Pause_1_Second () ------------------------;
Pause_1_Second
          MOVLW   OUTER_MAX_PLUS_1
          MOVWF   OUTER_DELAY
          MOVLW   REMAINDER
          MOVWF   REMAINDER_DELAY
Inner_Loop
          DECFSZ  INNER_DELAY, f
          GOTO    Inner_Loop
          MOVLW   INNER_MAX_PLUS_1
          MOVWF   INNER_DELAY
Middle_Loop
          DECFSZ  MIDDLE_DELAY, f
          GOTO    Inner_Loop
          MOVLW   MIDDLE_MAX_PLUS_1
          MOVWF   MIDDLE_DELAY
Outer_Loop
          DECFSZ  OUTER_DELAY, f
          GOTO    Inner_Loop
Remainder_Loop
          DECFSZ  REMAINDER_DELAY, f
          GOTO    Remainder_Loop
          RETURN

;------------Linear XOR Cascade Function-------------------------------;
Linear_Xor_Function     MACRO   LFS_Register, Reg_Size, Tap_Mask
          ;------ Pass the function arguments by Copy---------- ------;
          local i = 0
          while i < LFSR_SIZE
           MOVFW  (LFS_Register + i)       ; copy the LSFR to accumulator
           ANDLW  (Tap_Mask + i)           ; AND accum. copy with TAP_MASK
           MOVWF  (LOCAL_LFSR + i)         ; move from accum. to local
              function variable
           i += 1
          endw
          ;---------- Initialize Local Variables --------------------;
          CLRF    PROP_SIGNAL             ; initialize XOR signal to 0
          MOVLW   (8 * Reg_Size)          ; start XOR adding at most
             significat bit
```

```
        BCF     STATUS, C               ; set initial XOR add input to
           0
Xor_Propagation_Loop
        ;---------- rotate 1 bit from local copy of LSFR ----------;
        RLF_Word        LOCAL_LFSR, LFSR_SIZE
        ;---------- Add Carry Bit to PROP_SIGNAL --------------------;
        MOVFW   STATUS          ; get status register for the carry bit
        ANDLW   1               ; keep only the carry bit
        ADDWF   PROP_SIGNAL, 1  ; add carry bit to prop signal
        DECFSZ  BIT_INDEX, 1    ; break loop after all bits propagated
        GOTO    Xor_Propagation_Loop
;-------Return (1s-Digit of PROP_SIGNAL) via Carry Flag ------------;
        RRF     PROP_SIGNAL, 1
        ENDM


;--------------------Initialize Data Memory-------------------------------;
Initialize
        ;---------- Initialize I/O
        BANKSEL TRISD           ; select Register Bank 1
        CLRF    TRISD           ; make Port D all output pins
        BANKSEL PORTD           ; back to Register Bank 0
        ;------------Initialize LSFRs-------------------------------------;
        MOVLW   INITIAL_VALUE
        MOVWF   LFSR1

;-----------------Begin Main Program Loop-----------------------------------;
Main
        ;--------------Update LED Display-----------------------------------;
        MOVFW   LFSR1
        ANDLW   LED_MASK
        MOVWF   PORTD
        ;---------------------Cycle LSFR -------------------------------------;
        Linear_Xor_Function     LFSR1, LFSR_SIZE, LFSR1_TAP_MASK
                ; return value is in the Carry Flag
        RLF_Word        LFSR1, LFSR_SIZE        ; shift LSFR with
           feedback from STATUS, C
        ;-----------------Delay 1 Second---------------------------------;
        CALL    Pause_1_Second
        GOTO    Main

        END
```

8.2 Alternating Step Generator

```
; asg.asm
; 3-LFSR Alternating Step Generator on PIC Development Board
; Ben Lorenzetti
; Embedded Systems Design, Fall 2015

#include <p16f887.inc>
        __CONFIG        _CONFIG1, _LVP_OFF & _FCMEN_OFF & _IESO_OFF &
           _BOR_OFF & _CPD_OFF & _CP_OFF & _MCLRE_OFF & _PWRTE_ON &
           _WDT_OFF & _INTRC_OSC_NOCLKOUT
        __CONFIG        _CONFIG2, _WRT_OFF & _BOR21V

;------------LFSR Bit/Byte Sizes, Tap Locations, and Initial Values
   -------------;
#define LFSR_SIZE 2             ; LFSR byte size
#define LFSR0_TAP_MASK0 0x02    ; 14-bit LFSR lower byte tap mask
#define LFSR0_TAP_MASK1 0x38    ;   the high byte tap mask
#define LFSR1_TAP_MASK0 0x00    ; 15-bit LFSR lower byte tap mask
#define LFSR1_TAP_MASK1 0x60    ;   the low byte tap mask
#define LFSR2_TAP_MASK0 0x00    ; 16-bit LFSR lower byte tap mask
#define LFSR2_TAP_MASK1 0xB4    ;   the lower byte of tap mask
#define INITIAL_VALUE   1

;-------------------------Organize Program Memory-----------------------;
Reset_Vector
        ORG 0
        GOTO Initialize

Interupt_Vector
        ORG 4

;----------------------Allocate Static Variables-------------------------;
        cblock  0x20
        OUTER_DELAY
        MIDDLE_DELAY
        INNER_DELAY
        REMAINDER_DELAY
        PROP_SIGNAL
        BIT_INDEX
        ASG
        COUNTER
        LOCAL_LFSR: LFSR_SIZE
        LFSR0: LFSR_SIZE
        LFSR1: LFSR_SIZE
        LFSR2: LFSR_SIZE
        endc
```

```
;-------- void rotate_word_left (word*, word_size); ----------------;
RLF_Word        MACRO   Word_Addr, Word_Size
        local i = 0
        while i < Word_Size
          RLF   (Word_Addr + i), 1
          i += 1
        endw
        ENDM

#define OUTER_MAX_PLUS_1 60
#define MIDDLE_MAX_PLUS_1 40
#define INNER_MAX_PLUS_1 40
#define REMAINDER 1
;-------------------- void Pause_1_Second () ------------------------;
Pause_1_Second
        MOVLW   OUTER_MAX_PLUS_1
        MOVWF   OUTER_DELAY
        MOVLW   REMAINDER
        MOVWF   REMAINDER_DELAY
Inner_Loop
        DECFSZ  INNER_DELAY, f
        GOTO    Inner_Loop
        MOVLW   INNER_MAX_PLUS_1
        MOVWF   INNER_DELAY
Middle_Loop
        DECFSZ  MIDDLE_DELAY, f
        GOTO    Inner_Loop
        MOVLW   MIDDLE_MAX_PLUS_1
        MOVWF   MIDDLE_DELAY
Outer_Loop
        DECFSZ  OUTER_DELAY, f
        GOTO    Inner_Loop
Remainder_Loop
        DECFSZ  REMAINDER_DELAY, f
        GOTO    Remainder_Loop
        RETURN

;------------Linear XOR Cascade Function-------------------------------;
Linear_Xor_Function     MACRO   LFS_Register, Reg_Size, Tap_Mask
        ;------ Pass the function arguments by Copy---------- ------;
        local i = 0
        while i < LFSR_SIZE
         MOVFW  (LFS_Register + i)      ; copy the LSFR to accumulator
         ANDLW  Tap_Mask#v(i)           ; AND accum. copy with TAP_MASK
         MOVWF  (LOCAL_LFSR + i)        ; move from accum. to local
             function variable
         i += 1
        endw
```

```
        ;---------- Initialize Local Variables ----------;
        CLRF    PROP_SIGNAL             ; initialize XOR signal to 0
        MOVLW   (8 * Reg_Size)          ; start XOR adding at most
           significat bit
        BCF     STATUS, C               ; set initial XOR add input to
           0
Xor_Propagation_Loop#v(LFS_Register)
        ;---------- rotate 1 bit from local copy of LSFR ----------;
        RLF_Word        LOCAL_LFSR, LFSR_SIZE
        ;---------- Add Carry Bit to PROP_SIGNAL ----------;
        MOVFW   STATUS          ; get status register for the carry bit
        ANDLW   1               ; keep only the carry bit
        ADDWF   PROP_SIGNAL, 1  ; add carry bit to prop signal
        DECFSZ  BIT_INDEX, 1    ; break loop after all bits propagated
        GOTO    Xor_Propagation_Loop#v(LFS_Register)
;-------Return (1s-Digit of PROP_SIGNAL) via Carry Flag ----------;
        RRF     PROP_SIGNAL, 1
        ENDM

;--------------------Initialize Data Memory--------------------------------;
Initialize
        ;---------- Initialize I/O
        BANKSEL TRISD           ; select Register Bank 1
        CLRF    TRISD           ; make Port D all output pins
        BANKSEL PORTD           ; back to Register Bank 0
        CLRF    PORTD
        ;----------Initialize Memory-------------------------------------;
        MOVLW   INITIAL_VALUE   ; initialize LFSRs to INITIAL_VALUE
        MOVWF   LFSR0
        MOVWF   LFSR1
        MOVWF   LFSR2
        MOVLW   8               ; initialize COUNTER to 8
        MOVWF   COUNTER
        CLRF    ASG             ; initialize ASG to 0


;----------------Begin Main Program Loop-------------------------------;
Main
        ;-------------------Cycle LFSR2-------------------------------------;
        Linear_Xor_Function     LFSR2, LFSR_SIZE, LFSR2_TAP_MASK
        RLF_Word        LFSR2, LFSR_SIZE
        ;---------Test Carry Bit and Cycle Appropriate Register-----;
        BTFSS   STATUS, C       ; skip next instruction if Carry=1
        GOTO    Cycle_LFSR0
        GOTO    Cycle_LFSR1
        ;--------------Rotate Result into ASG--------------------------;
Rotate_C_Into_ASG
        RLF     ASG, 1
```

```
        ;------------------Decrement and Test Counter-------------------;
        DECFSZ  COUNTER, 1      ; decrement counter, skip next if 0
        GOTO    Main
        ;-----------Reset Counter, Update Display, & Delay-------------;
        MOVLW   8
        MOVWF   COUNTER         ; set counter to 8
        MOVFW   ASG
        MOVWF   PORTD           ; write ASG to the LEDs
        CALL    Pause_1_Second  ; pause remainder of 1 second
        ;------------------End of Main Loop----------------------------;
        GOTO    Main

;-----------------Cycling of the two subjected LFSRs-------------------;
Cycle_LFSR0
        Linear_Xor_Function     LFSR0, LFSR_SIZE, LFSR0_TAP_MASK
        RLF_Word        LFSR0, LFSR_SIZE
        GOTO    Rotate_C_Into_ASG       ; return to main program
Cycle_LFSR1
        Linear_Xor_Function     LFSR1, LFSR_SIZE, LFSR1_TAP_MASK
        RLF_Word        LFSR1, LFSR_SIZE
        GOTO    Rotate_C_Into_ASG       ; return to main program

        END
```