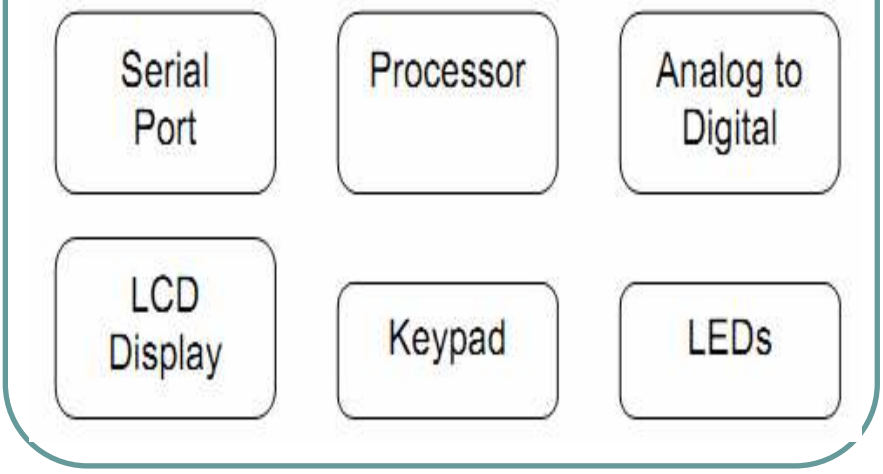
ARM PROGRAMMING

Bùi Quốc Bảo

Example of a data logger



Beginning of the code

No RTOS

```
main()
{
Inialize();
   while (1)
   {
      // no code yet
   }
}
```

Using FreeRTOS

```
main()
{
Inialize();
   while (1)
   {
      // no code yet
   }
}
```

Add serial port functionality

- MAIN MENU
- ---------
- 1. Read A/D
- 2. Write GPIO outputs
- 3. Read GPIO inputs
- 4. Set LEDs
- 5. Write string to LCD display

Add serial port functionality

```
void ProcessSerialPort()
{
  char s[80];
  while (1) {
    char c = getchar();
    switch c
    {
      case 1: // Read and display ADC value
            break;

      case 2: // Prompt for value and write to GPIO outputs
            printf("Enter GPIO value: ");
            fgets(s, 80, stdin);  // will block until user hits <enter>
            break;
```

Add serial port functionality

```
case 3: // Read and display GPIO input values
            break;
      case 4: // Prompt for value and write to LEDs
            printf("Enter LED value: ");
            fgets(s, 80, stdin);  // will block until user hits <enter>
            break;

      case 5: // Prompt for value and write to LCD display
            printf("Enter display string: ");
            fgets(s, 80, stdin);  // will block until user hits <enter>
            break;

      default: // print error message
    }
  }
}
```

Non-RTOS

```
void ProcessSerialPort()
{
  // serial port code goes here
}


main()
{
  while (1)
  {
    if ( SerialCharAvail() )
      ProcessSerialPort();
  }
}
```

Using FreeRTOS

```
void ProcessSerialPort(void *pvParameters )
{
  // serial port code goes here
}
void Main(void)
{
 xTaskCreate(ProcessSerialPort, "serial Task", 1000, NULL,
   serialPri, NULL );
  while (1)
  {
    // no code yet
  }
}
```

Compare

- Khi không dùng RTOS, trong vòng lặp chính phải có lệnh kiểm tra serial port.
- Chương trình sẽ bị "block" khi chờ người dùng nhập dữ liệu từ serial port.
- Với RTOS, ta không cần thêm code để kiểm tra serial port.
- Tác vụ `ProcessSerialPort` sẽ bị block khi chờ người dùng nhập dữ liệu, nhưng chương trình chính sẽ không bị block

Add Keypad Functionality

- Keypad trong trường hợp này là 1 ma trận phím.
- Khi phím được nhấn, 1 ngắt được tạo ra.
- Khi không dùng RTOS, 1 flag sẽ được set trong ISR.
- Với FreeRTOS, ISR được dùng để "give" semaphore cho keypad

Non-RTOS

```
void ProcessKeypad()
{
   char key = ScanKeypad();
   // code to process key goes here
}
```

Free-RTOS

```
void ProcessKeypad(void *pvParameters )
{
  while (1)
  {
     xSemaphoreTake( xKeyPadSemaphore,
                                portMAX_DELAY );
    char key = ScanKeypad();
    // code to process key goes here
   }
}
```

Non-FreeRTOS

```
main()
{
  while (1)
  {
    if ( SerialCharAvail() )
      ProcessSerialPort();

    if ( Keypressed )
      ProcessKeypad();

  }
}
```

```
Void main(void)
{
 xTaskCreate(ProcessSerialPort, "serial Task", 1000,
   NULL, serialPri, NULL );
xTaskCreate(ProcessKeypad, "keypad Task", 1000,
   NULL, keyPadPri, NULL );
  while (1)
  {
    // no code yet
  }
}
```

- Khi không dùng RTOS, nếu chương trình đang chờ serial port, các phím nhấn trên keypad sẽ không được xử lý
- Nếu dùng ISR cho keypad để xử lý phím, ISR sẽ chạy trong thời gian dài. Điều này làm giảm tốc độ đáp ứng của hệ thống.

- Với FreeRTOS, nếu độ ưu tiên của ProcessKeypad cao hơn ProcessSerialPort, chương trình sẽ chuyển sang ProcessKeypad ngay khi có phím nhấn, xử lý và quay trở lại ProcessSerialPort.

Adding ADC function

- Khi ADC có dữ liệu, một ngắt sẽ được tạo ra. ISR sẽ đọc giá trị ADC vào 1 biến toàn cục.
- Khi không dùng RTOS, ISR sẽ set 1 cờ.
- Khi dùng với freeRTOS, ISR sẽ “give” một semaphore.
- Sau khi giá trị ADC được đọc, nó sẽ được xử lý và lưu vào 1 mảng.

Non-RTOS

```
void ProcessADC()
{
   if ( SampleReady )
   // code to read and store ADC value
}
```

Non-FreeRTOS

```
main()
{
  while (1)
  {
    if ( SerialCharAvail() )
      ProcessSerialPort();

    if ( Keypressed )
      ProcessKeypad();

    if ( SampleReady )
      ProcessADC();

  }
}
```

- Dữ liệu có thể bị mất nếu chương trình chưa gọi ProcessADC() trước khi có dữ liệu tiếp theo (vì phải chờ serialport, …).
- Dữ liệu có thể sai nếu chương trình chính và ngắt cùng truy cập biến toàn cục lưu giá trị của ADC cùng lúc.

With FreeRTOS

```
void ProcessADC(void *pvParameters)
{
while (1)
  {
   xSemaphoreTake( xADCSemaphore,
                                   portMAX_DELAY
   );
     // code to read and store ADC value
   }
}
```

With FreeRTOS

```
Void main(void)
{
 xTaskCreate(ProcessSerialPort, "serial Task", 1000, NULL,
    serialPri, NULL );
xTaskCreate(ProcessKeypad, "keypad Task", 1000, NULL,
    keyPadPri, NULL );
xTaskCreate(ProcessADC, "ADC Task", 1000, NULL, ADCPri,
    NULL );

  while (1)
  {
    // no code yet
  }
}
```

- Để đảm bảo ADC luôn được xử lý trước, độ ưu tiên của tác vụ ADC được set cao hơn các tác vụ còn lại.
- Thay vì ghi giá trị vào 1 biến, ISR có thể ghi giá trị ADC vào 1 queue.

Conclusion

- Nếu không dùng RTOS, programmer phải thiết kế một scheduler để quản lý các sự kiện.
- Việc dùng FreeRTOS để thiết kế chương trình sẽ làm cho việc thiết kế dễ dàng hơn.
- FreeRTOS đã được debug và đảm bảo bug-free, do đó sử dụng FreeRTOS sẽ làm cho chương trình ổn định hơn