Continuous Improvement in Teaching Microprocessor Systems Design A Review of Efforts in Using Different Tools, Techniques and Methods to Satisfy Students' Needs

Prof. Jie Sheng, University of Washington, Tacoma

Jie Sheng received her Ph.D. in Electrical Engineering in 2002 from the University of Alberta, Canada. Since then, she has been an NSERC Postdoctoral Fellow at the University of Illinois, Urbana-Champaign; a Lecturer at the University of New South Wales, Australia; and an Assistant Professor at DigiPen Institute of Technology, Redmond, USA. Sheng is currently an Associate Professor at the School of Engineering and Technology, University of Washington, Tacoma. Her research interests include signals and systems, embedded systems, robotics, and engineering education.

Continuous Improvement in Teaching Microprocessor Systems Design

A Review of Efforts in Using Different Tools, Techniques and Methods to Satisfy Students' Needs

Jie Sheng
School of Engineering and Technology
University of Washington, Tacoma
1900 Commerce Street, Tacoma 98402 WA
email: shengj2@uw.edu

Abstract

Microprocessor Systems Design is a core course in our curricula of both Computer Engineering and Systems (CES) program and Electrical Engineering (EE) program. It is offered to seniors in the autumn quarter and requires prerequisite on Computer Architecture which covers subjects including instruction set design, and assembly programming.

As a continuation of a 200 level core course - Introduction to Logic Design, and a 300 level core course - Digital Systems Design with FPGA using Verilog, also functioning as a bridge to Senior Project, our 400 level Microprocessor Systems Design course focuses on introducing hardware and software design techniques for microprocessor-based systems.

Back to a decade ago, when first designing this course, several processor families were considered and compared. We finally chose Microchip's 8-bit microcontroller PIC18F4520 as the study topic; correspondingly, the tools included Microchip PICDEM2 Plus board, MPLAB IDE as well as the MPLAB ICD2 evaluation kit. With the advancement of techniques, and to meet students' needs, we redesigned the course and switched to Texas Instrument's Tiva C series Microcontroller TM4C1294NCPDT - an IoT enabled High performance 32-bit ARM® Cortex®-M4F based MCU. Labs and course project were designed based on the Connected LaunchPad EK-TM4C1294XL; students got programming experience with both Keil µVision IDE and Code Composer Studio (CCS) IDE.

To achieve teaching effectiveness, the main method we took is the project-centered pedagogy which has been endorsed at many universities world-wide. Each lab is regarded as a small project. After the concepts and basic principles are delivered and discussed during lectures, students need to find out all the necessary information by reading textbooks and digging into piles of technical documents, and generate the problem solution, which should also satisfy the project requirements.

Taking into account ABET assessment criteria and changes of students group - CES students only in the first several years and a mixed class with both CES and EE students recently, we also adopted the strategy of cooperative learning. In addition to encouraging students to contribute to discussions on weekly small projects/labs, the final course project requires group work. Each group was formed by members with different background, e.g., one from CES program and another from EE program. Individual efforts were assessed based on group work evaluation to ensure fairness and equity. According to students' feedback, the cooperative learning method has successfully promoted students' learning and decision making; it also greatly enhanced students' racial tolerance and critical thinking capability.

The contribution of this paper is that we provide a review to share our experience in teaching Microprocessor Systems Design in the past decade. Details to be presented include: (1) how we design

our curriculum course sequence to ensure students get both the fundamentals and the hands-on exercise in one quarter; (2) how we help prepare students for their future career by teaching them state-of-the-art tools and techniques; (3) how we continuously improve our teaching methods by considering ABET assessment criteria, students' course evaluation/feedback, and changes in the students group caused by program's expansion. The effectiveness of our teaching is supported and verified by students' evaluations.

I. Introduction

Microprocessors/Microcontrollers (µPs/ µCs) are the brain of modern embedded digital systems and have been listed as one of the cores of almost all college-level engineering program curriculums. Teaching μPs/ μCs has been widely discussed in literatures; it can be done in various ways depending on the group of students. Examples include using FPGA to deliver the basic concepts of μPs/ μCs as presented in [1,2], and using troubleshooting problem-solving method to teach high school students as presented in [3], just to name a few. Back to 90's a traditional course on μPs/ μCs mainly involves assembly programming and software-only lab using simulators as mentioned in [4] and the references therein. Nowadays, more and more college-level courses on μPs/ μCs would cover scopes from the architecture of processors to the big picture of embedded systems, and combine lectures with mixed software/hardware-based lab experience. This trend of introducing the ever-growing complexity of μPs/ μCs to students and preparing them to design larger and more complex systems have brought educators big challenges and discussions on how to improve the teaching effectiveness as given in [5]. In our course on μPs/ μCs, we have followed this trend by bringing both fundamentals and hands-on experience to students. In particular, considering each lab as a small project, students learned subjects by project-centered pedagogy [6]; assigning students with different background as a team, they benefited from the cooperative learning [7]. More details will be provided in later sections.

The School of Engineering and Technology (SET) at the University of Washington Tacoma (UWT) was launched in 2001 and named initially as the Institute of Technology. It currently has two engineering programs: the Computer Engineering and Systems (CES) program started in 2007 and got ABET accreditation in 2011; the Electrical Engineering (EE) program was newly added in 2017 and just went through the ABET accreditation review. TCES430 – Microprocessor Systems Design is currently a core course of both engineering programs. It was offered to CES students only before 2017 and now to all the engineering students.

This paper provides a review of our continuous improvement in teaching TCES430 in the past decade. In particular, our efforts in using different platforms, tools/techniques, and methods to satisfy students' needs; meanwhile to meet changes of the class size, especially when students came from different engineering disciplines.

The remaining of this paper is organized as follows: Section II presents our course design focusing on bringing hands-on experience to students based on project-centered pedagogy. We also give details about two platforms and relevant tools to show our efforts in giving students the most-up-to-date skills and techniques. Section III discusses the change of our course population, and accordingly our efforts in satisfying the diversified student groups. Section IV shows how we map TCES430 to ABET Student Outcomes and contribute to programs' ABET accreditation. Section V will share some students' feedback from course evaluation, followed by conclusions in Section VI where a brief summary and future work will be given.

II. Continuous Improvement in Course Design

As an urban-serving campus, the majority of UWT students are transferred students from one of the many community colleges in the area or from other universities. The mission of engineering programs are to prepare our students with the up-to-date techniques and skills, and provide students with the fundamental knowledge and skills needed to be responsible and productive engineers who can improve the quality of life in the community and become leaders in the field [8, 9]. Meeting ABET criteria [7], the TCES430 - Microprocessor Systems Design course adopts a model of 4 hours lecture + 2 hours lab per week. A continuation of a 200 level core course - Introduction to Logic Design, and a 300 level core course - Digital Systems Design with FPGA using Verilog, also functioning as a bridge to Senior Project, TCES430 focuses on introducing hardware and software design techniques for microprocessor-based systems. It aims to (1) give students experience designing and implementing a system using current technology and components; (2) provide students the opportunity to interface microprocessors to external devices; (3) give students experience using state-of-the-art development systems and procedures.

Back to 2008, when the course was first designed to be offered, we examined several processor families that were taught by most universities at that time; vendors including Freescales, Renesas, Microchips, and TI. We finally chose PIC18F452/4520 - the 8-bit microcontroller from Microchip Technology [10] for its easy-to-understand RISC architecture and the strong technical support. Students were given opportunities to program the platform PICDEM2 board [11] with an in-circuit programmer/debugger MPLAB® ICD 2. In addition to the technical datasheet [12], we also adopted the textbook by Robert Reese [13]. A master course schedule is given in Table 1.

Table 1: Weekly schedule of TCES430 in fall 2009

Week	TCES 430 Topics	Book Chapters	Labs
1	Review of digital logic	1	
2	Architecture and organization PIC18FXX2 introduction	2,3	Intro Lab
3	Assembly language Branching, logical and looping	4	Assembly Lab
4	C language Signed operations	5	C Language Lab
5	Arrays and pointers Subroutines	6	Lab3
6	Parallel port I/O Midterm exam	8	
7	Asynchronous serial I/O	9	Lab4
8	Interrupts	10	
9	Timers	10,13	
10	Synchronous serial I/O	11	Lab5
11	A/D conversion	12	Lab6
12	Final exam		

As shown by the schedule, it is challenging for both students and educators to cover a range of topics within 11 weeks. From the architecture including instruction set to assembly programming, and later C programming to interfacing with external world using a variety of available peripherals, we found project-centered pedagogy extremely helpful, similar as those shared in [6]. Students enhanced their concepts and knowledge learned from class lectures by sitting in the lab working on platform board. We are using the following example to show how we design, test and assess students' learning on the subject regarding one serial I/O protocol - I2C, which is a synchronous communication protocol and has been widely used in connecting processor and small IC chips like EEPROM, RTC, sensors, etc. In the exam, students will be checked on their knowledge about basics of I2C like number of bits of each frame in the I2C transmission as shown in Fig. 1. Students were also checked individually regarding how much they have learned from lab experience (they were allowed to work on lab assignments in groups). A question in the final exam was designed for this purpose, and is given in Fig. 2.

TCES430, Autumn 2009

(d) What is the total number of I2C bit times involved when to writing one byte to the 24LC256 serial EEPROM? Assume that the start and stop conditions each count as one bit time and that the EEPROM is ready for a write, so that no polling is necessary.

Fig. 1. A basic I2C question in TCES430 exam

TCES430, Autumn 2009

4. (30 marks total) In the following, for any required I2C functionality, use subroutine calls *i2c_start(), i2c_rstart(), i2c_stop(), unsigned char i2c_send(unsigned char data), unsigned char i2c_receive(unsigned char ackbit), i2c_init(), delayms(unsigned char cnt)* as you used in the lab. Remember that if you use i2c_send, you must pass in as an argument the byte that is to be written to the I2C bus. If you use i2c_receive, you must pass in an as argument the bit value to be sent back as the acknowledge bit value. If you use delayms, you must pass in an as argument the delay value.

   (a) (15 marks) Write C code for main() which reads two bytes starting at location 0x7A40 from the 24LC256 EEPROM, where the 24LC256 has A2 = 0, A1 = 1 and A0 = 1. The 2nd byte read from EEPROM must be sent to one of two different MAX517 DACs connected to the I2C bus (Address format of MAX517 DAC is shown in Figure 3). If the first byte is zero, write the 2nd byte to a MAX 517 which has its A1,A0 pins as A0 = 0 and A1 = 1; if the first byte is not zero, write the 2nd byte to a MAX 517 where A0 = 1 and A1 = 0. Assume that the EEPROM is ready for a read operation, so that no polling is necessary.

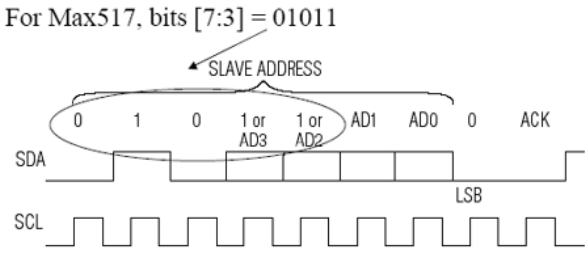

Figure 3

   (b) (15 marks) Write C code for main() which successively retrieves 64 strings (63 printed characters and a null character to end the string) stored in program memory starting at location 0x200, writes them to starting location 0x7A40 of the 24LC256 EEPROM and reads them back to the data memory you defined. You should use pointers to retrieve data. You should use I2C synchronous serial communication to talk to the EEPROM with full page write/read operations, and assume 24LC256 has A2 = 0, A1 = 0 and A0 = 0.

Fig. 2. A programming I2C question in exam checking Individual efforts

One feature of our teaching of μPs/μCs is that students are required to write to low level registers directly instead of calling existing library functions. Still using I2C lab as an example, students needed to generate the right sequences of calling functions in transmitting/receiving data as shown below in Fig. 3, with all the functions writing to low levels by referring to datasheets [12], as shown in Fig. 4.

```
madd_hi = 0x00;                 //high memory address
madd_lo = 0x10;                 //low memory address
while (1){                      //0x-0 is page boundary
        i2c_start();            //initiate start
        i2c_send(waddr);        //send device address w/ write
        i2c_send(madd_hi);      //send hi byte mem address
        i2c_send(madd_lo);      //send lo byte mem address
        i2c_send(data1);        //send first data byte
        i2c_send(data2);        //send second data byte
        i2c_stop();             //initiate stop
        delayms(10);
        i2c_start();            //initiate start
        i2c_send(waddr);        //send device address w/ write
        i2c_send(madd_hi);      //send hi byte mem address
        i2c_send(madd_lo);      //send lo byte mem address
        i2c_rstart();           //send restart
        rdata1 = i2c_receive(0); //receive first byte, ack
        rdata2 = i2c_receive(1); //receive second byte, noack
        i2c_stop();             //initiate stop
        delayms(10);
        data1 = data1 << 1;     //change data1
        data2 = data2 << 1;     //change data2
        delayms(10);
```

Fig. 3. Implementing I2C communication by calling functions in `main()`

```
void i2c_idle(void){
        unsigned char stat;
        do{
                stat = SSPCON2 & 0x1f;      //i2c function control status
        } while(stat | SSPSTATbits.R);      //wait for stat = 0
        return;                             //and no transmit
}

void i2c_start(void){
        i2c_idle();                         //wait for idle bus
        SSPCON2bits.SEN = 1;                //initiate start
        while(SSPCON2bits.SEN);             //wait for start to finish
        return;
}

void i2c_rstart(void){
        i2c_idle();                         //wait for idle bus
        SSPCON2bits.RSEN = 1;               //initiate restart
        while(SSPCON2bits.RSEN);            //wait for restart to finish
        return;
}
```

Fig. 4. Example code segments of developing I2C functions by writing to registers directly

We note here that when the microcontroller is 8-bit like PIC18F452/4520, looking for information from a 400 pages datasheet already requires certain amount of time. It becomes overwhelmingly time-consuming when the microcontroller becomes more complex.

With the advancement of new technology and to satisfy students' needs in grasping a powerful microcontroller, we redesigned the course TCES430; from fall 2015 we switched to a different platform - TI's TM4C1294 Connected LaunchPad Evaluation Kit which highlights TM4C1294NCPDT MCU [14]. The course schedule used in fall 2019 was given below in Table 2. Although the schedules shown in Table 1 and 2 look very similar, the architecture including ISA of ARM Cortex M4F is far more complicated than that of PIC18F452/4520, e.g., the datasheet of TM4C1294NCPDT is nearly 2000 pages [15]!

Table 2: Weekly schedule of TCES430 in fall 2019

Week	TCES 430 Topics	Labs
1	Course Introduction; Review on Digital Logic; Intro. to IDEs; Intro. to ARM	No Lab
2	ARM Cortex™-M Assembly	Lab 1 – Intro. to Assembly Programming using Keil MDK
3	ARM Cortex™-M Architecture	Lab 2 – Assembly Programming: Branches and Function Calling
4	Basic concepts of I/O and TI uC I/O	Lab 3 – Intro. to C Programming using Code Composer Studio IDE
5	Modular Programming	Lab 4 – Keil C Project: Switching Interfacing using Polling Method
6	Midterm review; Midterm exam	Midterm; No Lab
7	Timers and Interrupts	Lab 5 – CCS C Project: Interrupts and Timer
8	Asynchronous serial I/O: UART	Lab 6 – CCS C Project: Asyn. Serial Communication
9	A/D conversion	Thanksgiving; No Lab
10	Synchronous serial I/O: I2XC	Course Project
11	Embedded ARM Applications	Course Project (cont.)
12	Final exam	

Our efforts in effectively teaching ARM-Cortex M4F microcontroller - TM4C1294NCPDT have been greatly benefited from materials shared by TI as well as Prof. J.W. Valvano at the University of Texas at Austin. One of the textbooks we mandatorily required is authored by Prof. Valvano [16]; half of our lab assignments were from TI's training workshop materials [17]. Equipped with knowledges of computer architecture and MIPS assembly programming from pre-requisite course, students spent the first several weeks to pick up basics of ARM Cortex M4F instructions as well as assembly programming. In align with this learning curve, we used ARM®'s Keil™ μVision IDE for assembly programming assignments. Once we started C programming to communicate with external world using different peripherals, we switched to

a combination of tools including TI's Eclipse-based Code Composer Studio™, TivaWare™, as well as TI's training workshop material [17].

Due to the amount of technical documents' reading involved, we spent certain time in lectures teaching students how to quickly find the useful information from thousands of pages. We still required students write to registers directly in their homework assignments, for example, one requires students to manage interrupt configuration and develop interrupt service routine (ISR) on the TM4C1294 Connected LaunchPad; one requires to generate an 1-second interrupt to control the blinking of LED1 on board using two different types of timers available on board – the general purpose 16-bit Timer 1A, and the 24-bit System Timer. Hints were given in lectures so the students could narrow down the number of registers relevant to the required tasks. While during the lab time, students were given opportunities to try three labs instructed by TI's training workshop [17], including Lab02 - flashing LEDs (the codes were given below), Lab05 – interrupts, and Lab10 - UART communication. As an example, when working on the flashing LEDs lab, students could do just copy and paste to build a CCS project and then download the code to board. They then observed how LEDs were controlled. By looking into the datasheet [15] and TivaWare Peripheral Driver Library [18], the class were required to understand how each library function was developed by writing to registers, and were encouraged to translate these labs into the version by writing to registers directly by them own.

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"

uint8_t ui8PinData=1;

int main(void)
{
      SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ | SYSCTL_OSC_MAIN |
SYSCTL_USE_PLL | SYSCTL_CFG_VCO_480), 120000000);

      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);
      GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN_0|GPIO_PIN_1);
      GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_0|GPIO_PIN_1, 0x00);

      while(1)
      {
            GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_0 | GPIO_PIN_1,
ui8PinData);
            SysCtlDelay(2000000);
            if(ui8PinData==4) {ui8PinData=1;} else {ui8PinData=ui8PinData*2;}
      }
}
```

The purpose of these lab design is to expose students to different IDEs and different solutions to practical scenarios, then push them find the commonness among these solutions, so that by the end of the quarter, they can quickly integrate the skills of (1) writing to registers directly; and (2) calling available TivaWare Peripheral driver functions, to finish their course project in groups.

In the course project, each group was provided the following devices:

- a temperature sensor
- a real time clock
- a 7-segment Display and
- an LCD

they were expected

- To use the ADC module in reading temperature sensor measurement (required),
- To use the I2C module in communicating with RTC (required), and
- To display measured data on a 7-segment display or an LCD (optional).

Each group was allowed the flexibility to fulfill the requirements by calling TivaWare functions only, or also writing to registers directly. One extra point (to the final course grade) would be given if they could apply both skills to achieve the goals. As the basic requirements the group should be able to: (1) correctly read the temperature sensor; (2) correctly read the RTC. Although they could monitor the readings by simply looking at IDE's watch window, or UART terminals, students could earn another extra point (to the final course grade) if they were be able to display the readings correctly on either a 7-segment display or an LCD.

It turned out that one out of 17 groups successfully finished all the tasks including the extra 2 bonus points requirements. 10 groups successfully displayed their results using either LCD or 7-segment LEDs, as shown in Fig. 5 and Fig. 6.

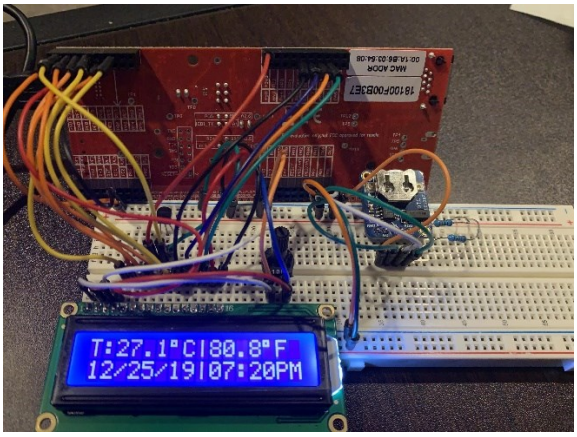

Fig. 5. Displaying sensor readings on LCD

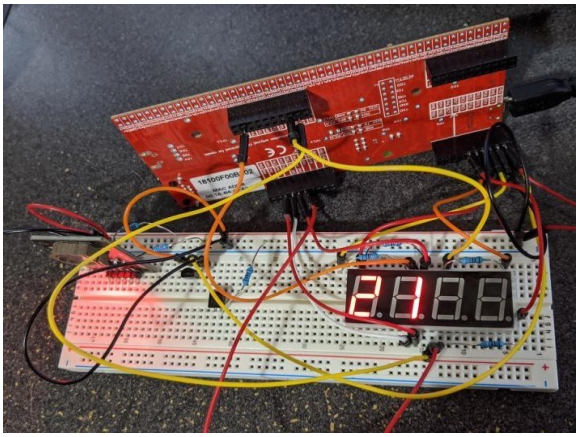

Fig. 6. Displaying sensor readings on 7-seg LEDs

Most groups monitored their results using either the watching windows of IDEs or terminals via UART communication (which they learned from one of the lab assignments). As an illustration of students' project, reading the temperature sensor using ADC peripheral can be achieved by writing to registers directly as below (only main code segments were given here):

```
//This program reads analog temperature data from a
//temperature sensor (Analog Devices TMP36), and converts the
//analog data to a digital format via the on-board
//ADC on the TivaWare micro-controller. ADC module0 is used with
//sample sequencer 3 (SS3).
```

```
:
:
:
void Delay ( unsigned long counter);
void GPIO_init (void);
void ADC_init (void);
void ADC_int_handler (void);

// User main program
int main (){
    //initialize GPIO
    GPIO_init();

    //mask the NVIC to ensure ADC module 0 sample sequencer 3 will generate
interrupt
    NVIC_EN1_R |= NVIC_EN1_INT33;

    //initialize ADC
    ADC_init();

    while(1) {};

}

void ADC_int_handler (){
    //clear interrupt
    ADC_ISC_R |= 0x8;
    int result = ADC_SSFIFO3_R & 0xfff;
}

void GPIO_init (){
    SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOE; //enable clock for PORT E
    Delay(DELAY);
    GPIO_PORTE_AFSEL_R |= GPIO_PIN_4;     //allow GPIO function as peripheral
    GPIO_PORTE_DEN_R   &= ~GPIO_PIN_4;    //PortE pin4 as analog signals
    GPIO_PORTE_AMSEL_R |= GPIO_PIN_4;     //PortE pin4 as analog function
}

void ADC_init (){
    SYSTCTL_RCGCADC |= ADC_MODULE;        //enable clock for ADC module0
    Delay(DELAY);
    PLL_FREQ |= PLL_MASK;                 //set PLL freq to system clock freq
    ADC_CLK_CONFIG |= 0x00;
    ADC_ACTSS_R &= ~0x8;                 //disable sample sequencer3
    ADC_EMUX_R |= 0xF000;                //set MSB for continuous samplling
    ADC_SSMUX3_R |= 0x9;                 //enable AIN9 to be sequencer input
    ADC_SSCTL3_R |= 0x6;                 //enable raw interrupt
    ADC_IM_R |= 0x8;                     //control for SS3 interrupt
    ADC_ACTSS_R |= 0x8;                  //enable sample sequencer (SS) 3
}

void Delay ( unsigned long counter ){
    unsigned long i = 0;
    for (i =0; i< counter ; i ++) ;
}
```

The same goal can also be achieved by calling TivaWare Peripheral Driver Library functions (again we only gave main code segments here):

```
int main(void) {
    // Set clock frequency to 16 MHz
    SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ | SYSCTL_OSC_MAIN | SYSCTL_USE_PLL
| SYSCTL_CFG_VCO_480), SYSTEM_CLK_FREQ);

    // Setup clock for Port E module
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);

    // Setup clock for ADC 0 module
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);

    // Define inputs and outputs
    // Set PortE pin4 as an input
    GPIOPinTypeGPIOInput(GPIO_PORTE_BASE, GPIO_PIN_4);

    // Set PortE pin4 as analog-to-digital converter input
    GPIOPinTypeADC(GPIO_PORTE_BASE,GPIO_PIN_4);

    // Configure the clock for the ADC0 module
    ADCClockConfigSet(ADC0_BASE, ADC_CLOCK_SRC_PLL | ADC_CLOCK_RATE_EIGHTH ,
30);

    // Enable Sample Sequence 3 with continual ADC triggering
    ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_ALWAYS, 0);

    // Enable Sample Sequence 3 interrupts at the end of the sequencing
    ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_CH13 | ADC_CTL_IE |
ADC_CTL_END);

    // Enable Sample Sequencer 3
    ADCSequenceEnable(ADC0_BASE, 3);

    // Enable interrupt for Sample Sequencer
    ADCIntEnable(ADC0_BASE, 3);

    // Enable NVIC controller
    IntEnable(INT_ADC0SS3);

    // Enable global interrupts (allows the processor respond to interrupts)
    IntMasterEnable();

    while(1);
}

void ADC0SS3_Handler(void) {
    // Clear the ADC interrupt
    ADCIntClear(ADC0_BASE, 3);
    // Read ADC Value.
    ADCSequenceDataGet(ADC0_BASE, 3, DATA);
    // reading the 12 bit result
    int result = DATA[0]&0xFFF;
```

In short, project-centered pedagogy works effectively in teaching μPs/μCs at UWT. In the past decade, our engineering students benefited greatly by this learning-by-working method. Hands-on experience enhanced their understanding of fundamentals of μPs/μCs, and helped them smoothly into the stage of senior project design. Exposing them to the powerful ARM MCUs challenged them with reading thousands of pages of technical documents, but also trained them to grasp skills required by future industrial projects. Students' reflection on this effectiveness will be given later in Section IV.

III. Efforts in Meeting Students' Needs from Different Disciplines

Starting 2017, UWT launched the EE program, and TCES430 was offered to both CES and EE students as a big class. This brought new challenges to effective teaching due to (1) the large class size (2) diverse engineering backgrounds of the student group.

Our efforts to ensure students' learning outcomes include the following:

(1) To maximize project-centered pedagogy benefits, we offered multiple lab sessions so that each lab session won't be overcrowded and each student can have his/her questions answered during the lab time. We also recorded all the lecturing sessions so that students can watch repeatedly at their own pace in case they missed something during the lecturing time.
(2) To minimize the negative impact brought by diverse engineering backgrounds, we strongly encouraged cooperative learning [7] in both lecturing and lab times. CES and EE students were mixed into groups to participate in-class activities/discussions and lab/projects. We also helped to form groups taking into account their gender identity, religious beliefs, heritage, and life experience.
(3) Due to the background difference, we offered separate 300 level computer architecture prerequisite courses to CES and EE students. Before, both courses used the same textbook by Patterson and Hennessy [19]; in winter 2019, the course EE students took used a different version on ARM (instead of MIPS). The purpose was to help reducing the gap between CES and EE students on their computer programming fluency when taking TCES430 together in fall 2019. Not surprisingly, with the foundation they built earlier in the prerequisite, several EE students even behave superiorly over their CES peers in the fall 2019 quarter.

IV. Measuring Outcomes and Objectives for ABET Accreditation

As a core course in both CES and EE curriculums, TCES430 has been used to mapping Student Outcomes (SO) 2 and 3 of ABET Criteria [20].

> *SO (2). an ability to apply engineering design to produce solutions that meet specified needs with consideration of public health, safety, and welfare, as well as global, cultural, social, environmental, and economic factors*
>
> *SO (3). an ability to communicate effectively with a range of audiences*

In our design of TCES430, we have used following rationales for the selected student outcomes:

> SO (2) -- Students follow the engineering design process to design, develop, implement and test the course projects using the required ARM microprocessor evaluation board. Their solutions should satisfy practical limitations due to technological, economic, and safety constraints.

SO (3) -- In one of the required labs, students study and configure the analog-to-digital module on the microcontroller. They will collect and analyze the data when the external information is measured and sampled.

And the following assessment rubrics were applied to assess student work described above. Table 3 shows the rubric for SO (2), and Table 4 is the rubric for SO (3).

The CES program at SET got ABET accreditation in 2011; the EE program started in 2017 and just went through the ABET accreditation review. The success couldn't be achieved without the team work in all the curriculum courses. Efforts in improving TCES430 we presented in this paper is just one of them.

Table 3: Assessment Rubric of SO (2)

S.O. 2) An ability to apply engineering design to produce solutions that meet specified needs with consideration of public health, safety, and welfare, as well as global, cultural, social, environmental, and economic factors.				
	NOVICE	**APPRENTICE**	**PROFICIENT**	**EXPERT**
Descriptor	**0**	**1**	**2**	**3**
understanding of design objectives	Does not understand design objectives; haphazard approach	Has a narrow view of what is involved in the design and misses key principles and components required for a complete design process	Produces a design statement that is generally correct; may be missing some prior knowledge, but understands the design objectives	Develops a design strategy with clear objectives, including a plan of attack, decomposition of work into subtasks, development of a timetable
interpretation of prior knowledge into design strategy	Unable to relate prior knowledge to the design problem	Uses prior knowledge occasionally to design individual pieces of equipment when guided to do so	Uses prior knowledge frequently to design individual pieces of equipment when guided to do so	Understands how areas interrelate and demonstrates ability to integrate prior knowledge into a new problem
implementation of design strategy	Does not implement a design strategy	Conducts design as needs arise; has no formal strategy or management of the project	Selects and performs appropriate stage at the appropriate point in the design project; may not re-evaluate progress or need to change strategy	Produces a design strategy and uses it to guide the design project; re-evaluate progress and potential need to change strategy
use of models of simulation software	No use of models or simulation software	May not use models or uses limited tools that do not provide appropriate information; does calculation disconnected from design decisions	Selects and performs models or simulations at appropriate points in the project; may not analyze output as part of a design strategy	Selects and performs models or simulations at appropriate points in the project; evaluates not only model outputs but also quality and appropriateness of model for given task
Realistic Design Constraints	Some realistic constraints considered, but many relevant constraints ignored	Most realistic constraints considered, but a few relevant constraints ignored	Constraints largely mirror those that would be considered in industry	Constraints fully mirror those that would be considered in industry
Creativity	Mimics existing applications without showing creativity	Design shows very limited creativity	Design shows reasonable creativity	Design shows true originality and creativity
Research	No research of data and information accomplished	Some attempt at searching data and information related to the design	Moderate research of data and information related to the design	Excellent research of data and information related to the design
%Class sampled_____ Total # Students_____				
Number of students scoring 2 or higher: Percentage:	_____			
Number of students scoring 3 or higher: Percentage:	_____			

Table 4: Assessment Rubric of SO (3)

S.O. 3) An ability to communicate effectively with a range of audiences				
	NOVICE	APPRENTICE	PROFICIENT	EXPERT
Descriptor	0	1	2	3
articulation of ideas-written	Student does not articulate ideas at all	Text rambles, points made are only understood with repeated reading, and key points are not organized	Articulates ideas, but writing is somewhat disjointed and difficult to follow	Articulates ideas clearly and concisely
professionalism -written	The writing style is inappropriate for the audience and for the assignment	Style is informal or inappropriate, jargon is used, improper voice, tense, etc	Usually uses good professional writing style	Uses good professional writing style
organization - written	Little or no structure or organization is used	Some structure and organization is used	Generally organized well but paragraphs combine multiple thoughts or sections are identified clearly	Organized written materials in a logical sequence to enhance the reader's comprehension
quality of work-written	Work is not presented neatly; spelling/grammar errors present throughout more than 1/3rd of the paper	Work is not neatly presented throughout, one or two spelling/grammar errors per page	Written work is usually presented neatly and professionally; grammar and spelling are usually correct	Written work is presented neatly and professionally; grammar and spelling are correct
use of graphs/tables	No figures, tables, or graphics are used at all	Figures, tables, and graphics are presented but are flawed (axes mislabeled, no data points, etc)	Use of figures, tables, and graphics that are usually in the proper format	Use of figures, tables, and graphics that are all in the proper format
articulation of ideas-oral	Presentation is unclear and not understandable; omits key material during presentation	Presents key elements adequately but presentation contains excessive or insufficient detail for time allowed and level of audience	Presents key elements adequately; presentation has enough detail and technical content for the time constraint and the audience	Plans and delivers an oral presentation effectively; presentation has enough detail and technical content for the time constraint and the audience
quality of work-oral	Uses poor English; does not listen carefully to questions, does not provide an appropriate answer or is unable to answer	Occasionally uses an inappropriate style of English--too conversational; sometimes misunderstands questions, does not respond appropriately or has trouble answering questions	Usually uses proper English; sometimes misunderstands questions listens and responds to questions	Uses proper English; Listen carefully and responds to questions clearly
organization-oral	Talk is poorly organized	Talk has issues with flow due to organization problems	Presentation is mostly organized	Presentation is organized in a logical sequence
professionalism-oral	Appearance is not professional; presentation is poor (no eye contact, difficult to hear or understand, reads from prepared script, distracting nervous habits)	Appearance is too casual for the circumstances; has some difficulties with the mechanical aspects of the presentation (eye contact is sporadic, sometimes difficult to understand or hear, etc)	Professional appearance; usually presents well mechanically (makes eye contact, can be easily heard, speaks comfortable, no distracting nervous habit)	Professional appearance; presents well mechanically (makes eye contact, can be easily heard, speaks comfortable, no distracting nervous habit)
%Class sampled_____ Total # Students_____				

Number of students scoring 2 or higher: Percentage:	_____
Number of students scoring 3 or higher: Percentage:	_____

V. Students Feedback

Mapping student learning outcomes to ABET criteria, introducing hands-on experience, pushing writing to low level registers, adopting project-centered pedagogy, challenging students with tons of technical documents, and encouraging cooperative learning, have all contributed to students' learning, as indicated by the following quoted student feedback.

"I love labs because we have to use all of the information we learned in lecture."

"This class was a bit harder than 330. My thinking was stretched a LOT!!!!!!!!!!!! it was good though."

"… Very much. Because it dealt with lower-level coding and pin configurations which required a major understanding of how hardware component interactions works."

"This class was intellectually stimulating because the course material is extremely applicable to the real world. It was really cool to learn to code at the register level!"

"Taught me how to use registers and their locations on the boards instead of using functions to program the boards which was cool."

"The concepts covered in class were interesting, and there were many parts of the class that I enjoyed. There was much knowledge that I gained about microprocessors and how to program them through our homework assignments. It also gave me a window in getting software and hardware to communicate with each other, and just how difficult it is to do that effectively."

"This class really stretched my thinking because I needed to combine many elements from previous and current coursework in order to better understand the role that micro-processors and micro-controllers play in designing computer systems. I especially appreciate the opportunity to learn about the subject by studying real world applications like ARM and the Connected Launchpad."

"*Tons of documentation. I actually liked using the spec sheets and other documents, but the ability to develop confidence with a technology varies inversely with the amount of reference material required to implement that technology. Just wish there was more time and more labs.*"

"The lab was very challenging (in a good way). When programming at such a low level there are more opportunities for bugs and other mistakes to arise and I have spent more time troubleshooting both the hardware and software than in any other class. I feel as though this lab has strengthened my confidence in resolving difficult to find issues / bugs."

"The class really requires that you look through multiple pieces of literature (component specifications, data sheets, instructor material, etc.) to be able to accurately program the microprocessor and allow it to communicate with other serially connected devices. If you didn't collect all the critical information, then you couldn't properly put the all the pieces together which would make the assignments work. In this way, the course really stretched your thinking. You had to be able get the information you needed and be able to apply it to the assignment."

"The labs and discussion with other class members regarding class and lab assignments helped a lot."

"Group work contributed the most to my learning. I struggled on my own. When we worked in groups it allowed for correct steps to be taken to finish our projects.''

"Working with other students on homework assignments; lots of things can go wrong programming the controllers and everyone has different strengths to lend and teach each other. Computer Engineering and Electrical Engineering students in the same class helps a lot, since the computer engineers understand programming better, and electrical engineers understand physical aspects better."

VI. Conclusions

As a summary, this paper gives a review and shares our experience in teaching Microprocessor Systems Design at UWT in the past decade. Details presented in the paper include how we design the course to ensure students get both the fundamentals and hands-on exercise within one quarter of 11 weeks; and how we continuously improve our teaching methods to help students ready for their future industrial jobs. We note that the programming codes given in Section II are simply reflecting the step-by-step

procedures, either following the I2C protocol or following the hardware activation rules. Using flow charts might be a good idea to show the logistics behind the design. Many students drew flow charts in their work, but we just skipped them in the paper. We want to highlight our efforts which mainly involve teaching students state-of-the-art tools and techniques, mapping student learning outcomes to ABET criteria, seriously digesting students' course evaluation and feedback, and adjusting teaching strategies accordingly when the student group becomes more diverse than before. Teaching effectiveness is reflected and supported by quoted students' evaluations, which also convinced us that in our future teaching of engineering courses, no matter how advanced the technologies are, how diverse the student group would be, keeping them motivated and engaged, and challenging them both intellectually and hands-on, are always the essentials to guarantee success.

References

[1] J. Olivares, "Teaching Microprocessors Design Using FPGAs," in *Proc. of the IEEE EDUCON 2010*, April 14 - 16, 2009, Madrid, SPAIN, session T1A pp. 1-5.
[2] K. Karyono and A. Wicaksana, "Teaching microprocessor and microcontroller fundamental using FPGA," in *Conference on New Media Studies (CoNMedia)* 2013.
[3] A. Papadimitriou, "Teaching Microprocessors through Troubleshooting Problem-Solving at Technical High Schools," *International Journal of Information and Communication Sciences*, 2(4), pp. 38-44, 2017.
[4] R.B. Reese, "Embedded System Emphasis in an Introductory Microprocessor Course," in *Proc. of the 2005 American Society for Engineering Education Annual Conference & Exposition*, 2005, pp. 10.525.1-7.
[5] R.B. Reese and B.A. Jones, "Improving the Effectiveness of Microcontroller Education," in *Proc. of the IEEE Southeast Con 2010*, 2010, pp. 172-175.
[6] X. Wu, M. Obeng and J. Wang, "Project-centered pedagogy and practice in teaching microprocessor and embedded systems design to undergraduate students," in *Proc. of the IEEE SoutheastCon 2010*, March 2010, pp. 102-105.
[7] R. M. Felder and R. Brent, "Designing and teaching courses to satisfy ABET Engineering criteria", *Journal on Engineering Education*, Vol. 92, No 1, 2003, pp. 7 – 25.
[8] https://www.tacoma.uw.edu/about-uw-tacoma/about-university-washington-tacoma
[9] https://www.tacoma.uw.edu/set/mission-vision-values
[10] https://www.microchip.com/wwwproducts/en/PIC18F4520
[11] https://www.microchip.com/Developmenttools/ProductDetails/DM163022-1
[12] https://ww1.microchip.com/downloads/en/DeviceDoc/39631E.pdf
[13] R.B. Reese, *Microprocessors: From Assembly Language to C Using the PICI8FXX2*, Da Vinci Engineering, 2005.
[14] http://www.ti.com/product/TM4C1294NCPDT
[15] http://www.ti.com/lit/ds/symlink/tm4c1294ncpdt.pdf
[16] J.W. Valvano, *Introduction to ARM Cortex-M Microcontrollers*, CreateSpace Independent Publishing Platform; 5th edition, 2012.
[17] http://software-dl.ti.com/trainingTTO/trainingTTO_public_sw/Tiva_CLP/CLP_Workbook.pdf
[18] https://www.ti.com/lit/ug/spmu298d/spmu298d.pdf
[19] D. Patterson and J. Hennessy, *Computer Organization and Design -- The Hardware/Software Interface*, Morgan Kaufmann; 5 edition, 2013.
[20] https://www.abet.org/accreditation/accreditation-criteria/criteria-for-accrediting-engineering-programs-2019-2020/#GC3