## CA11 - 15%

Build an IoT System that does the following:

* Using an AVR measure the ambient temperature using an LM35 temperature sensor.  Temperature readings should be triggered by an interrupt generated by a user button.
* Once a temperature reading is made upload the reading to the cloud using [sigfox](https://moodle.itb.ie/mod/url/view.php?id=168789" \o "sigfox).
* The arduino systems should be in low-power mode as much as possible to save battery life.
* Forward the uploaded temperature to a Watson IoT instance.
* Provide a [node-RED](https://moodle.itb.ie/mod/url/view.php?id=143219) application on [IBM Cloud](https://moodle.itb.ie/mod/url/view.php?id=143221) that will serve the last transmitted temperature on a webpage.
* Using a Raspberry Pi connected to your Watson IoT instance indicate with a LED whether the temperature is above or below 20C (or appropriate threshold).
* Ensure the application on the Pi starts automatically when the Pi boots.

**Upload:**

* All code from the AVR.
* A video demonstration of your working system - demo a button push resulting in the temperature on the webpage updating and the LED on the Pi changing if the threshold crossed - demo LED going on and off.
* A link to your temperature webpage.
* A screenshot of your [sigfox](https://moodle.itb.ie/mod/url/view.php?id=168789" \o "sigfox) callback setup.
* A screenshot of your Watson IoT "Devices" page showing your Pi and your [sigfox](https://moodle.itb.ie/mod/url/view.php?id=168789" \o "sigfox) devices.
* A screenshot of your [IBM Cloud](https://moodle.itb.ie/mod/url/view.php?id=143221) [node-RED](https://moodle.itb.ie/mod/url/view.php?id=143219) application.
* A screenshot of your Raspberry Pi [node-RED](https://moodle.itb.ie/mod/url/view.php?id=143219) application.

## CA10 - 15%

In this exercise you are asked to read the temperature from an LM35 (see [Component Datasheets](https://moodle.itb.ie/mod/page/view.php?id=160048)) analog temperature sensor when a button is pressed on your board and upload this temperature to the [sigfox backend](https://moodle.itb.ie/mod/url/view.php?id=168790" \o "sigfox backend).

This can be achieved in a number of ways with varying degrees of complexity.

The simplest model is to use polling I/O in all cases - button, [adc](https://moodle.itb.ie/mod/resource/view.php?id=165901), uart.

A more sophisticated (but more complex) approach is to use interrupts where possible/practical and put the AVR in sleep mode as often as possible.  The button, [adc](https://moodle.itb.ie/mod/resource/view.php?id=165901) an uart can all generate interrupts.

To achieve an A grade in this assignment it would be necessary to use interrupts.

In saying that a passing or good grade can be achieved using polling.

A suggested approach is to use the most straightforward approach in each case (button, [adc](https://moodle.itb.ie/mod/resource/view.php?id=165901" \o "ADC), uart) to get the system working and then remove any final polling at a later stage.

Upload:

* Zip of eclipse project folder
* A video showing the system working.  Video the screen of your laptop/PC to show the debug info appearing in [Coolterm](https://moodle.itb.ie/mod/url/view.php?id=163168" \o "Coolterm)and the data appearing on the [sigfox backend](https://moodle.itb.ie/mod/url/view.php?id=168790" \o "sigfox backend) after pressing the button.

## CA9 - 15%

In this exercise you are asked to read the temperature from an LM35 (see [Component Datasheets](https://moodle.itb.ie/mod/page/view.php?id=160048)) analog temperature sensor when a button is pressed on your board and upload this temperature to the [sigfox backend](https://moodle.itb.ie/mod/url/view.php?id=168790" \o "sigfox backend).

This can be achieved in a number of ways with varying degrees of complexity.

The simplest model is to use polling I/O in all cases - button, [adc](https://moodle.itb.ie/mod/resource/view.php?id=165901" \o "ADC), uart.

A more sophisticated (but more complex) approach is to use interrupts where possible/practical and put the AVR in sleep mode as often as possible.  The button, [adc](https://moodle.itb.ie/mod/resource/view.php?id=165901" \o "ADC) an uart can all generate interrupts.

To achieve an A grade in this assignment it would be necessary to use interrupts.

In saying that a passing or good grade can be achieved using polling.

A suggested approach is to use the most straightforward approach in each case (button, [adc](https://moodle.itb.ie/mod/resource/view.php?id=165901" \o "ADC), uart) to get the system working and then remove any final polling at a later stage.

Upload:

* Zip of eclipse project folder
* A video showing the system working.  Video the screen of your laptop/PC to show the debug info appearing in [Coolterm](https://moodle.itb.ie/mod/url/view.php?id=163168" \o "Coolterm) and the data appearing on the [sigfox backend](https://moodle.itb.ie/mod/url/view.php?id=168790" \o "sigfox backend) after pressing the button.

## CA9 - 20%

You are to build a lighting controller system similar to that found in the YakinduLightswitch example series - version 3.

You will be using your keypad for the buttons on the system.  The '1' key will be the "ON" button, the '2' key will be the "OFF" button and the '3' key will be the "MODE" button.

You can either use the "L-LED" (pin 13 / PB5) on the board or any other pin with a LED (not forgetting current limiting resistor) on the board.

Unlike the statechart in the example this system will move between manual and automatic mode using the "MODE" button.  This system will only have a light sensing automatic mode and no motion sensing.

**Your system must operate in two modes:**

* Manual mode - allows switching and off the the light (PB5) using the ON and OFF buttons.
* Automatic mode - the light will come on when it gets dark and go off when it gets bright - experiment with the LDR to derive a suitable threshold value that will allow simulation when the LDR is covered by a hand.
* The MODE button toggles between the two modes.

Your [ADC](https://moodle.itb.ie/mod/resource/view.php?id=165901) code should be in a separate module ([adc](https://moodle.itb.ie/mod/resource/view.php?id=165901" \o "ADC).h and [adc](https://moodle.itb.ie/mod/resource/view.php?id=165901" \o "ADC).c).  The API should be:

* **void adcInit(uint8\_t channel);** //Set up the [ADC](https://moodle.itb.ie/mod/resource/view.php?id=165901) for a single read, AVcc as ref, 128 prescale
* **uint16\_t adcRead(void);**  //Perform a single read from the [ADC](https://moodle.itb.ie/mod/resource/view.php?id=165901). Waits for [ADC](https://moodle.itb.ie/mod/resource/view.php?id=165901) completion.

**Additional Requirements:**

* While running the system should output the current [ADC](https://moodle.itb.ie/mod/resource/view.php?id=165901) reading and the threshold being used on the USART.
* Your system should be driven with a hardware timer and not use the \_delay\_ms() function.

**Upload:**

* Zip your final eclipse project and upload here.
* Also upload a video of the working system.

## CA8 - 5%

Using the [millis.h](https://moodle.itb.ie/mod/resource/view.php?id=202728" \o "millis.h) and [millis.c](https://moodle.itb.ie/mod/resource/view.php?id=202730" \o "millis.c) template files available on moodle implement the millis module.

**Test 1:**

Test your new millis module by using it to control the timing of a blinking LED on your board.

* Toggle the LED every 250mS.
* Put the MCU in IDLE mode between LED toggles.
* Answer these questions in your screencast:
  + What is waking up your CPU?
  + How often is your CPU waking up?
  + What would be the effect of changing the millis module such that the millis count was only incremented every 10ms instead of every 1ms? Could you save some power?  What would need to change?

**Test 2:**

* Make a copy of your project that toggles the LED using the external INT0 interrupt.
* Copy in your [millis.h](https://moodle.itb.ie/mod/resource/view.php?id=202728" \o "millis.h) and [millis.c](https://moodle.itb.ie/mod/resource/view.php?id=202730" \o "millis.c).
* Modify your code (NOT [millis.c](https://moodle.itb.ie/mod/resource/view.php?id=202730" \o "millis.c) or [millis.c](https://moodle.itb.ie/mod/resource/view.php?id=202730" \o "millis.c)) such that it does not use the delay functions (\_delay\_ms() etc).
* Your code should still toggle the LED using the external INT0 interrupt.
* Your code should still debounce the button press correctly and only toggle the LED on a press and not a release.
* Explain how your code works in your screencast.

**Upload:**

* Your [millis.h](https://moodle.itb.ie/mod/resource/view.php?id=202728" \o "millis.h) and [millis.c](https://moodle.itb.ie/mod/resource/view.php?id=202730" \o "millis.c)
* Your [main](https://moodle.itb.ie/mod/resource/view.php?id=166836" \o "main).c from your Test 1 project.
* Your [main](https://moodle.itb.ie/mod/resource/view.php?id=166836" \o "main).c from your Test 2 project.
* One or two screencasts explaining how your code works and answering the questions above.

## CA7 - 5%

Re-implement the functionality of CA5 (keypad) but this time your system should be interrupt driven using the pin-change interrupt mechanism.  Watch the keypad interrupt videos for more information.

Your system should be in sleep mode whenever no key is being pressed.

Your LCD should use your sr595 library to interface it.

**Upload:**

* Zip your entire eclipse project folder (just this project not the entire workspace) and upload here
* A short screencast explaining how the interrupt and power-saving aspects of you code works.

## CA6 - 5%

Design and implement a reusable code library for the 74HC595 8-bit shift register.

Name your files sr595.h and sr595.c.

Make appropriate use of #defines to ensure that a change of pin allocation does not require any changes to sr595.c.

Use the following API:

* sr595Init(); //Set up pin directions etc
* sr595Send(uint8\_t byteToSend); //Send the passed byte via the shift register

Test your new library by connecting a 74HC595 to a LED bar (or 8 individual LEDs) via limiting resistors.

Once complete create a modified version of your LCD library that uses the sr595 library and test.  You should be able to run your code from a previous CA that "bounces" your name across the screen using the new version of the LCD library with no changes to [main](https://moodle.itb.ie/mod/resource/view.php?id=166836" \o "main).c

**Upload:**

* sr595.h and sr595.c
* Modified [lcd.h](https://moodle.itb.ie/mod/resource/view.php?id=194491" \o "lcd.h) and [lcd.c](https://moodle.itb.ie/mod/resource/view.php?id=194494" \o "lcd.c)
* [main](https://moodle.itb.ie/mod/resource/view.php?id=166836).c that simply tests sr595 library.
* [main](https://moodle.itb.ie/mod/resource/view.php?id=166836).c that uses the modified LCD library.
* A short video of the working system.

## CA5 - 3%

* Watch and follow along the [Keypad Interfacing Video](http://moodle.itb.ie/mod/url/view.php?id=162255).  
  Write the keypad.c file and test so that all the keys work.  
    
  Keypad wiring as per the [keypad.h](https://moodle.itb.ie/mod/resource/view.php?id=162416" \o "keypad.h) on moodle.  Rows 1->4 wired to PC0->3.  Columns 1-3 -> wired to PB2-4.
* Build a system that will allow you to enter digits using the keypad and also do a backspace and a clear screen.
  + Use the \* key as the "backspace".
  + To make this work you will need to do the following each time the \* key is pressed:
    - Move the CURSOR (see Table 2 in the [lcd1](https://moodle.itb.ie/mod/resource/view.php?id=161427) document) one place to the LEFT.
    - Write a SPACE ' ' character to blank out whatever needs cleared.
    - Move the CURSOR one place to the LEFT again (this is because it will have moved to the right after entering the SPACE.
  + Use the # key as the "clearscreen" key.
    - To make this work simply issue the "Clear Display" command (see Table 2 in [lcd1](https://moodle.itb.ie/mod/resource/view.php?id=161427)) when the '#' key is pressed.

**Upload:**

* Well commented code. (Zip your entire eclipse project folder for this project).
* A video of the working system - use your phone.
* A screencast video of you explaining how the code works.

## CA4 - 5%

Using the agreed header file provided ([lcd.h](https://moodle.itb.ie/mod/resource/view.php?id=194491" \o "lcd.h)) design, build and test an LCD interface library that can be reused in future C projects.

Ensure that any changes to the physical connection of the LCD only require changes to [lcd.h](https://moodle.itb.ie/mod/resource/view.php?id=194491" \o "lcd.h) and not the accompanying C file.

As proof and testing of your new library provide a test [main](https://moodle.itb.ie/mod/resource/view.php?id=166836) program that exercises the library.  Do this by displaying text on both lines of the LCD (e.g. your name) and then scrolling this text right and left such that it "bounces" from each edge of the screen.  Make sure no cursor or underline are visible.

**Upload:**

* [lcd.h](https://moodle.itb.ie/mod/resource/view.php?id=194491)
* [lcd.c](https://moodle.itb.ie/mod/resource/view.php?id=194494)
* [main](https://moodle.itb.ie/mod/resource/view.php?id=166836).c
* A short video demonstrating your working code.

**N.B.**

* Grading will also consider coding standard issues such as vertical and horizontal layout of code, appropriate commenting, correct definition of #defines etc.

**CA1 - 5% is due**

Fri, 12 Oct 2018, 11:55 PM

Course event

You are to build a simple system that displays a rolling count on a 7-Segment Display.

You can use whichever IDE you prefer but try the Eclipse AVR-Development tools provided in the tools section as we will be migrating to this one exclusively soon.

**N.B.** For all these exercises you will have to temporarily disconnect the segments connected to pins 0 and 1 of the Arduino to download the code to the board.  Reconnect once downloaded.

Firstly wire up your 7-seg display as per the datasheet - see Resources.  
Use the following wiring: seg a, b, c, d, e, f, g, dp wired to Arduino Digital pins 0->7.

* ***Exercise 1:***  
  Create a new Arduino project and write an **Arduino Wiring** program that switches on and off all the segments together once every 500mS.
* **Exercise 2:**  
  Modify Exercise 1 to turn on each segment in turn.  Only one segment at a time should be illuminated.  Each segment should be illuminated for 500mS.
* ***Exercise 3:***Create a new **Arduino Project** and write a program to output a count on your 7-seg display from zero to nine with a one second delay between each increment.  When the count reaches nine it should loop around to zero again and keep going.

**When you are finished upload:**

1. A short video of the working system.
2. Your ".ino" code file.

[MIOT H6011 - Embedded Systems](https://moodle.itb.ie/course/view.php?id=1506)

Sun, 21 Oct 2018, 11:59 PM

Course event

Build a system on your breadboard that will convert the 4-bit binary read in on a bank of switches to the appropriate hex digit displayed on a seven segment display.

* Connect the segments as follows: a->PD0, b-> PD1, c->PD2, d->PD3, e->PD4, f->PD5, g->PD6, dp->PD6.
* You will have to temporarily disconnect PD0 and PD1 each time you want to upload a new program to the arduino board.
* Use DDRD and PORTD instead of pinMode() and digitalWite().
* Use #defines to avoid "magic numbers" in your code.
* Connect your switches to PB0, PB1, PB2, PB3.  Connect the LSB least significant bit to PB0.
* Use [masking](https://moodle.itb.ie/mod/resource/view.php?id=161411) to blank out the upper PB bits.

**Upload**

* A video of your working board.
* Your code.

[MIOT H6011 - Embedded Systems](https://moodle.itb.ie/course/view.php?id=1506)

CA 3

Build a system on your breadboard that will convert the 4-bit binary read in on a bank of switches to the appropriate hex digit displayed on a seven segment display.

* Connect the segments as follows: a->PD0, b-> PD1, c->PD2, d->PD3, e->PD4, f->PD5, g->PD6, dp->PD6.
* You will have to temporarily disconnect PD0 and PD1 each time you want to upload a new program to the arduino board.
* Use DDRD and PORTD instead of pinMode() and digitalWite().
* Use #defines to avoid "magic numbers" in your code.
* Connect your switches to PB0, PB1, PB2, PB3.  Connect the LSB to PB0.
* Use [masking](https://moodle.itb.ie/mod/resource/view.php?id=161411) to blank out the upper PB bits.

**Upload**

* A video of your working board.
* Your code.

Ca 4

***Build an AVR based system that will count button presses and display the count on a seven segment display.***

* Connect the button to PB4.
* Connect the segments of the display as in the previous CA:
  + a->PD0, b-> PD1, c->PD2, d->PD3, e->PD4, f->PD5, g->PD6, dp->PD7
* The count should roll over from 9 to 0. (or A to 0 if you prefer!).
* Write the program in C without using any of the Arduino libraries.  You can use the Arduino IDE though.
* The code should only react to a button press and not a release.
* Debounce your button is software.

***Upload:***

* Well commented C code file. (it may be an .ino file if you use the Arduino IDE.)
* A short video of your working system.