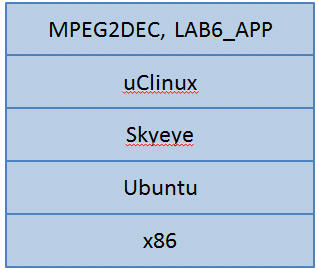
**Embeded system**  Dept of Computer Science Education

**Final project : : MPEG2 decoder with a device driver**

**2009190711 배주현 2009190714 김기훈**

1. **Introduction** lab7를 통해 uClinux 위에서 MPEG2 Decoder를 실행해 보았다. Encoding 된 MPEG파일을 Decoding 하는 과정 중 마지막쪽에 있는 IDCT (Inverse Discrete Cosine Transform) 는 가장 많이 시간을 소모하는 작업이라는 것을 알 수 있었다. 빠른 처리 속도를 보장해줘야 하는 동영상이기에 비용을 투자해 추가적인 하드웨어를 설치하고 이에 맞는 device driver를 구현해 줌으로써, Real time property를 높이고자 한다.
2. **Before project   
   1) Project 환경** Intel x86 컴퓨터에 Ubuntu OS를 설치하고 Atmel 사의 System on chip(soc) At91를 모델링한 Skyeye 위에서 프로젝트를 실시하였다.   
   **2) Lab6의 반복** Lab6에선 I/O Multiplexer Device addtion and Device drive로, Skyeye 소스코드 수정을 통해 multiplier하드웨어를 추가 했고, device driver를 uClinux 내부에 설치하여, uClinux안에서 Application이 실행 될 수 있도록 하였다. 여기서 숫자 ‘3’과 ‘4’를 곱하는 과정을 보면, Lab6\_app.c는 device drive 의 write(address, data) 함수를 통해 buf[10] 에 넣어놓은 ‘3’과 ‘4’를 실제 하드웨어Skyeye로 데이터를 보내고 Skyeye Multiplexer hardware는 2개의 input이 들어온 순간, 데이터를 연산하게 하였다. 연산결과는 바로 output 레지스터에 들어간다.  
    Lab6와 같이 Final project에서도 Skyeye에 이번엔 multiplexer가 아닌 IDCT Application을 설치, multiplexer device driver 대신 Idct\_driver를 설치하면 되는 것으로 판단. 하나하나 작업을 시행 하였다.
3. **What have you done?** 시작하기에 앞서 새로운 Peripheral Device IDCT의 MEMORY ADDRESS BASE는   
   0xffff\_c000 로 지정했다. 0xffff\_c000부터 2바이트 씩 64개의 레지스터를 마련했고, 장치에서 사용할 연산에 필요한 9개의 temp레지스터도 만들었다. 마지막으로 status 레지스터도 추가해 놓았다.
4. **Skyeye Code 수정**skyeye\_mach\_at91.c를 수정하여 새로운 Peripheral Device IDCT 장치를 설치 하였다. 즉 이 장치는 MPEG2DEC/Idct.c에 있는 idct.c 단순 반복 계산 과정을 데이터가 주어졌을 때 반복하는 장치이다.Multiplexer와 마찬가지로 struct 설정, read 기능, write기능을 구현하였다.

|  |
| --- |
| // final project start  typedef struct at91\_idct\_s{  short blk[64];  int x0, x1, x2, x3, x4, x5, x6, x7, x8;  uint32\_t status;  } at91\_idct\_t;  // final project end |
| static uint32\_t idct\_read(uint32\_t address) {  uint32\_t data;  switch (address & 0x00000fff) {  case 0x0:  data = io.idct.blk[0];  printf("Final: msg from h/w: idct blk0 read = 0x%08x\n", data);  break;  case 0x2:  data = io.idct.blk[1];  printf("Final: msg from h/w: idct blk1 read = 0x%08x\n", data);  break;  …………  case 0x80:  data = io.idct.status;  printf("Final: msg from h/w: idct status read = 0x%08x\n", data);  break;  default:  break;  }  return data;  } |
| static void idct\_write(uint32\_t address, short data) {  int i;  io.idct.status = 0;  switch (address & 0x00000fff) {  case 0x0:  io.idct.blk[0]=data;  printf("Final: msg from h/w: idct blk0 write = 0x%08x\n", data);  break;  case 0x2:  io.idct.blk[1] = data;  printf("Final: msg from h/w: idct blk1 write = 0x%08x\n", data);  break;  ………..  case 0x7e:  io.idct.blk[63] = data;  printf("Final: msg from h/w: idct blk63 write = 0x%08x\n", data);  for (i=0; i<8; i++)  idctrow(io.idct.blk+8\*i);  for (i=0; i<8; i++)  idctcol(io.idct.blk+i);  io.idct.status = 1;  break;  default:  printf("Final: msg from h/w: idct write out of bound\n");  break;  }  } |
| Idctrow(short \*blk) 추가  idctcol(short \*blk) 추가  위 두 함수는 기존에 mpeg2dec/idct.c에 있는 함수와 내용이 같다. |

1. **Device driver 생성( idct\_driver.c, Kconfig, Makefile )**Application과 Hardware를 소통해주는 device driver를 생성한다.  
   Base = 0Xffff\_c000 를 이용해 적절히 번지수를 조절해가며 Read와 Write를 사용한다.

|  |
| --- |
| static int idct\_driver\_open(struct inode \*inode, struct file \*filp); static int idct\_driver\_release(struct inode \*inode, struct file \*filp); static struct file\_operations idct\_driver\_fops  static int \_\_init idct\_driver\_init(void); **ssize\_t idct\_driver\_write (struct file \*filp, const short \*buf, size\_t length, loff\_t \*f\_pos)** {  short i;  // Base Address of Idct Device  unsigned int \* idct\_base = (unsigned int \*) 0xffffc000;  for(i=0; i<64 ; i++){  \*(idct\_base + i) = \*(buf+i); // input 64 operands }  return 0; } **ssize\_t idct\_driver\_read(struct file \*filp, short \*buf, size\_t \*status, size\_t count, loff\_t \*f\_pos)** {  short i;  // Base Address of IDCT Device  unsigned int \* idct\_base = (unsigned int \*) 0xffffc000;  short size\_short\_64 = 64;  for(i=0; i < 64; i++)  buf[i] = \*(idct\_base + i) ;   status[0]=\*( idct\_base)+sizeof(short)\*64;  return 0; } |
| **static struct file\_operations idct\_driver\_fops** = {  open : idct\_driver\_open,  read : idct\_driver\_read,  write : idct\_driver\_write,  release : idct\_driver\_release,  };  static int \_\_init idct\_driver\_init(void)  {  int result, ret;  result = register\_chrdev(IDCT\_DRIVER\_MAJOR, IDCT\_DRIVER\_NAME, &idct\_driver\_fops);  printk("%s : MAJOR# = %d\n", IDCT\_DRIVER\_NAME, IDCT\_DRIVER\_MAJOR);  return 0;  }  static void \_\_exit idct\_driver\_exit(void)  {  unregister\_chrdev(IDCT\_DRIVER\_MAJOR, IDCT\_DRIVER\_NAME);  printk("Release %s Module MAJOR %d\n", IDCT\_DRIVER\_NAME, IDCT\_DRIVER\_MAJOR);  } |
| Kconfig, Makefile은 lab6를 참조로하여 똑같이 만들었다. |

1. **MPEG2Decoder Application 수정과 Device driver 연결**idct.c 에 있던 idctrow() , idctcol() 기능을 부르는 함수가 같은 파일 내에 있는 fast\_idct 이고 이를 부르는 것이 getpic.c에 있었기 때문에, 그 부분에서 device driver를 불러 idct하드웨어에 접근하도록 하였다.

|  |
| --- |
| /\* ISO/IEC 13818-2 section Annex A: inverse DCT \*/  if (Reference\_IDCT\_Flag)  Reference\_IDCT(ld->block[comp]);  else{  fd = open("/dev/idctdriver", O\_RDWR);  write(fd, ld->block, 64);  //do{  read(fd, ld->block, 64);  //}while(status[0]==0);  close(fd);  } |

1. **Makefile 및 Kconfig 수정, device driver 설치**Lab6에서 했던 작업을 그대로 따라 device driver를 설치하고 Makefile 수정을 통해 새로운 device driver를 uClinux make과정에 포함시키도록 하였다. Menuconfig를 통한 uClinux make를 통해 device driver와 수정된 mpeg2dec과정도 함께 포함시켰다. 새로운 skyeye 모델링을 make하였다.
2. **What were the issues performing the project?   
   - Peripheral Device와 Application 을 연결해주는 Device driver설치** 요즘은 그렇지 않지만 예전처럼 프린터를 이용하고 싶다면, 컴퓨터와 연결 하는 것 뿐만 아니라 CD-ROM을 통해 Driver를 설치 했던 것 처럼, 무작정 투자를 하여Peripheral Device를 추가적으로 설치한다고, 이를 바로 이용할 수 있거나 그런 것은 아니다. 컴퓨터 운영체제로 하여금 컴퓨터의 어떤 장치와 통신하여 그것을 제어하게 할 수 있는 프로그램, Device driver를 통해 우리는Peripheral Device를 이용할 수 있다.   
   **- Hardware와 Software의 co-work** 프로그램이 Realtime property를 보장해 주지 못하고 있는 상황이다. 그럴 경우 해결책에 대해 의문을 제기한다. 보다 Compact하고 세련된 프로그래밍을 통해 소프트웨어를 만들어 Performance를 높여 주는 방법 뿐만 아니라, 하드웨어가 소프트웨어가 해야 될 일을 이렇게Peripheral Device를 이용해 나누어 처리를 해줄 수 가 있다. 문제가 발생했을시에 하드웨어 소프트웨어 어느 한쪽만을 통한 해결이 아니라 함께 co-work함을 통해 상황을 타개할 수 있다는 것을 알 수 있었다.
3. **Screenshots of the project outcome** 