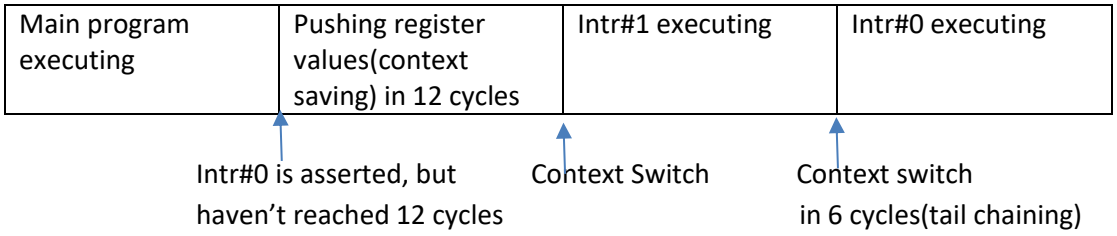
Solver: Andre Sutiono

Email Address: asutiono001@e.ntu.edu.sg

1. (a)
   (i) Privilege access level provides access to hardware resources and restricted memory region. It is used upon reset in order to initialize or set up the hardware properly. On the other hand, user access level is used when user application is executing. It is used to prevent misuse by users or bugs in the code which may access restricted memory region.
   (ii) Handler mode always uses main stack, but Thread mode can use either main stack or process stack. The main stack is usually used to perform OS related tasks. The process stack is used to perform specific user process. This separation technique is a good practice to isolate OS related task from user task.
   (b)



2. (a)
   (i) Volatile is useful when a variable's value is read repeatedly and an ISR is used to modify the variable's value. In that case, the compiler will optimize the code such that the variable's value from the first reading is put into register and each subsequent reading take it as the value. This is obviously not what we want. Volatile prevents optimization from the compiler and force the program to re-read the variable at each use.
   (ii)

```
volatile Int16_t valueA;
:
:
for(int8_t i = 5; i>0; i--)_{
   if(valueA>4000)
         valueA =4000;
   else
         valueA *=4;
}
```

   Assumption:
   - The variable valueA may be changed by ISR at any time

   Optimization:

- Add volatile keyword so that the program is forced to reread the valueA every time reading is needed
- Loop counting down is used so that there is no register space needed to store the for loop terminating condition
- *= is used to reduce code size

(b)

```
int statePrint, stateScan, stateFax, eventPrint, eventScan,
eventFax;
int printIdle, printPrinting;
int scanIdle, scanScanning;
int faxIdle, faxSending;
int eventPrintJobComes, eventPrintCancel, printDone;
int eventScanJobComes, eventScanCancel, scanDone;
int eventFaxJobComes, eventFaxCancel, faxDone;

void print(void){
      switch(statePrint){
            case(printIdle){
                  if(eventPrint==eventPrintJobComes}
                        statePrint=printPrinting;
                  break;
                  if(eventPrint==eventPrintCancel)
                        statePrint=printIdle;
            }
            case(printPrinting){
                  //print letters to the paper
                  if(eventPrint==eventPrintCancel){
                        //cancel the printing process
                        statePrint=printIdle;
                  }
                  if(eventPrint==printDone)
                        statePrint=printIdle;
                  break;
            }
      }
void scan(void){
      switch(stateScan){
            case(scanIdle){
                  if(eventScan==eventScanJobComes}
                        stateScan=scanScanning;
                  break;
                  if(eventScan==eventScanCancel)
                        stateScan=scanIdle;
            }
            case(scanScanning){
                  //scan paper
```

```
                        if(eventScan==eventScanCancel){
                              //cancel the scanning process
                              stateScan=scanIdle;
                        }
                        if(eventScan==scanDone)
                              stateScan=scanIdle;
                        break;
                  }
            }
      void fax(void){
            switch(stateFax){
                  case(FaxIdle){
                        if(eventFax==eventFaxJobComes}
                              stateFax=faxSending;
                        break;
                        if(eventFax==eventFaxCancel)
                              stateFax=faxIdle;
                  }
                  case(faxSending){
                        //fax the letter
                        if(eventFax==eventFaxCancel){
                              //cancel the faxing process
                              stateFax=faxIdle;
                        }
                        if(eventFax==faxDone)
                              stateFax=faxIdle;
                        break;
                  }
            }
      }

      void main(void){
            statePrint = printIdle;
            stateScan = scanIdle;
            stateFax = faxIdle;

            while(1){
                  print();
                  scan();
                  fax();
            }
      }
```

3. (a)
   (i) Code execution:
   - Task1 will be executed first
   - The while loop is entered

- The while loop exits when the button is pressed and set tx_msg to 1
- Task1 posts the message to Task2 and delays itself (task switching)
- Task2 takes the message from Task1 and check the value of it and if it is more than 1, it executes the function button_pressed()
- Task2 executes the master while loop until the Task1's timer timeout

(ii) 1 way to improve efficiency is to use OSSemPend at Task1 to wait for button press instead of polling it continuously by calling button_pressed() and use ISR to increase post semaphore value o Task1.

(iii)
```
void Task1(void){

      int tx_msg;

      while(1){

            OSSemPend(button);

            tx_msg = 1;

            OSTaskQPost( &Task2, tx_msg);

            OSTimeDlyHMSM(0,0,1,0);

      }

}

void isr(void){

      if button is pressed{

            OSSemPost(button);

      }

}
```

(b)

(i) Drive Strength register is used to set the amount of current output from the GPIO pin to represent logic 1. Device which is connected to the output pin of the microcontroller may have different standard for logic 1 and so, the register is used to set the current so that the device can read logic 1 and 0 properly.

(ii) Pull-Up resistor is used to set logic 1 to GPIO pin when the pin is disconnected.

(c) Watchdog timer is a hardware counter that counts to zero at a fixed rate. If the timer timeouts to zero, the watchdog timer will send a reset signal to the CPU. It is useful when the system is hanged and therefore needs to be reset. It may not be able to capture a system deadlock because in the event of deadlock, the watchdog timer reset ISR can still execute.

(d)

(i) A watchdog timer can be used. In critical application, watchdog timer can be set to do safety critical actions before sending reset signal to the CPU.
(ii) The system can be designed such that it saves current state periodically to a secondary memory.

4. (a)
   (i) When the count value is >= 50, the sensor_value is set into count*3, but if input_isr modifies the count value to less than 50 shortly after that, the sensor_value will be wrong.

(ii)
```
int i = 0;
void main(void)

    {

        while(1){

            if(count >= 50)

            {

                sensor_value = count*3;

                i = 0;

            }

    }

    void input_isr(void)

    {

    if(i == 0){

        if(count>=50){

            count = port1.0;

            i = 1;

        }

        else

            count = port1.0;

    }

    }
```

(b)

Task1(priority1)

		Resource Owned by Task1
Running	Pending(resource owned by Task3)	Running

Task2(priority2)

	Priority Inversion	
Ready	Running	Pending

Task3(priority3)

		Resource Released
Pending(resource owned by Task3)	Running	Ready

Unbounded Priority Inversion occurs when a third (medium-priority) thread preempts the low-priority thread during the inversion, thus delaying the high-priority thread even more. It is called “unbounded” because it will persist as long as the medium-priority thread has all the resources it needs to continue running, which is unrelated to the resource shared by the other two threads and can be unpredictable.

(c)

(i)

```
Void TaskA(void){
      double temp = read_temperature();
      OSTaskQPost(&TaskB, temp);
      OSTimeDlyHMSM(0,0,5,0);
}
Void TaskB(void){
      double temp = OSTaskQPend();
      send_zigbee(temp);
}
```

(ii) Run Mode Clock Gating Register, Sleep Mode Clock Gating Register and Deep Sleep Mode Clock Gating Register can be used to control the clocking of individual peripherals depending on the power mode of the microcontroller. If the clock of peripherals is slowed down or turned off, the peripheral will consume less power.

For reporting of errors and errata, please visit pypdiscuss.appspot.com
Thank you and all the best for your exams! ☺