To further leverage the capabilities of an FPGA a high-level synthesis (HLS) system can be created by utilising the previously generated PWM IP block with the Vivado softcore MicroBlaze processor. This lab aims to create an HLS that allows the FPGA to produce a PWM so that desired notes can be produced as well as set of respective LEDs to flash in unison. The final product will be able to play "The Pokemon Centre theme" composed by Junichi Masuda.

An HLS that produces a PWM syntheissed by producing a new IP integrated circuit through the IPI interface of Vivado. To achieve this, the MicroBlaze softcore processor will need to be insaniated as well as its AXI_GPIO IP blocks to allow for general-purpose input/output communication between the peripherals of the Nexys DDR 4. Furthermore, the UART, LEDs/switches, and PWMAudio IP will require instantiation to allow those onboard peripherals to be utilised.

Furthermore, these newly instantiated peripherals' names must match the constraints file to ensure the correct routing of the circuit. Finally, the clockwizard is utilised to ensure an adequate timing execution amongst peripherals. These pheripherials will be autorouted so that the PWM is triggered by a correlating set of 9 switches (declared in the GPIO), which also controls its corresponding LED.

The bitstream can then be generated to produce an HLS circuit that can be accessed on Vivado Visits so that C code can be programmed onto The Nexys DDR4. This is achieved through Vivado HLS as it creates an RTL implementation from the C-level source code and extracts the control and data flow from the source code. It implements the design based on default and user-applied directives.

$$N = \frac{100Mhz}{f*1024}$$

Using the equation above, the desired frequency can be generated on the new HLS by treating it as an input for the PWMAudio IP block. I.e. if the desired note is 'A' 4th octave, it would require a frequency of 440 Hz; resulting in an N value of 222. N value may require rounding as it will be stored as an integer to avoid occupying excessive memory and is used to internally triggers the desired onboard switches. A scalable music player that outputs PWM can be built by creating arrays that represent N values for each note of each octave; however, the memory requirements of 88 integers are a minor disadvantage.

```
const int A[9] = {3551, 1776, 888, 444, 222, 111, 55, 28, 14};
const int As[9] = {3351, 1676, 838, 419, 209, 105, 52, 26, 13};
const int B[9] = {3163, 1582, 791, 395, 198, 99, 49, 25, 12};
const int C[9] = {5973, 2986, 1493,747, 373, 187, 93, 47, 23};
const int Cs[9] = {5638, 2818, 1409, 705, 352, 176, 88, 44, 22};
const int D[9] = {5322, 2660, 1330, 665, 333, 166, 83, 42, 21};
const int Ds[9] = {5021, 2511, 1256, 628, 314, 157, 78, 39, 20};
const int E[9] = {4741, 2370, 1185, 593, 296, 148, 74, 37, 19};
const int F[9] = {4473, 2237, 1119, 559, 280, 140, 70, 35, 17};
const int Fs[9] = {4224, 2111, 1056, 528, 264, 132, 66, 33, 16};
const int G[9] = {3986, 1993, 996, 498, 249, 125, 62, 31, 16};
const int Gs[9] = {3762, 1881, 941, 470, 235, 118, 59, 29, 15};


int notes[] = {D[4],A[4],D[4],A[5],G[4],Fs[4],E[4],Cs[4],2,Cs[4],A[4],Cs[4],Fs[4],E[4],
               Cs[4],D[4],Fs[4],2,D[4],Cs[4],D[4],A[5],G[4],Fs[4],E[4],Cs[4],2,Cs[4],A[4],
               Cs[4],Fs[4],E[4],Cs[4],Ds[4],Fs[4],2,Fs[4],A[4],G[4],A[4],G[4],Fs[4],E[4],
               Cs[4],E[4],Fs[4],G[4],Fs[4],E[4],D[4],Fs[4],A[4],G[4],Fs[4],G[4],A[5],B[5],
               A[4],G[4],Fs[4],G[4],Fs[4],G[4],Fs[4],E[4],D[4]};


int delayTime[] =
{1,1,1,2,2,1,1,3,4,1,1,1,2,2,1,1,3,4,1,1,1,2,2,1,1,3,4,4,4,1,1,1,4,4,4,1,1,1,1,4,4,4,1,1,1,1,4,2,1,1,4,1,1,1,1,4};

int music[66];
```

Figure 1: All playable piano notes integrated into the HLS

Testperiph.c, provided in the lab script, can be used to verify these N values since it has access to Discretewrite and Discreteread. As a result, switches, LEDs, or desired functions of PWM pitch can be activated and deactivated. Thus, a visual and audible lighting show will result as the switches trigger the PWM generator.

Through the use of combining the elements of each piano note, in a note array (delcared in Figure 1 as **notes[]**), this program can now play more complex pieces such as The Pokemon Centre theme by integrating a delay system that replicates musical delay notations onto this HLS. Delays can be achieved by creating another subroutine that plays the delay's respective note for the stated amount of time (declared in Figure 1 as **delayTime[]**) in Figure 2.

```
int compose(u16 DeviceID_LED, u16 DeviceID_LED_Audio, u32 GpioWidth){

	volatile int Delay;
	int Status;
	Status = XGpio_Initialize(&Gpio, DeviceID_LED);
	Status = XGpio_Initialize(&Gpio2, DeviceID_LED_Audio);

	if(Status != XST_SUCCESS){
		return XST_FAILURE;
	}

	XGpio_SetDataDirection(&Gpio, 1, 0x0);
	XGpio_SetDataDirection(&Gpio, 2, 0x0);

	for(int i = 0; i < sizeof(music)/sizeof(music[0]); i++){
		XGpio_DiscreteWrite(&Gpio, CHANNEL, music[i]);
		XGpio_DiscreteWrite(&Gpio2, CHANNEL, music[i]);

		for(Delay = 0; Delay < DELAY*delayTime[i]; Delay++);

		XGpio_DiscreteClear(&Gpio2, CHANNEL, 0);
		XGpio_DiscreteClear(&Gpio, CHANNEL, 0);

	}

	XGpio_DiscreteWrite(&Gpio, CHANNEL, 2);
	XGpio_DiscreteWrite(&Gpio2, CHANNEL, 2);

	return XST_SUCCESS;

}
```

Figure 2: HLS PWM player with encorporated piano delays and BPM

This HLS is further improved by integrating a delay multiplier **(DELAY)** that uses BPM as a variable and can be calculated using the equation below. It considers the clock speed divided by 2^(available bits (9) - 1) * beats per second. This equation is a blackbox and was calibrated by comparing it against a metronome. Furthermore, not every piece with musical notations can be played, as note N values may be out of range for the PWM generator and will not produce the desired pitch. As a result, only octaves 4-9 are audible.

$$DELAY = \frac{2*100Mhz}{2^8} * \frac{BPM}{60}$$