**Methodology**

**Part 1: PWM Output**

Programmable Counter Array (PCA) allows more timing capabilities with less CPU intervention than the standard timer/counters. It is a 16 bit counter which counts up at the PCA clock rate. After counting 65535, the count overflows, and generate an overflow interrupt. The count then resets back to 0 and starts again.

In the AT89C51AC3 microcontroller, all PCA modules can be used as PWM outputs. In this part of the laboratory, module 0 will be used as a PWM output. This involves the following steps:

1. Turn the PCA counter on (CCON register).
2. Set module 0 to be a PWM output (CCAPM0 register). You need to concentrate on the ECOM and PWM bits.
3. Set the PWM duty cycle by changing the CCAP0H register.

In the first step, the CCON register is set to 0x40. This sets the PCA timer on. The second step sets module 0 as the PWM output. This is done using the CCAPM0 register, which corresponds to module 0. This register is set with a value of 0x42. This corresponds to setting bit 1 and 7 to be on. Bit 1 configures the module 0 to be an 8-bit pulse width modulator with output waveform on CEX0 pin. Bit 7 enables the capture mode on module 0. This allows an interrupt to be triggered when the PCA count is equal to some predefined value. The last steps sets the PWM duty cycle. This uses CCAP0H register, which is a PCA high byte compare/capture module 0 register.

Changing the value CCAP0H results in the PWM duty cycle to change, where 0xFF results in a 100% duty cycle, and 0x00 results in a 0% duty cycle and 0x7F corresponds to a 50% duty cycle.

The next task is to use the I/O module to change the value of the across the CCAP0H register. A DC motor is used to be able to control the RPM of the motor by changing the value on the I/O module which in turn changes the value across CCAP0H register. This is done by setting CCAP0H as P0, since the I/O module is connected to port 0 of the microcontroller.

**Part 2: LDRs and External Interrupts**

The circuit below is impmented to observe the variation in resistance of LDR. A table is shown below with three different conditions and the LDR corresponding resistance for that specific condition.

////insert pics and table here

The next part uses a LM311 voltage comparator. A comparator compares two analogue input voltages and generate a digital output. The output is high if the positive input is at a higher voltage than the negative input, and a low output if the negative input voltage is higher than the positive input voltage. It should be noted the analogue inputs (pin2 and 3) should not exceed 5v

For this task, the voltage comparator should output 5V when the LDR detects that it is in the dark. To implement this task, two voltage dividers are used. One for the reference voltage, and one for the LDR voltage.

For the voltage divider that takes the voltage out of the LDR uses a resistor of 1.2kΩ. This value was chosen to allow a 2.5V across the LDR, as the LDR has a resistance of roughly 1.2kΩ during ambient light conditions in the laboratory. For the reference voltage divider, a 2kΩ and a 3kΩ were used to get a 3V output. This allows enough headroom for comparator to not output a 5V in ambient light conditions.

//voltage divider circuits here

The LM311 circuit is set up for a 5V operation, which at pin 3 is connected to the reference voltage divider output, and pin 2 is connected to LDR voltage.

//insert lm311 circuit here

This task is required to count the number of interrupts which is when there is a falling or rising edge. To do this, the interrupt needs to be initialise. The steps is as follows:

1. Enabling external interrupt 0
2. Disabling all other interrupts
3. Specifying the interrupt to be edge triggered.
4. Write an interrupt procedure to increment the count variable whenever there is a falling or rising edge.

The first step is enabling the external interrupt 0. This is done using the IEN0 register, and setting pin 7 to be on. This enables all interrupts. To enable external interrupt 0, the EX0 bit was set to 1. To specify the interrupt to be edge triggered, the TCON register is used. Setting the bit IT0 to 1 selects external interrupt 0 to be edge triggered. Next is to write the interrupt procedure. An interrupt address vector needs to be specified in the heading of the interrupt procedure called ***MyIntHandler().***The interrupt address vector is set to 0 which corresponds to external interrupt 0 which is being used for this task. Whenever a falling edge occurs, the procedure is called. This increments the counter by 1. The current state of the program is paused and is resumed once the interrupt procedure is completed. Within the main function, the microcontroller prints out the counter variable onto the LCD. The at89c51ac3 automatically resets the external interrupt flag once the procedure is completed, so there is no need to turn the interrupt flag manually.

**Part 3: Timers and Internal Interrupts.**

This task requires the microcontroller to generate an interrupt using the internal timers. This implements a Timer 0 which will be use as the clock source for the three PCA modules.

The procedure for this task is as follows:

1. Initialise Timer 0 as an 8 bit timer with auto reload, and turn it on.
2. Set up 3 PCA modules as 16-bit software timers and enable the PCA interrupt for each.
3. Enable compare function for each module
4. Enable match bit for each module
5. Enable PCA counter overflow interrupt
6. Enable the PCA EWC count Pulse Select to use Timer 0 overflow.
7. Turn on PCA timer on.

To initialise timer 0, The TMOD function register is used. Setting timer 0 as an 8 bit timer with auto reload requires bits 0 and bits 1 to be 1 and 0 respectively. This corresponds TMOD to have a value of 0x02. Timer 0 is then turned on by setting the TR0 bit of the TCON register to be high. To set up 3 PCA modules, CCAPM0, CCAPM1 and CCAPM2 are initialised. For each modules, a value 0x49 is set. This sets the ECCFn and the MATn and ECOMn bits for each PCA module. The ECCFn bit enable CCFx bit in CCOn register to generate an interrupt request. The MATn is set when a match of the PCA counter with the compare/capture register sets CCFx bit in CCOn register, flagging an interrupt. ECOMn is enabled to use the compare function for each PCA module. To enable the PCA counter overflow, the CMOD register is used. This corresponds to setting the ECF bit high. This allows the CF bit in the CCON register to generate an interrupt. Since the clock source of the PCA is the timer 0, the PCA EWC count pulse uses the timer 0 overflow. This corresponds bits 1 and 2 to be set 0 and 1 respectively. At the end the PCA timer is turned on by setting the CR bit of the CCON to be 1.

For each PCA module, CCAMPxH is set different values, where x corresponds to the PCA module. This allows the PCA interrupt for each module to be triggerd at different times. This occurs when the PCA mouodle is equal to the values stored in CCAPxH and CCAPxL. CCAPxL is set as 0 for this task.

The interrupt procedure uses the PCA interrupt address 6. Within the interrupt procedure called ***intHandlerPCA()***, it checks each PCA timer flag if an overflow occurs. As a result the count variable associated for each module would increment at different times.

**Part 4: Capturing Interrupts.**

This part of the laboratory uses the PCA module in capture mode to calculate the RPM of the motor. In capture mode, an interrupt is triggered by an external source and the current state of the PCA count is automatically stored.

Knowing the PCA clock frequency of 1Mhz, the time for the PCA counter to count up 1 bit is the inverse of the PCA clock frequency which is 1µs. The difference between the current value stored in the CCAP0H and CCAP0L and the previous value stored in these registers gives us the count difference between the two interrupt events. However, overflows occurred during that time, so the current value stored must be added to the number of overflows that has occurred between the last interrupt. To add the two 8 bit values in the CCAP0H and the CCAP0L registers, a bit shift right is done on the value stored in the CCAP0H, and is added to the value stored in CCAP0L. By subtracting the current value stored with the previous value allows us to get the count difference. 300000 is divided by the count value to get the RPM value. The current value stored in the function registers then becomes the previous value. This computation occurs within the interrupt procedure, which is called whenever the PCA interrupt (address 6) flag is true. Within the PCA interrupt procedure, when the CCF0 interrupt flag is true. This occurs when a capture occurs, which is when a hole passes through the opt switch on the motor. It then checks if any overflow occurs, and if it has, it increments the overflow variable, which will be later used to determine the RPM. At the end of the procedure, CCF0 and CF flags are set back to 0, and all EA and EC are set back to 1. The overflow counter variable is also reset to 0.

Within the main function, the LCD displays the RPM of the motor inside the while loop.

**Part 5: Light Control System**

This exercise is to design a system with the following specifications:

1. Upon a significant drop in light intensity in the room, a lamp and a DC motor are switched on for 30 seconds.
2. On the LCD display, the time elapsed since the change in light occurred, and a count of how many holes of the motor disk have passed the optoswitch.
3. Display an indication of motor speed in RPM.

This system re-uses the PWM, the LDR and the RPM initialisation code. The logic of the system revolves around the idea to turn on a global flag whenever the LDR detects a drop in light intensity. The global flag is then turned off when the 30 seconds is up. The flag is only turned on when the external interrupt is turned on. This occurs in the LDRinterrupt() interrupt procedure with an address of 0.

The PCAinterrupt() procedure deals with the calculations of the number of holes that occur, the number of seconds that has elapsed and the RPM of the motor currently. It uses an address of 6. To calculate the time that has been elapsed, the RPM of the DC motor and the number of holes that has passed, the global flag variable must be true. To determine the current time, the number of overflows that has occurred since the light intensity drop can be calculated by dividing the overflow by 15.26. If the time was greater or equal than 30, the global flag is set to 0 and the overflow and holes variable is reset back to 0. To calculate the RPM is the same procedure outlined in part 4. The RPM depends on the number of overflows that has occurred between the adjacent interrupt events.

Within the while loop, what gets printed out depends on the current state of the global flag. If it is true, then it would print out the number of holes it has passed since the 30seconds countdown began, the current RPM of the DC motor, and the countdown itself. The CCAP0H register is set to 0xFF which corresponds to a 100% duty cycle. This means that the motor is at full speed within that 30 seconds. However, it is not moving when the global variable is false. So the PWM duty cycle is 0% when the value in the CCAP0H register is set to 0x00.