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

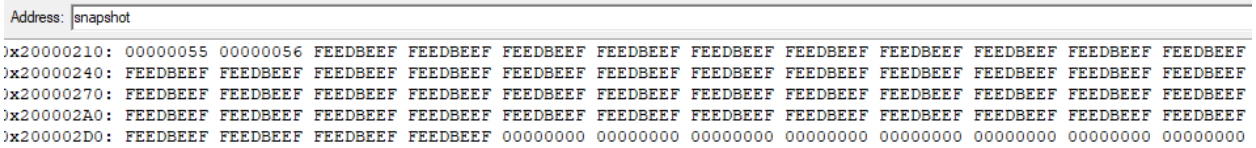
Make a copy of Project 2, we automated the snapshot process in the main while true loop that happens every 1 s. The code is supposed to not work correctly, without any other modifications. We will add

Results

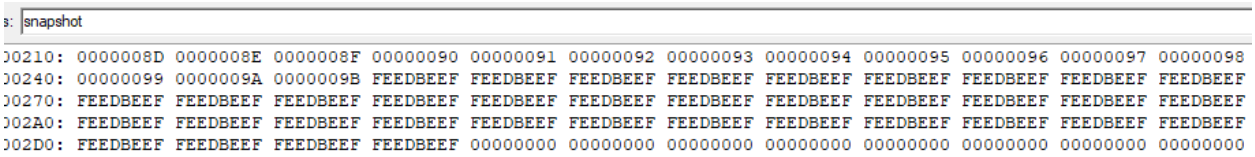
Place the results of your experiments and tasks here. Tables, charts, screenshots, etc.. Answer any questions from the lab document here.

Project 1

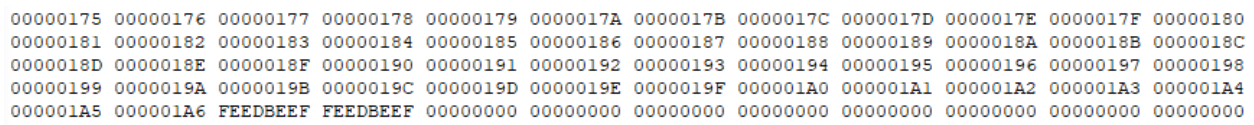
Results:



Snapshot after approx 1s



Snapshot after approx 3s



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 */
```

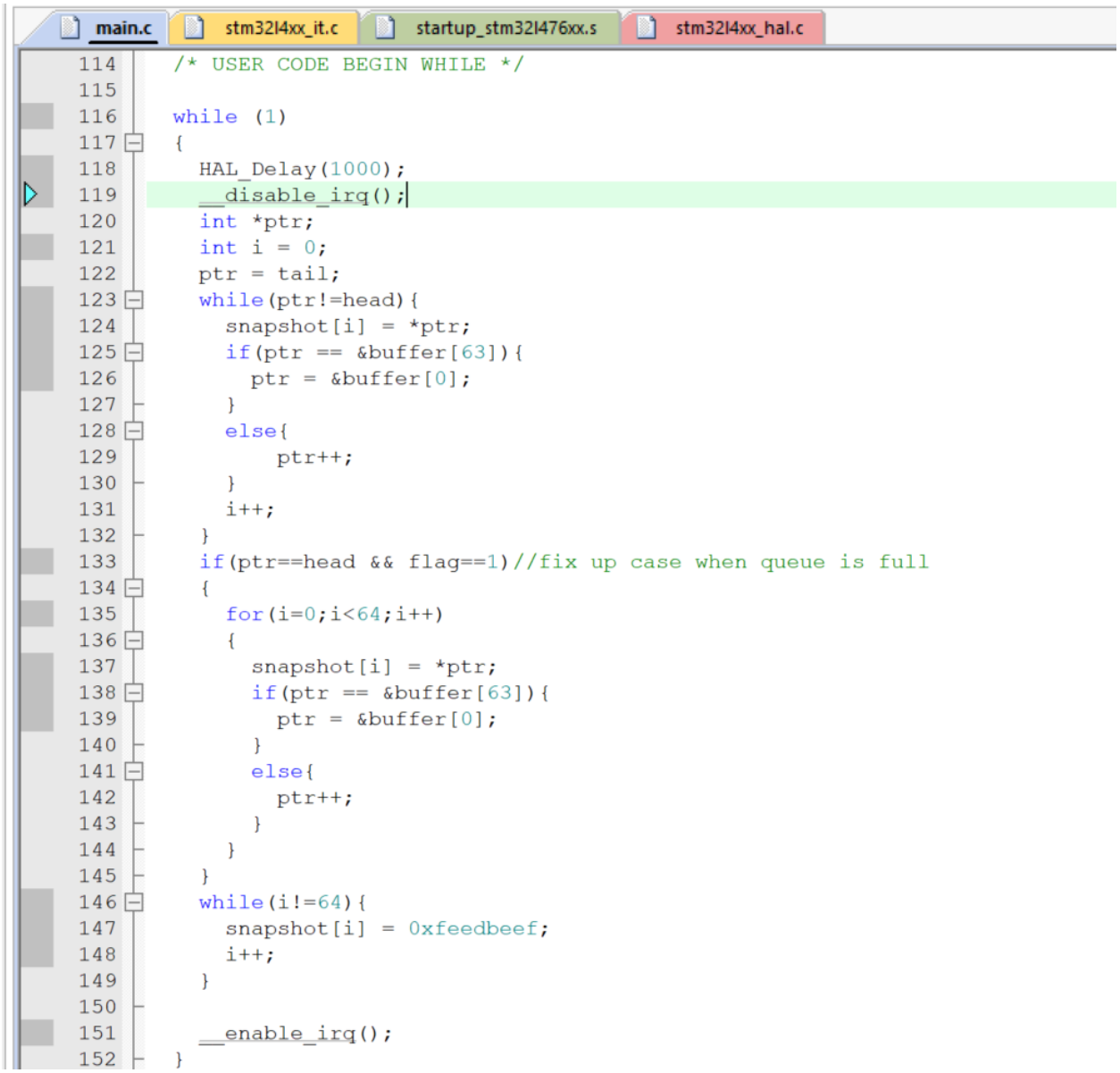
Center Joystick Snapshot

```
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++;//increament 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_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=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_IRQn 1 */
```

Timer 6 Buffer Removal



Snapshot in main

```c
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

```c
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++;//increament 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