

PORTFOLIO

Cho Mingi, SW Engineer

Age

1997/11/21 (26세)

Email

ring9714@gmail.com

Phone

+82 10-6639-5650

About Me

안녕하세요, 조민기 입니다.

학부 시절 전자공학을 전공했으며 OS, 자료구조, 알고리즘 등의 흥미를 바탕으로 전자공학도로서 HW를 이해하는 SW 엔지니어의 꿈을 이루고 있습니다.

특히 ARM 기반 Firmware 개발 및 스케줄러 설계, SoC 기반 SW 개발 역량과 함께 전장 SW 엔지니어로서의 비전을 가지고 있습니다.

제가 가진 문제 해결 역량을 바탕으로 세상의 변화에 기여하고, 배움을 나누며 함께 성장하고 싶습니다.



Non-Volatile Memory Systems # Real - Time System #Arm Embedded Systems
System Software # Web Development # HW Design

취미 헬스, 바둑, 여행
공인 어학 2023.08.30 OPIC IH

학력 사항

EDUCATION

경북대학교 전자공학부 졸업(B.S, 2016.02 ~ 2022.02) **3.62 / 4.5**

- Undergraduate Research Intern at <https://ai-soc.github.io/> (2020.08 ~ 2021.03)
Paper : 폴링기반 통신 시스템을 위한 에너지 인지적인 동적 주파수 조절 알고리즘, 한국정보통신학회논문지 Vol. 26, No. 9: 1405~1411, Sep. 2022
 - Work Experience as Embedded SW Engineer (2020.12 ~ 2021.02) at IEETU(<http://www.eplatform.co.kr/company/>)
 - KNU HustarICT Embedded SW Engineer Track (2020.03 ~ 2022.02) <https://hustar-ict.knu.ac.kr/>
-

EXPERIENCE

대구경북과학기술원 전기전자컴퓨터학과(Integrated M.S/B.S, 2022.03 ~ 2023.06)

- I was member of <https://rtcl.dgist.ac.kr/>
- Teaching Experience : Algorithm (2022, Fall)

삼성 청년 SW 아카데미(2024.01 ~)

- Academic Excellence Award
- Acquired Samsung SW Certification

Skills & Tools

소프트웨어 능력	
소프트웨어 명	주요 사용 능력
Programming Language	C (firmware), Java (web)
Vivado	FPGA(Zynq) based SoC Firmware development
Linux	Linux System Programming, Driver development
STM32 CubeIDE, S32 Design Studio	MCU firmware Development (STM32, NXP board)
Spring Tool Suite	Web development (Frontend / Backend)

Project 경험	
프로젝트 주제	내용
ARM MCU firmware development	ARM기반 MCU상에서 드라이버 설계 및 싱글코어 기반 임베디드 OS 개발
Linux mq-block driver development	리눅스 멀티큐 구조에 대한 학습 및 커널 드라이버 개발
SSD flash translation layer design	낸드플래시 기반의 SSD가 가진 문제점에 대해 학습하고 매핑 테이블 사이즈 및 응답시간 및 지연율을 줄이기 위한 FTL 알고리즘 개발
FPGA based HW/SW co-design	Xilinx사의 Zynq FPGA를 활용한 이미지 필터링 가속기 설계(SW펌웨어 개발)

PROJECTS

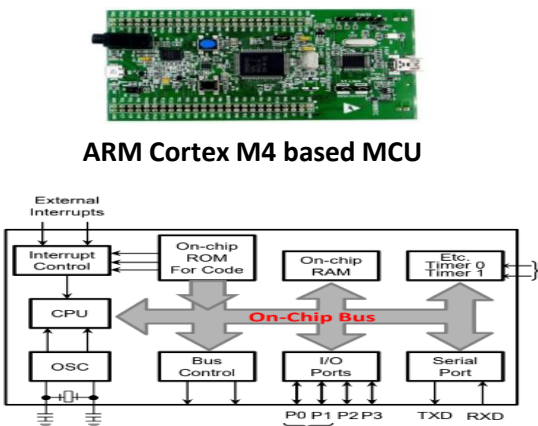
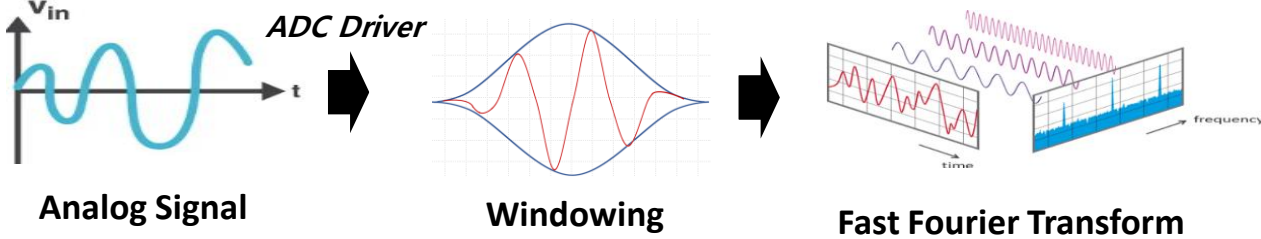
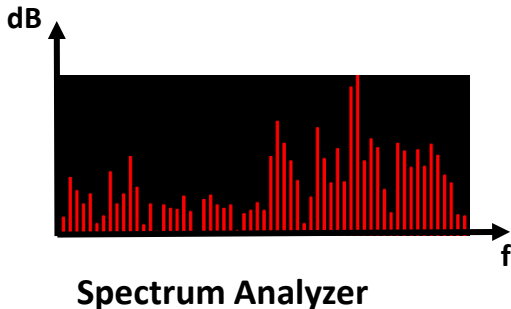
1. ARM based MCU Development

2. Flash Translation Layer Design

3. Linux multi-queue driver Development

4. Zynq Image Filtering Acceleration

1-1. ARM based MCU Development – Spectrum Analyzer

프로젝트 주제	ARM 기반 MCU 펌웨어 및 임베디드 OS 설계 (2021.03~ 2021.06)
개요	임베디드 시스템 및 ARM구조에 대해 학습한 내용을 바탕으로 실제 MCU보드의 부트로더 및 드라이버를 구현하고 ADC드라이버를 통해 입력되는 아날로그 음성신호에 대해 고속푸리에 변환(FFT)을 통해 주파수 영역대로 변환, MATLAB연동 실시간 시각화를 목표로 하였습니다.
내용	<div><div><div><div>1. HW Specification에 기반한 부팅프로세스 및 레지스터 매핑, 인터럽트 설정 등을 통한 드라이버 개발</div><div>2. ADC드라이버를 통해 입력되는 음성신호를 바탕으로 FFT연산(쿨리-튜키 알고리즘), 노이즈제거를 위한 Hamming Window 도입</div><div>3. UART기반 MATLAB연동 실시간 시각화(Tone Generator를 통한 주파수 분해 확인)</div></div><div><div><div>1. MCU Peripheral 설계</div><div><p>ARM Cortex M4 based MCU</p></div><div><div>2. FFT Processing</div><div><p>Analog Signal</p><p>ADC Driver</p><p>Windowing</p><p>Fast Fourier Transform</p></div><div><div>3. UART통신 기반 MATLAB 연동 실시간 시각화</div><div><p>Spectrum Analyzer</p></div></div></div></div></div></div></div>
고찰	<ul style="list-style-type: none">프로젝트 진행 시 임의의 주파수 영역대에 대한 출력이 되지 않는 오류를 파악했습니다. 원인분석 결과 펌웨어 설계 시 하드웨어 인터럽트 부분에서 데이터 수신 및 FFT 연산에 대한 처리를 모두 수행해 연산 수행 시간 동안 입력데이터에 대한 손실이 발생한다는 점을 파악하였습니다.이에 리눅스 커널의 인터럽트 후반부 처리방식 및 더블 버퍼링을 활용해 데이터 수신부와 실제 데이터 처리부를 구분하였고, 그 결과 입력 데이터 손실을 제거할 수 있었습니다.

1-2. ARM based MCU Development – Embedded RTOS

프로젝트 주제	ARM 기반 MCU 펌웨어 및 임베디드 OS 설계 (2021.09~ 2021.11)	
개요	싱글코어 기반 임베디드 시스템에서 멀티태스킹 구조의 필요성을 느끼고 직접 우선순위 기반 비선점 스케줄러를 구현하였습니다.	
소스코드	https://github.com/MinkiJo/MINTOS	
개발환경	Linux, QEMU(MCU 가상화), OPENOCD, MCU-STM32F4Discovery (ARM Cortex-M)	
내용	<div>1. 태스크 구조 작성 : 태스크의 스택 사이즈 설정 및 스택 base address 설정을 통한 정적할당, 태스크 상태(Sleep, Running) 변수 설정</div> <div>2. ARM Instruction, 함수호출규약을 통한 어셈블리 레벨 문맥교환 구현 :링커레지스터 및 스택포인터, 프로그램 카운터등의 레지스터 및 ARM의 r0~r12의 범용 레지스터들에 대한 Load 및 Save 함수 구현</div> <div>3. 우선순위 큐 설계 : Heap자료구조 기반 우선순위 큐 설계</div> <div>4. 함수 API사용 태스크 내부 yield호출을 통해 태스크가 직접 자원을 반납하는 비선점 스케줄링 구현</div>	<div>Stack base + Task Size</div> <div>4byte</div> <div>TaskContext</div> <div>SP</div> <div>Stack base</div>
고찰	<ul style="list-style-type: none">해당 구조를 통해 태스크의 상태 및 우선순위를 조정하여 응답성을 높일 수 있으며, 앞선 FFT 연산과 같은 인터럽트 후반부 처리에 대해 스케줄링 가능한 구조로 프로그래밍할 수 있음을 보였습니다.실시간 OS에 대해 학습하게 된 계기가 되었습니다. Deadline이 존재하는 hard real-time system의 경우 비선점 스케줄링 방식보다 timer 인터럽트를 통한 선점형 방식이 사용되고, 이에 따라 싱글코어 기반의 EDF 및 RM 등의 스케줄링 개념에 대해 이해했습니다. 더 나아가 해당 임베디드 OS를 사용해 I/O 워크로드가 많은 시스템에서 동적 주파수 조절 알고리즘을 설계하고, 한국정보통신학회에 게재하였습니다.	

2. Flash Translation Layer Design

프로젝트 주제	낸드플래시 기반 SSD에서 기존의 DFTL방식을 개선한 FTL 설계 (2021.06 ~ 2021.08, 2023.03 ~ 2023.06)																																																																				
개요	<p>SSD와 같은 저장장치에서 플래시의 WAE (Write After Erase) 특성으로 인해 논리주소와 물리주소의 매핑테이블 및 GC (Garbage Collection)동작과 같은 추가적인 오버헤드 발생함을 파악하였습니다.</p> <p>DFTL (Demand-based FTL) 알고리즘은 기존의 Page 단위 매핑방식을 채택하되 LRU방식을 통해 필요한 Entry (1 lpn-ppn mapping)에 대해 캐싱함으로써 메모리 사용량을 줄이지만, Cache Miss시 오버헤드가 크다는 단점이 존재, 이에 캐싱단위를 Translation Page단위로 바꾼다면 순차 접근에 대해 공간(spatial) 지역성을 활용할 수 있어 성능이 개선될 것으로 예측했습니다. (아래 그림 좌 - 기존 DFTL, 우 - 공간 지역성을 활용한 SFTL)</p>																																																																				
내용	<div><div><div><div>1. 시뮬레이터 선정 및 분석</div><div>2. DFTL 설계 : Entry-Level LRU 캐시 구현 및 캐시 miss 로직 설계, GTD (Global Translation Directory)설계 및 Translation Block설계</div><div>3. SFTL 설계 : Page-Level LRU 캐시 구현 및 캐시 miss 로직 설계</div><div>4. 캐시 Wrtite Back Policy 설계 및 GC (Garbage Collection) 알고리즘 선택(Greedy)</div></div><div><div><div><div><div>(1) LPN = 1025</div><div>Entry Mapping Table</div><table><tr><th>LPN</th><th>PPN</th></tr><tr><td>7</td><td>130</td></tr><tr><td>31</td><td>200</td></tr><tr><td>97</td><td>11</td></tr></table><div>victim</div></div><div><div>(2)</div><div>GTD</div><table><tr><th>VPN</th><th>PPN</th></tr><tr><td>0~511</td><td>21</td></tr><tr><td>512~1023</td><td>17</td></tr><tr><td>1024~2047</td><td>15</td></tr></table></div><div><div>DRAM</div><div>SSD</div><div><div>PPN = 111</div><table><tr><th>LPN</th><th>PPN</th></tr><tr><td colspan="2">Data</td></tr><tr><td colspan="2">OOB</td></tr></table></div><div><div>PPN = 15</div><table><tr><th>LPN</th><th>PPN</th></tr><tr><td>1024</td><td>110</td></tr><tr><td>1025</td><td>111</td></tr><tr><td>...</td><td>...</td></tr><tr><td>1535</td><td>560</td></tr></table></div><div>(3)</div><div>(4)</div></div></div></div><div><div><div>(1) LPN = 1025</div><div>Trans Page Mapping Table</div><table><tr><th>LPN</th><th>PPN</th></tr><tr><td>0~511</td><td>...</td></tr><tr><td>30720~...</td><td>...</td></tr><tr><td>184320~...</td><td>...</td></tr></table><div>victim</div></div><div><div>(2)</div><div>GTD</div><table><tr><th>LPN</th><th>PPN</th><th>V</th></tr><tr><td>0~511</td><td>21</td><td>O</td></tr><tr><td>512~1023</td><td>17</td><td>X</td></tr><tr><td>1024~2047</td><td>15</td><td>O</td></tr></table></div><div><div>Whole Trans Page</div><div>SSD</div><div><div>PPN = 111</div><table><tr><th>LPN</th><th>PPN</th></tr><tr><td colspan="2">Data</td></tr><tr><td colspan="2">OOB</td></tr></table></div><div><div>PPN = 15</div><table><tr><th>LPN</th><th>PPN</th></tr><tr><td>1024</td><td>110</td></tr><tr><td>1025</td><td>111</td></tr><tr><td>...</td><td>...</td></tr><tr><td>1535</td><td>560</td></tr></table></div><div>(3)</div><div>(4)</div><div>(5)</div></div></div></div></div></div>	LPN	PPN	7	130	31	200	97	11	VPN	PPN	0~511	21	512~1023	17	1024~2047	15	LPN	PPN	Data		OOB		LPN	PPN	1024	110	1025	111	1535	560	LPN	PPN	0~511	...	30720~...	...	184320~...	...	LPN	PPN	V	0~511	21	O	512~1023	17	X	1024~2047	15	O	LPN	PPN	Data		OOB		LPN	PPN	1024	110	1025	111	1535	560
LPN	PPN																																																																				
7	130																																																																				
31	200																																																																				
97	11																																																																				
VPN	PPN																																																																				
0~511	21																																																																				
512~1023	17																																																																				
1024~2047	15																																																																				
LPN	PPN																																																																				
Data																																																																					
OOB																																																																					
LPN	PPN																																																																				
1024	110																																																																				
1025	111																																																																				
...	...																																																																				
1535	560																																																																				
LPN	PPN																																																																				
0~511	...																																																																				
30720~...	...																																																																				
184320~...	...																																																																				
LPN	PPN	V																																																																			
0~511	21	O																																																																			
512~1023	17	X																																																																			
1024~2047	15	O																																																																			
LPN	PPN																																																																				
Data																																																																					
OOB																																																																					
LPN	PPN																																																																				
1024	110																																																																				
1025	111																																																																				
...	...																																																																				
1535	560																																																																				

2. Flash Translation Layer Design

프로젝트 주제	낸드플래시 기반 SSD에서 기존의 DFTL방식을 개선한 FTL 설계 (2021.06 ~ 2021.08, 2023.03 ~ 2023.06)																															
Simulator	FlashFTLDriver (https://github.com/dgist-datalab/FlashFTLDriver)																															
Configuration	Read:50us Write:500us Erase:2000us, SSD : 32GB, Page:4KB, Page Per Block : 512, LRU Cache Size : 32KB																															
Workload	MSR trace(trace-driven, https://trace.camelab.org/Citation.html)																															
결과	<div><div><p>Cache Hit Ratio (%)</p><table><tr><th>Workload</th><th>DFTL (%)</th><th>SFTL (%)</th></tr><tr><td>rsrch_0</td><td>63</td><td>83</td></tr><tr><td>usr_0</td><td>80</td><td>72</td></tr><tr><td>src2_1</td><td>70</td><td>90</td></tr><tr><td>proj_0</td><td>52</td><td>86</td></tr></table></div><div><p>Average Response Time</p><table><tr><th>Workload</th><th>DFTL</th><th>SFTL</th></tr><tr><td>rsrch_0</td><td>1.0</td><td>0.85</td></tr><tr><td>usr_0</td><td>1.0</td><td>0.95</td></tr><tr><td>src2_1</td><td>1.0</td><td>0.7</td></tr><tr><td>proj_0</td><td>1.0</td><td>0.8</td></tr></table></div><ul style="list-style-type: none">SFTL의 성능이 대부분 더좋은 성능을 보였으나, 경우에 따라 SFTL보다 DFTL의 hit ratio 성능이 높게 나온경우가 있었습니다. 이는 워크로드의 특성에 따라, 혹은 캐시가 워크로드의 지역성을 커버하지 못할때로 예상됩니다.Response Time의 경우 Hit ratio에 기반한 Flash 추가접근 감소에 따라 SFTL의 성능이 DFTL의 우위에 있었습니다. 하지만 write 워크로드에 따른 GC동작이 응답시간의 대부분을 차지하기 때문에 큰 성능의 개선은 없는경우가 존재했습니다. 반면 read 요청 비율이 많은 src2_1 워크로드에서는 30%이상의 응답시간 감소를 보였습니다.</div>		Workload	DFTL (%)	SFTL (%)	rsrch_0	63	83	usr_0	80	72	src2_1	70	90	proj_0	52	86	Workload	DFTL	SFTL	rsrch_0	1.0	0.85	usr_0	1.0	0.95	src2_1	1.0	0.7	proj_0	1.0	0.8
Workload	DFTL (%)	SFTL (%)																														
rsrch_0	63	83																														
usr_0	80	72																														
src2_1	70	90																														
proj_0	52	86																														
Workload	DFTL	SFTL																														
rsrch_0	1.0	0.85																														
usr_0	1.0	0.95																														
src2_1	1.0	0.7																														
proj_0	1.0	0.8																														
고찰	<ul style="list-style-type: none">구현에 있어 시뮬레이터에 대한 분석이 가장 시간이 많이 소모되었습니다. 대규모 소스코드를 분석하고 변형해 본 경험을 쌓았습니다. 또한 캐시 크기가 너무 커졌을 시 hit ratio가 약 99%에 도달하여 성능차이가 없음을 확인하고 적절한 크기로 선택했으며, 워크로드의 read/write 비율, locality에 따른 성능의 차이를 분석해 봄으로써 아키텍처에 대한 고찰을 다시 한번 할 수 있었습니다.해당 설계에서 또 하나의 최적화 방법으로 논리주소와 물리 주소의 선형 관계성을 활용해 메모리 사용량을 감소시켰습니다. 이에 추가적으로 linear regression 등의 머신러닝 기법을 활용해 메모리 사용을 최적화하는 연구로 활용될 수 있음을 확인하였습니다.																															

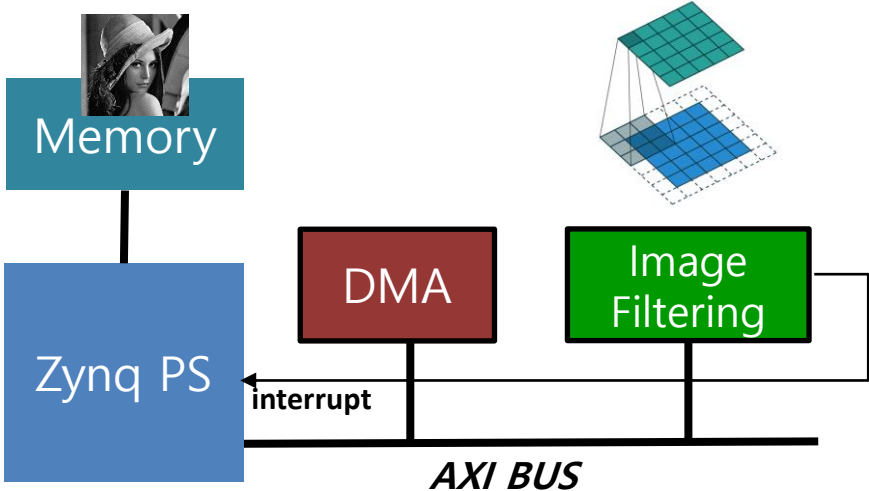
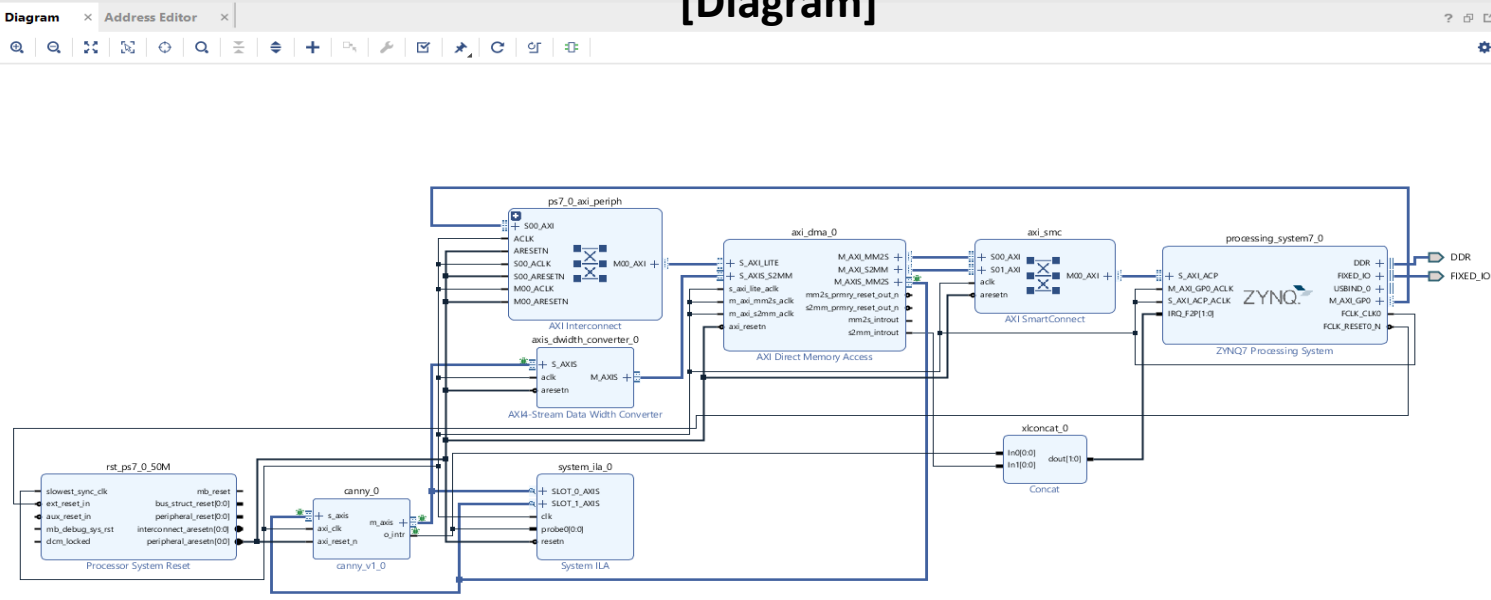
3. Linux multi-queue driver development

프로젝트 주제	리눅스 단일큐 기반 블록 드라이버와 멀티큐 기반 블록드라이버 성능비교 (2022.04 ~ 2022.06)
개요	NVMe SSD와 같은 빠른 저장장치 등장에 따라 기존의 커널구조가 가진 단일 큐 방식의 I/O가 lock등의 오버헤드를 발생시킴을 파악하고 이에 여러 큐를 사용해 동시에 여러 I/O 요청을 하여 병목현상을 줄이는 다중큐 기반의 드라이버를 설계하였습니다.
소스코드	https://github.com/MinkiJo/mqbrd
내용	<div><div><div><div><div>1. 개발환경 선정(Linux Kernel 4.4.1), 커널 빌드 및 설치</div><div>2. Skeleton 코드 작성및 request_fn 형태의 callback기반 I/O수행 확인, bio 및 reques단위 I/O 동작 파악</div><div>3. Null_blk 기반 싱글큐 드라이버 콜스택 분석 (Null_blk은 실제 리눅스 커널상의 연습용 드라이버임) 및 램드라이브 설계</div><div>4. 멀티큐 드라이버 콜스택 분석(SW-HW queue mapping)</div><div>5. Fio 벤치마크 기반 성능 비교</div></div><div><div><div><div><div>File System</div><div>submit_bio</div><div>make_request_fn</div><div>Block Layer(Plugging, Elevator..)</div><div>driver Init function map queue init..</div><div>mybrd_request_fn</div><div>device</div></div><div>Single Queue based</div></div></div><div><div><div><div><div>File System</div><div>submit_bio</div><div>blk_sq_make_request</div><div>sw queue</div><div>hw queue</div><div>blk_mq_run_hw_queue</div><div>init hctx blk_mq_tag_set</div><div>mybrd_queue_rq</div><div>mybrd_request_fn</div><div>device</div></div><div>Multi Queue based</div></div></div></div></div></div></div></div>

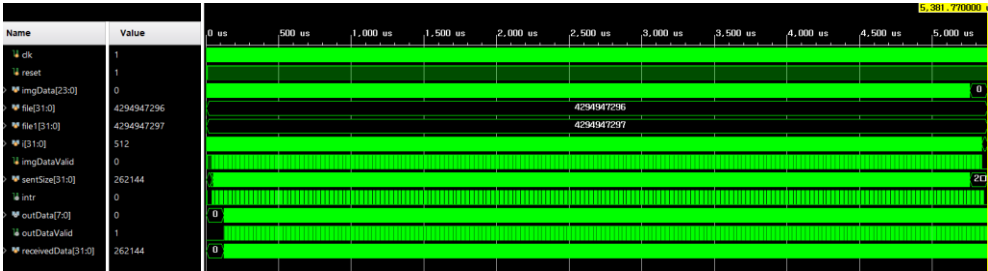
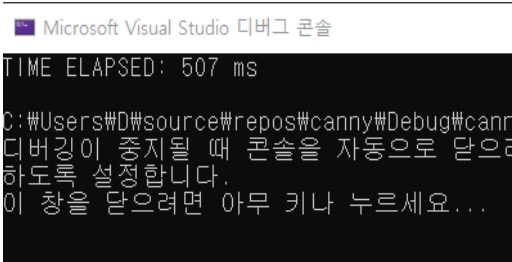

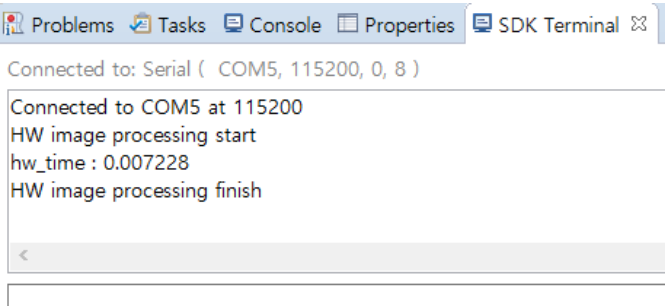
3. Linux multi-queue driver development

프로젝트 주제	리눅스 단일큐 기반 블록 드라이버와 멀티큐 기반 블록드라이버 성능비교 (2022.04 ~ 2022.06)													
개발환경	Linux Kernel 4.4, virtualbox ubuntu 16.04, CPU=4													
BenchMark	Flexible I/O Tester (FIO), 16GB libaio, direct=1, 512 job을 생성하여 멀티코어기반 시스템에서 성능을 비교													
결과	<div><pre>root@ubuntu2:/home/ming/sqbrd# fio --filename=/dev/mqbrd0 --direct=1 --ioengine=libaio --numjobs=512 --iodepth=32 --rw=randread --bs=4k --size=16G --runtime=60 --time_based --group_reporting --name=brdtest brdtest: (g=0): rw=randread, bs=4K-4K/4K-4K/4K-4K, ioengine=libaio, iodepth=32 ... fio-2.2.10 Starting 512 processes Jobs: 42 (f=0): [(429), E(1), r(1), E(1), r(1), E(2), (3), E(1), (2), r(1), (2), r(1), E(1), (1), r(1), (1), r(1), (1), r(1), E(4), r(2), (2), r(2), (1), r(1), E(3), (2), E(2), (1), E(1), r(4), (1), r(7), (1), r(2), E(2), r(1), E(2), r(2), (1), r(2), (1), r(5), E(1), r(5)] [6.8% done] [8779MB/0KB/0KB /s] [2247K/0/0 iops] [eta 14m:11s] brdtest: (group=0, jobs=512): err= 0: pld=6118: Tue Jun 18 02:10:01 2024 read : io=10104MB, bw=10104MB/s, lops=2580.0K, runt= 60731msec slat (usec): min=0, max=2878.3K, avg=147.62, stdev=6474.50 clat (usec): min=0, max=2878.4K, avg=6086.55, stdev=41137.72 lat (usec): min=1, max=2878.4K, avg=6238.38, stdev=41638.30 clat percentiles (usec): 1.00th=[55], 5.00th=[60], 10.00th=[63], 20.00th=[70], 30.00th=[81], 40.00th=[93], 50.00th=[97], 60.00th=[101], 70.00th=[105], 80.00th=[109], 90.00th=[116], 95.00th=[125], 99.00th=[259072], 99.50th=[276480], 99.90th=[399360], 99.95th=[473088], 99.99th=[749568] bw (Kb /s): min= 42, max=81228, per=0.20%, avg=20373.04, stdev=4263.38 lat (usec): 2=0.01%, 4=0.01%, 10=0.01%, 20=0.01%, 50=0.19% lat (msec): 100=56.94%, 250=40.49%, 500=0.03%, 750=0.01%, 1000=0.01% lat (msec): 2=0.01%, 4=0.01%, 10=0.02%, 20=0.02%, 50=0.03% lat (msec): 100=0.03%, 250=0.96%, 500=1.24%, 750=0.03%, 1000=0.01% lat (msec): 2000=0.01%, >=2000=0.01% cpu : usr=0.38%, sys=1.19%, ctx=123730, majf=0, minf=22711 IO depths : 1=0.1%, 2=0.1%, 4=0.1%, 8=0.1%, 16=0.1%, 32=100.0%, >=64=0.0% submit : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0% complete : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.1%, 64=0.0%, >=64=0.0% issued : total=157084050/w=0/d=0, short=r=0/w=0/d=0, drop=r=0/w=0/d=0 latency : target=0, window=0, percentile=100.00%, depth=32 Run status group 0 (all jobs): READ: io=10104MB, aggrb=10104MB/s, minb=10104MB/s, maxb=10104MB/s, mint=60731msec, maxt=60731msec Disk stats (read/write): mqbrd0: ios=156582162/0, merge=0/0, ticks=135708/0, in_queue=126224, util=87.70% root@ubuntu2:/home/ming/sqbrd# fio --filename=/dev/mqbrd0 --direct=1 --ioengine=libaio --numjobs=512 --iodepth=32 --rw=randread --bs=4k --size=16G --runtime=60 --time_based --group_reporting --name=brdtest</pre></div> <div><p>Fio 벤치마크 수행</p></div>	<div><p>Linux multi queue block driver</p></div> <table border="1"><thead><tr><th>Operation</th><th>single queue [IOPS / K]</th><th>multi queue [IOPS / K]</th></tr></thead><tbody><tr><td>Random Read</td><td>~150</td><td>~2600</td></tr><tr><td>Random Write</td><td>~150</td><td>~2500</td></tr><tr><td>Random R/W</td><td>~50</td><td>~1200</td></tr></tbody></table>	Operation	single queue [IOPS / K]	multi queue [IOPS / K]	Random Read	~150	~2600	Random Write	~150	~2500	Random R/W	~50	~1200
Operation	single queue [IOPS / K]	multi queue [IOPS / K]												
Random Read	~150	~2600												
Random Write	~150	~2500												
Random R/W	~50	~1200												

4. Zynq Image Filtering Accerelation

프로젝트 주제	Xilinx Zynq 기반 이미지 필터링 가속기 설계(2021.03 ~ 2021.06)
개요	Verilog 언어 기반 RTL수준으로 하드웨어상의 이미지 처리 가속 IP설계 후, 소프트웨어상에서 이미지 처리연산과 FPGA 디바이스 상의 하드웨어 연산을 통한 처리속도를 비교하였습니다.
팀원	4 (역할 : SW 상에서 이미지 연산속도 측정, DMA 및 Image Processing Interrupt Service 개발)
내용	<div><div><div><div>1. Verilog언어 활용 RTL레벨 Image Processing IP 설계 : Convolution 연산의 PipeLining구축</div><div>2. DMA 사용 디바이스 내 Memory-> IP (load) 및 FIFO를 통한 IP->Memory (Save)설계, Image Processing 및 DMA receive완료시 인터럽트 설정을 통해 PS에 통지방식 선정, 이후 IP Integration수행</div><div>3. SDK활용 Software Firmware를 통한 장치 초기화 및 이미지 데이터 전송 처리(My Main Task!)</div><div>4. 소프트웨어(C based)상 연산 수행속도와 Image processing 가속기를 활용한 연산 수행속도 비교</div></div><div><div><div>[Architecture]</div></div><div><div>[Diagram]</div></div></div></div></div>

4. Zynq Image Filtering Acceleration

프로젝트 주제	Xilinx Zynq 기반 이미지 필터링 가속기 설계(2021.03 ~ 2021.06)		
Target Board	Zybo Z7-20(Zynq-7000)		
Environment	Vivado 2019.1, Xilinx SDK		
Image / Filter	Lena / simple blur (https://en.wikipedia.org/wiki/Kernel_(image_processing))		
결과	<div><div>[DUT based]</div><div></div><div>Test Bench(50Mhz): 0.5s</div></div>		
	<div><div>[Device based]</div><div><div>SW(C based)를 이용한 처리 시간 : 0.5s</div><div></div></div><div><div>98.6 % 감소</div><div></div><div><div>HW를 이용한 처리 시간 : 0.007s</div><div></div></div></div></div>		
고찰	<ul style="list-style-type: none">하드웨어 상에서 512 X 512의 이미지 파일을 처리하기 위해 메모리 공간에 대한 최적화가 필요하였습니다. 이에 Convolution 특성 상 다음 행에 대한 연산을 수행하기 위해 상위 두 행이 재사용되고, 이에 4개의 line buffer만을 도입하여 이미지 처리 이후 인터럽트를 통해 다음 Line을 가져오는 방식으로 FPGA 상의 메모리를 획기적으로 줄일 수 있었습니다.프로젝트 진행 중 가장 어려웠던 부분은 HW configuration부분 이었습니다. 특히 DMA 설정과 관련해 output data width를 일치시키기 위해 convertor를 써야 했던 점, FPGA 상의 메모리를 전달하기 위한 FIFO 설정, cache coherence 보장을 위한 ACP Port 설정 등 하드웨어 아키텍처에 대한 고찰이 필요했습니다.팀 프로젝트로써 HW 설계와 SW 설계 인원이 나누어져 있었지만, 문제를 해결하기 위해 공통의 지식이 필요하였습니다. 팀장으로서 개발 내용들을 빠짐없이 작성하여 주석 및 설명하는 것을 제안했고, 문서화를 통한 소통을 이루어냈습니다.		

Thank You