



Lab 5

Tuesday 2:15

Henry Fang Jack Landers

Introduction

In this lab we will build and test FIFO data buffers using timer and interrupt driven activity.

Procedure

Describe what you did during the lab. The way you wired up the board, what code you wrote (don't paste your actual code here), etc...

Step 1

Configure hardware settings in the CubeMX, we want a 20MHz CPU clock. And a timer TIM3 interrupts every 10 ms. And a timer TIM6 interrupts every 48 ms. And the center joystick to interrupt when pressed.

Step 2

Adding an integer variable Count, an integer array variable Buffer[64], and an integer array variable Snapshot[64].

Project 1

Implement a circular buffer queue with Buffer[]. We chose to use a flag instead of sacrificing an array index. For input interrupt, we set TIM3 to input value of Count every 10 ms if the queue is not full, and increment the Count. For output interrupt, we set TIM6 to remove at most 4 elements from the queue every 48 ms. We use the joystick to snapshot the buffer from head to tail, the empty spots in the snapshot will be filled with 0xfeedbeef.

Project 2

Make a copy of Project 2, we automated the snapshot process in the main while true loop that happens every 1 s. The code is supposed to not work correctly, without any other modifications. We will add

Results

Place the results of your experiments and tasks here. Tables, charts, screenshots, etc..
Answer any questions from the lab document here.

Project 1

Results:

Address:	snapshot
>x20000210:	00000055 00000056 FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF
>x20000240:	FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF
>x20000270:	FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF
>x200002A0:	FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF
>x200002D0:	FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

Snapshot after approx 1s

```
s: snapshot
00210: 0000008D 0000008E 0000008F 00000090 00000091 00000092 00000093 00000094 00000095 00000096 00000097 00000098
00240: 00000099 0000009A 0000009B FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF
00270: FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF
002A0: FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF
002D0: FEEDBEEF FEEDBEEF FEEDBEEF FEEDBEEF 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

Snapshot after approx 3s

```
00000175 00000176 00000177 00000178 00000179 0000017A 0000017B 0000017C 0000017D 0000017E 0000017F 00000180
00000181 00000182 00000183 00000184 00000185 00000186 00000187 00000188 00000189 0000018A 0000018B 0000018C
0000018D 0000018E 0000018F 00000190 00000191 00000192 00000193 00000194 00000195 00000196 00000197 00000198
00000199 0000019A 0000019B 0000019C 0000019D 0000019E 0000019F 000001A0 000001A1 000001A2 000001A3 000001A4
000001A5 000001A6 FEEDBEEF FEEDBEEF 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

Snapshot after approx 5s

All the numerical values are sequential, the data is uncorrupted. And empty slots are filled with 0xfeedbeef.

Project 2

In the main while true loop, we always disable the IRQ first before we start to snapshot the buffer. So while we are snapshotting, the buffer is not changing and the int counter is not incrementing. So our data is uncorrupted for sure. After the snapshotting, we enable the IRQ again to let the program keep running. Beside those, the adding and removing from buffer work exactly the same way as project 1.

Conclusion

Short paragraph talking about the takeaways from this lab.

Appendix

Place code and other data here

Project 1

```

void EXTI0_IRQHandler(void)
{
    /* USER CODE BEGIN EXTI0_IRQn 0 */

    /* USER CODE END EXTI0_IRQn 0 */
    HAL_GPIO_EXTI_IRQHandler(PINA0_Pin);
    /* USER CODE BEGIN EXTI0_IRQn 1 */
    int *ptr;
    int i = 0;
    ptr = tail;
    while(ptr!=head){
        snapshot[i] = *ptr;
        if(ptr == &buffer[63]){
            ptr = &buffer[0];
        }
        else{
            ptr++;
        }
        i++;
    }
    while(i!=64){
        snapshot[i] = 0xfeedbeef;
        i++;
    }

    /* USER CODE END EXTI0_IRQn 1 */
}

```

Center Joystick Snapshot

```

void TIM3_IRQHandler(void)
{
    /* USER CODE BEGIN TIM3_IRQn 0 */

    /* USER CODE END TIM3_IRQn 0 */
    HAL_TIM_IRQHandler(&htim3);
    /* USER CODE BEGIN TIM3_IRQn 1 */

    if(flag != 1){ //if buffer is not full
        *head = count;
        count++; //increment count
        if(head == &buffer[63]){ //increment head
            head = &buffer[0];
        }
        else
        {
            head++;
        }

        if(head == tail){
            flag = 1;
        }
    }

    /* USER CODE END TIM3_IRQn 1 */
}

```

Timer 3 Buffer Count

```
void TIM6_DAC_IRQHandler(void)
{
    /* USER CODE BEGIN TIM6_DAC_IRQHandler 0 */

    /* USER CODE END TIM6_DAC_IRQHandler 0 */
    HAL_TIM_IRQHandler(&htim6);
    /* USER CODE BEGIN TIM6_DAC_IRQHandler 1 */

    int elements;

    if(head > tail){
        elements = head - tail;
    }
    else if(head < tail){
        elements = 64 + (head - tail);
    }
    else
    {
        if(flag==0)
        {
            elements=0;
            return;
        }
        else
        {
            elements=64;
        }
    }

    if(elements>=4){
        for(int i=1;i<5;i++){
            if(tail == &buffer[63]){
                tail = buffer;
            }
            else{
                tail++;
            }
        }
        if(flag==1){
            flag = 0;
        }
    }
    else{
        tail = head;
    }

    /* USER CODE END TIM6_DAC_IRQHandler 1 */
}
```

Timer 6 Buffer Removal

Project 2

```
main.c  stm32l4xx_it.c  startup_stm32l476xx.s  stm32l4xx_hal.c

114  /* USER CODE BEGIN WHILE */
115
116  while (1)
117  {
118      HAL_Delay(1000);
119      __disable_irq();
120      int *ptr;
121      int i = 0;
122      ptr = tail;
123      while(ptr!=head){
124          snapshot[i] = *ptr;
125          if(ptr == &buffer[63]){
126              ptr = &buffer[0];
127          }
128          else{
129              ptr++;
130          }
131          i++;
132      }
133      if(ptr==head && flag==1)//fix up case when queue is full
134      {
135          for(i=0;i<64;i++)
136          {
137              snapshot[i] = *ptr;
138              if(ptr == &buffer[63]){
139                  ptr = &buffer[0];
140              }
141              else{
142                  ptr++;
143              }
144          }
145      }
146      while(i!=64){
147          snapshot[i] = 0xfeedbeef;
148          i++;
149      }
150
151      __enable_irq();
152  }
```

Snapshot in main

```

void TIM6_DAC_IRQHandler(void)
{
    /* USER CODE BEGIN TIM6_DAC_IRQn 0 */

    /* USER CODE END TIM6_DAC_IRQn 0 */
    HAL_TIM_IRQHandler(&htim6);
    /* USER CODE BEGIN TIM6_DAC_IRQn 1 */
    int elements;
    if(head > tail){
        elements = (head - tail);
    }
    else if(head < tail){
        elements = 64 + (head - tail);
    }
    else
    {
        if(flag==0)
        {
            elements=0;
            return;
        }
        else
        {
            elements=64;
        }
    }

    if(elements>=4){
        for(int i=0;i<4;i++){
            if(tail == &buffer[63]){
                tail = buffer;

            }
            else{
                tail++;
            }
            elements --;
        }
        if(flag==1){
            flag = 0;
        }
    }
    else{
        tail = head;
        elements = 0;
    }
    /* USER CODE END TIM6_DAC_IRQn 1 */
}

```

Timer 6 Buffer Removal

```

void TIM3_IRQHandler(void)
{
    /* USER CODE BEGIN TIM3_IRQn 0 */

    /* USER CODE END TIM3_IRQn 0 */
    HAL_TIM_IRQHandler(&htim3);
    /* USER CODE BEGIN TIM3_IRQn 1 */
    if(flag != 1){ //if buffer is not full
        *head = count;
        count++; //increment count
        if(head == &buffer[63]){ //increment head
            head = &buffer[0];
        }
        else
        {
            head++;
        }

        if(head == tail){
            flag = 1;
        }
    }
    /* USER CODE END TIM3_IRQn 1 */
}

```

Timer 3 Buffer Count