NOTES/HINTS FOR ASSIGNMENT 2 4d

• Up to 4d you generated the PN waveform and sync pulse using a rectangular pulse shape.

• The rectangle shape was generated by sending each PN code and sync pulse output 20 times over by setting $L_{-up} = 20$.

• In part (d) $L_{-up} = 4$ and an impulse is sent through a pulse shaping filter.
  - Zero sample values enter the filter for 3 of 4 samples.
  - This creates an upsample followed by an interpolation filter.
```c
// fm4_PN_intr_GUI.c

#include "fm4_wm8731_init.h"
#include "FM4_slider_interface.h"
#include "gen_PN.h"

#define L_UP 20

// Create (instantiate) GUI slider data structure
struct FM4_slider_struct FM4_GUI;
struct Mseq PN1;
struct Mseq PN2;
int16_t xLh, xRh;
volatile int16_t idx_up = 0;

void PRGCRC_I2S_IRQHandler(void)
{
union WM8731_data sample;
  int16_t xL, xR;
  gpio_set(DIAGNOSTIC_PIN,HIGH);
  // Get L/R codec sample
  sample.uint32bit = i2s_rx();

  // Process
  if (idx_up == 0) // Output a new sample every L_up samples
  {
    gen_PN(&PN1);
    gen_PN(&PN2);
    //write PN code bit to the left output sample
    // and the PN sync bit to the right output sample
    if (((int16_t)FM4_GUI.P_vals[5] > 0)
    {
      xLh = 20000*(int16_t)PN2.output_bit - 10000;
      xRh = 20000*(int16_t)PN2.sync_bit - 10000;
    }
    else
    {
      xLh = 20000*(int16_t)PN1.output_bit - 10000;
      xRh = 20000*(int16_t)PN1.sync_bit - 10000;
    }

    // Breakout and then process L and R samples with
    // slider parameters for gain control
    xL = (int16_t) (FM4_GUI.P_vals[0] * xLh);
    xR = (int16_t) (FM4_GUI.P_vals[1] * xRh);
    idx_up = (idx_up+1) % L_UP;

    // Return L/R samples to codec via C union
    sample.uint16bit[LEFT] = xL;
    sample.uint16bit[RIGHT] = xR;
    i2s_tx(sample.uint32bit);
    NVIC_ClearPendingIRQ(PRGCRC_I2S_IRQn);
    gpio_set(DIAGNOSTIC_PIN,LOW);
  }

int main(void)
{
  // Initialize the slider interface by setting the baud rate (460800 or 921600)
  // and initial float values for each of the 6 slider parameters

  * Defined on last page
  ```
init_slider_interface(&FM4_GUI, 460800, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0);

// Send a string to the PC terminal
write_uart0("Hello FM4 World!
\n");

// Set up PN generator
gen_PN_init(&PN1, 5, 3, 5, 0x1); // 5 stage
// gen_PN_init(&PN1, 10, 7, 10, 0x1); // 10 stage
// gen_PN_init(&PN2, 5, 3, 5, 0x1); // 5 stage
// gen_PN_init(&PN2, 10, 7, 10, 0x1); // 10 stage

// Some #define options for initializing the audio codec interface:
// FS_8000_HZ, FS_16000_HZ, FS_24000_HZ, FS_32000_HZ, FS_48000_HZ, FS_96000_HZ
// IO_METHOD_INTR, IO_METHOD_DMA
// WM8731_MIC_IN, WM8731_MIC_IN(boost), WM8731_LINE_IN
// Sampling rate (aps)
// Audio input port
// Audio samples handler
// Output headphone jack Gain (dB)
// Line-in input gain (dB)

while(1){
    // Update slider parameters
    update_slider_parameters(&FM4_GUI);
}

d.) **Required for ECE 5655, 20 pt Bonus for ECE 4655**: Pulse shape the bit sequence using a square-root raised cosine pulse shape FIR filter defined in the header file src\_shape.h. Additional signal processing is required to implement a pulse shaping scheme using a *raised cosine* (RC) pulse shape. This will give you a chance to again use the CMSIS-DSP library, this time try out filtering. The system block diagram is the following:

```
int8_t PN 0/1 {0,1} Code Bits mod 31 or 1023
Level Shift
\{±10000\}
12 kbits/s (effective)

{1±0000} Upsample by 4 means stuff 4-1 zero samples

float32_t float32_t float32_t int16_t
SRC FIR Filter
4 48 ksamples/s (actual)

Use CMSIS-DSP float32_t FIR functions

To codec output
```

The pulse shaping operation is jumping ahead to give you a taste of FIR filtering and impulse train modulation from digital communications applications. An upsampling factor of four is employed, which means on every fourth pass through the SPI2\_IRQHandler function you will draw a CA code value from the code arrays scaled to ±10000, modulo 1023. On the three remaining passes you insert 0 (zero). The values are passed into a linear filter as follows:

```
#include "b\_RC4.h" // bring in filter coefficients b\_RC4 and #define M\_FIR
...
// Create two filter instances, one for the PN code and one for the sync pulse
// Start by declaring the working variables globally
float32_t x1, y1, state1[M\_FIR]; // working variables for channel 1
arm\_fir\_instance\_f32 s1;
float32_t x2, y2, state2[M\_FIR]; // working variables for channel 2
arm\_fir\_instance\_f32 s2;
...
// In Main insert the following to fill the filter data structures with the needed
// variables for FIR filtering one sample at a time.
arm\_fir\_init\_f32(&s1, M\_FIR, h\_FIR, state1, 1); // 1 => process one sample only
arm\_fir\_init\_f32(&s2, M\_FIR, h\_FIR, state2, 1);
...
// Finally, in the ISR, for each channel, 1 and 2 (1 shown below)
// left output sample is +/- 10000.0f*codebit or 0.0f based on a modulo 4 index counter
// the codebit is taken from the PN generator as in part c.
// Repeat for producing the sync pulse out of the right channel.
float x = (float32_t)left\_out\_sample;
arm\_fir\_f32(&s, &x, &y, 1);
left\_out\_sample = (short)(y);
...
```

The bit rate will be 48/4 = 12 kbps (PN code bits per second). The header file `b\_RC4.h`/`b\_src.h` is supplied in the ZIP. The details of how to create it is included in the Jupyter notebook for Lab 2.