机器学习降维方法实验报告:

PCA Isomap LLE

李梓铉

学号 3118103163

指导老师: 孟德宇教授

2018 年 12 月 6 日

1 实验背景

1.1 实验目的

机器学习通常需要大量的训练样本，而在现实中得到的数据通常很难满足训练样本的密集采样，而且属性维度的增加若满足采样条件，会使得样本的数字达到天文数字。在高维情形下出现的数据样本稀疏，巨大的矩阵导致的计算距离困难等等，就是机器学习方法面临的严重障碍“维数灾难”（curse of dimensionality）. 解决维数灾难的重要方法之一是降维（dimension reduction）, 通过数学变换将高维空间转变为一个低维子空间使得样本密度提高，降低距离计算的复杂度。一般来说有效的降维方法要保持前后的样本之间的距离。

降维的方法主要分为两类：

第一类也是降维最简单的方法就是基于线性变换的线性降维方法，多维缩放（MDS），主成分分析（PCA）算法均属于此类降维方法。线性降维方法是高维到低维空间的线性映射，但对于现实任务中的很多数据，需要使用非线性映射才能找到适当的低维嵌入。

第二类方法就是非线性方法，非线性方法中的第一种是核化线性降维，如核主成分分析（KPCA），使用核函数，将高维特征空间中得数据投影到一个超平面上实现降维。第二种是流形学习（manifold learning），这是一种借鉴拓扑流形概念的降维方法。低维流形嵌入到高维空间的分布虽然很复杂，但局部仍然具有欧式空间的性质，因此可以在局部建议降维映射关系，然后在推广至全局。著名的流形学习方法有等度量映射（Isometric mapping）Isomapping，局部性嵌入（local linear embedding）LLE 等。

In this lab, students will use a round-robin with interrupt software architecture using the `LEDTest1.c` example program as the basis for the design. The software needs to do 4 main functions. First the SW needs to be able to initialize the peripheral units (timer and programmable flags). Second the SW needs to be able to respond to button. Third the SW needs to be able to respond to timer interrupts. And fourth, the SW needs to be able to control the LED's per the buttons and the timer interrupt. From an architectural standpoint, the SW can be broken into 4 main sections:

- LED Control – functions to control the LED's
- Interrupt – ISR
- Timer Control – functions to control the timer units
- Main Loop

Students will need one timer interrupt. In this interrupt, they should set a flag indicating that the ISR was asserted. The main control SW needs to be able to do 2 things. First it needs to initialize the timer unit, the programmable flags and the interrupt. Next it needs to read the button press, the flags updated by the ISR's, and output the correct LED's.

1.2 Equipment

There is a minimal amount of equipment to be used in this lab. The few requirements are listed below:

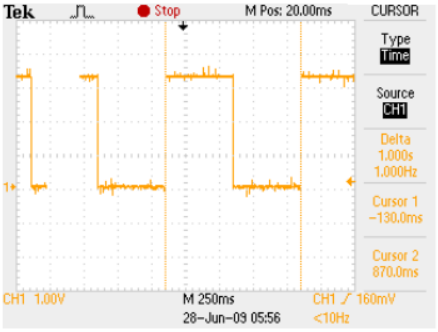
- Xilinx ISE Navigator Software (v10.0.1)
- Computer capable of running the software mentioned
- Spartan 3E FPGA Developer Board (For Hardware Simulation)

1.3 Procedure

1. Start the ISE Navigator. See the ISE 8.2i Quick Start Tutorial.
2. Create a new project.
3. Import copies of the Verilog modules AND_OR, MY_AND2, and MY_OR2 to the new project just created.
4. Create a Test Bench (called Test Fixture in Verilog).
5. Create the actual input stimulus.
6. Run the simulation, examine the waveforms, and verify functioniality.

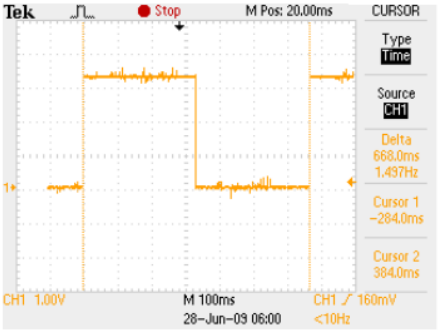
2 Schematic Diagrams

This section consists of screenshots taken during the laboratory procedure. One set of the screenshots is of the periods captured by the oscilliscope and the other set is captured by the logic analyzer.



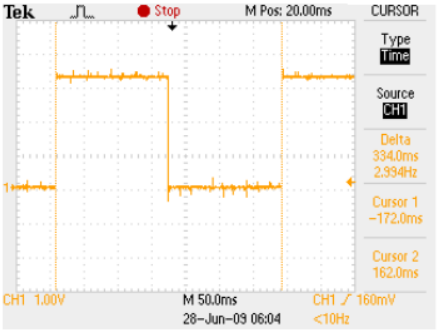
1S PERIOD FOR LED4

(a) LED4 Period



~666.7mS PERIOD FOR LED5

(b) LED5 Period



~333.3mS PERIOD FOR LED6

(c) LED6 Period

图 1: Period of LED blink rate captured by osciliscope.

3 Experiment Data

This section will consist of the important code blocks which were changed in order to meet the requirements of the lab.

```
/*****************************************************************
 Interrupt Service Routin for Timer 0.
*****************************************************************/
// LED4
static ADI_INT_HANDLER(Timer0_ISR) {

   // See if this is a timer 0 event, by calling a function to
   // read corresponding bit (16) in the the SIC_ISR register

   if (adi_int_SICInterruptAsserted(ADI_INT_TIMER0) == ADI_INT_RESULT_NOT_ASSERTED)

   // This return value tells the interrupt manager to process the next
   // ISR in the chain for this IVG, because we haven't yet serviced the
   // peripheral that interrupted this time

      return (ADI_INT_RESULT_NOT_PROCESSED);

   // clear timer 0 interupt
   adi_tmr_GPControl(ADI_TMR_GP_TIMER_0, ADI_TMR_GP_CMD_CLEAR_INTERRUPT, NULL);

   // toggle the specified LED
   // ON|ON|ON
   if(button0 && button1 && button2){
      ezToggleLED(EZ_FIRST_LED);          // LED4,5,6 BLINK
   }

   //***CODE SNIPPED****//

   return (ADI_INT_RESULT_PROCESSED);

}

   //***CODE SNIPPED****//

/*************************************************************************
  Function: Init_Timers

  Set up timers for PWM mode and enale them.
*************************************************************************/

void InitTimers(void)
{
   //Setting up command table for Timer 0
   ADI_TMR_GP_CMD_VALUE_PAIR Timer0ConfigurationTable [] = {
      { ADI_TMR_GP_CMD_SET_TIMER_MODE,          (void *)0x01           },
      { ADI_TMR_GP_CMD_SET_COUNT_METHOD,        (void *)TRUE           },
      { ADI_TMR_GP_CMD_SET_INTERRUPT_ENABLE,    (void *)TRUE           },
      { ADI_TMR_GP_CMD_SET_OUTPUT_PAD_DISABLE,  (void *)TRUE           },
      //***CODE SNIPPED****//
      { ADI_TMR_GP_CMD_SET_WIDTH,               (void *)0x00400000     },
      { ADI_TMR_GP_CMD_END,                     NULL                   },
   };

      //***CODE SNIPPED****//

}

/*****************************************************************
*  Function:    main
*****************************************************************/
void main(void) {

   u32 ResponseCount;
```

```
    void *pExitCriticalArg;
65  u32 i; //loop variable


67

    // initialize the EZ-Kit
69  ezInit(1);

71  // initialize the flag manager because the LEDs and buttons connect via flags
    // Since callbacks are not being used memory does not to be given to the service
73  ezErrorCheck(adi_flag_Init(NULL, 0, &ResponseCount, NULL));

75  //***CODE SNIPPED****//

77  InitTimers();

79  while (1) {
    //***CODE SNIPPED****//
81      }

83  } // END WHILE
} // END MAIN
```

3.1 Formulas and Overall Descriptions Used

This part of the laboratory was done for Feedback Control. Most of this laboratory's calculations were completed and compiled by the folks at Quanser (the manufacturer of the inverted pendulum) and will give the lab a good starting place. Below are the state equation and gain values used initially in the lab:

$$\begin{bmatrix} \dot{\alpha} \\ \ddot{\alpha} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 81.7 & 0 & 0 & -13.9 \\ 0 & 0 & 0 & 1 \\ 39.7 & 0 & 0 & -14.4 \end{bmatrix} \begin{bmatrix} \alpha \\ \dot{\alpha} \\ \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ 24.5 \\ 0 \\ 25.4 \end{bmatrix} V$$

$$K = \begin{bmatrix} 21 & 2.8 & -2.2 & -2.0 \end{bmatrix}$$

Other values, such as the $\frac{\text{Volts}}{\text{Degree}}$ and $\frac{\text{Degrees}}{\text{Volt}}$ were obtained by first determining the max angle of the pendulum on both extreme sides.

Using the max angles from above, these values were determined:

$$\alpha = 0.062 \frac{\text{Volts}}{\text{Degree}}$$

$$\alpha = 15.105 \frac{\text{Degrees}}{\text{Volt}}$$

I would also like to add that in order to calibrate α to get a perfect vertical = 0, a value of 0.09 needed to be added. The same applies to θ where 0.322 needs to be added.

3.2 DC Motor Transfer Function and Parameters

Definitions:

$$\theta(t) = AngularPosition$$
$$\dot{\theta}(t) = AngularVelocity$$
$$\triangle t = t_{10\%} - t_{90\%}$$
$$90\% = e^{-t_{10\%}/\tau}$$
$$10\% = e^{-t_{90\%}/\tau}$$

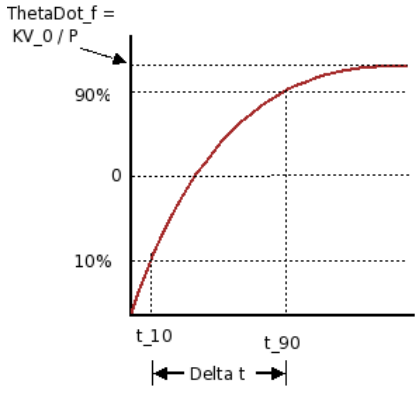
The Math:

$$\frac{s\theta(s)}{V_a(s)} = \frac{K}{s+P}$$
$$\text{Let } V_a(s) = \frac{V_0}{s}$$
$$s\theta(s) = \frac{KV_0}{(S+P)S} = \frac{KV_0}{\frac{P}{S}} - \frac{\frac{KV_0}{P}}{s+P}$$
$$L^{-1} \Rightarrow \dot{\theta}(t) = \frac{KV_0}{P}(1 - e^{-t/(1/P)})$$
$$\dot{\theta}(t) = (\dot{\theta}_i - \dot{\theta}_f)e^{-pt} + \dot{\theta}_f$$

Final equations:

$$\dot{\theta}_f = \frac{KV_0}{P} \tag{1}$$
$$\frac{1}{P} = \tau = \frac{\triangle t}{ln(9)} \tag{2}$$

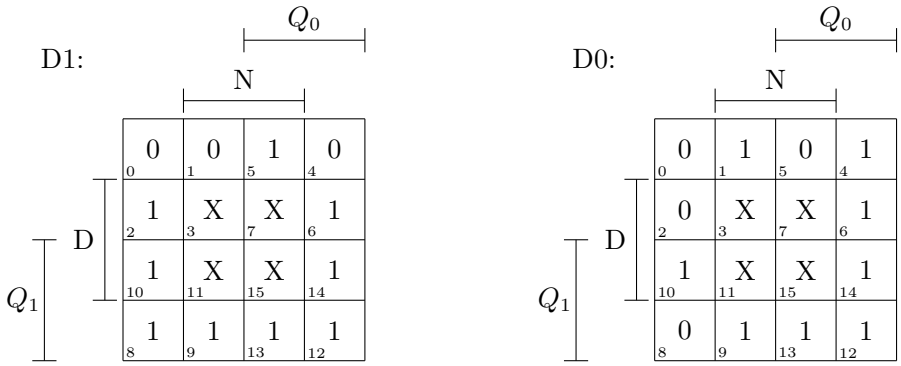
Graphically (Refer to Equation 1 and Equation 2) :



This section consists of tables and reductions which were used in this laboratory exercise.

PS	**D**	**N**	**NS**	**P**
\$0.00	0	0	\$0.00	0
	0	1	\$0.05	0
	1	0	\$0.10	0
	1	1	–	–
\$0.05	0	0	\$0.05	0
	0	1	\$0.10	0
	1	0	\$0.15	0
	1	1	–	–
\$0.10	0	0	\$0.10	0
	0	1	\$0.15	0
	1	0	\$0.15	0
	1	1	–	–
\$0.15	–	–	\$0.15	1

表 1: Symbolic Transition Table



(a) D1 = Q_1+D+Q_0N (b) D0 = $\bar{Q}_0$N + $Q_0\bar{N}$ + Q_1N + Q_1D

表 2: Karnaugh maps and the simplified results of the logic.

4 Discussion & Conclusion

The goal of this lab was to re-design the LED/Switch system to include a hardware timer. By pressing eight different combinations of the three buttons, the LEDs on the board were to act in different ways using these timers. There was not a Q&A requirement for this lab.

I was able to accomplish the requirements of the lab by utilizing the `IntMgrTimerExample.c` project found within the analog devices example programs folder (and mentioned in the class lecture). There were some stumbling blocks to overcome. The most difficult for myself was actually getting the period of the LEDs just right. I was able to get it very close to the $333.3ms$, $666.7ms$, and $1s$periods, but not exactly. My first method of getting these periods right was to take the clock speed in MHz, find the period by taking the inverse of the clock speed, and then solving for the value in hex that was needed to get the right period. This didn't yeild very accurate results at all, and so I then went through a trial and error session until I got a value of $1.1ms$. I used this value in hex to calculate the other periods. The results of this method can be seen in Figure 1 above in the schematics section.

Another observation I would like to point out is that I put all of my logic within the interrupts themselves. I feel that this was a hacked way of doing the lab to save time and that it's probably not the best programming method. After I was completed with my lab, I viewed other students solutions and they just seemed more elegant. Interestingly enough, the other students weren't incredibly happy with their solution either. If I were to go back and do this lab again, I would invest more time in both understanding how to utilze the interrupts as well as find a more elegant solution to blink the lights.

All in all, this laboratory gave me an insight on how interrupts work and I hope to be able to apply them to following labs…