Introducing Zynq-7000 EPP
The First Extensible Processing Platform Family

지난 봄 자일링스는 새로운 지금까지 나온 **FPGA** 와는 성격이 완전히 다른 새로운 제품을 출시 하였습니다.

자일링스는 이 제품의 이름을 "징크"라고 정했습니다.

징크라고 꼭 이름을 지은 이유를 알아보면 약간 철학적 의미도 있지만 여기서는 굳이 그 이름의 기원에 대해서는 굳이 설명하지는 않습니다.

"First Extensible Processing Platform Family"

굳이 번역하면 "첫번째 확장 가능한 프로세싱 플랫폼 패밀리" 이라고 할 수 있습니다.

사실 고객분들은 이런 비슷한 좋은 말을 하도 많이 들어서 그게 그거인 듯한 의미로 다가올 수 있습니다.

하지만 이 말은 저에게 다가오는 바는 적지 않습니다.

"확장 가능한" 이라는 말의 정확한 의미는 뒤에 좀더 자세히 나오겠지만 이런 말을 쓰게 된 배경에 대해서 잠시 얘기 하고자 합니다.

"확장 가능하다" 라는 것은 사실 설계 변경을 할 수 있다라는 것과 같은 의미로 받아들일 수 있습니다. 아니 설계 변경을 할 수 있다는 것으로는 좀 부족하고 "좀더 쉽게 (다른 말로 적은 비용으로) 좀더 빠르게" 설계 변경할 수 있다는 것을 의미 합니다.

그런데 **FPGA** 에서 이 확장성은 사실 가장 큰 무기 이기도 합니다.

그런데 왜 이런 말을 굳이 썼을까? 라는질문을 할 수 있습니다. (저만궁금한가요? 쩝...)

그래서 **Extensible Processing** 이라는 두개의 단어를 같이 엮어서 생각하면 징크라는 새로운 디바이스를 잘 이해할 수 있습니다.

징크에서 **"Processing"** 이라는 것은 **Arm processor** 를 의미 합니다.

아시다시피 **ARM** 은 프로세서 디바이스를 만들지 않고 **IP** 를 만드는 회사 입니다.

워낙 임베디드 시스템에 많이 사용하다보니 **ARM** 은 임베디드 시스템에서 자주 사용되는 여러 주변장치들도 **IP** 로 가지고 있습니다.

이 징크 디바이스에는 이런 **ARM Processor** 와 임베디드 시스템에서 자주 사용되는 **IP** 가 "하드코어" 형태로 내장되어 있습니다. 디바이스 입니다.

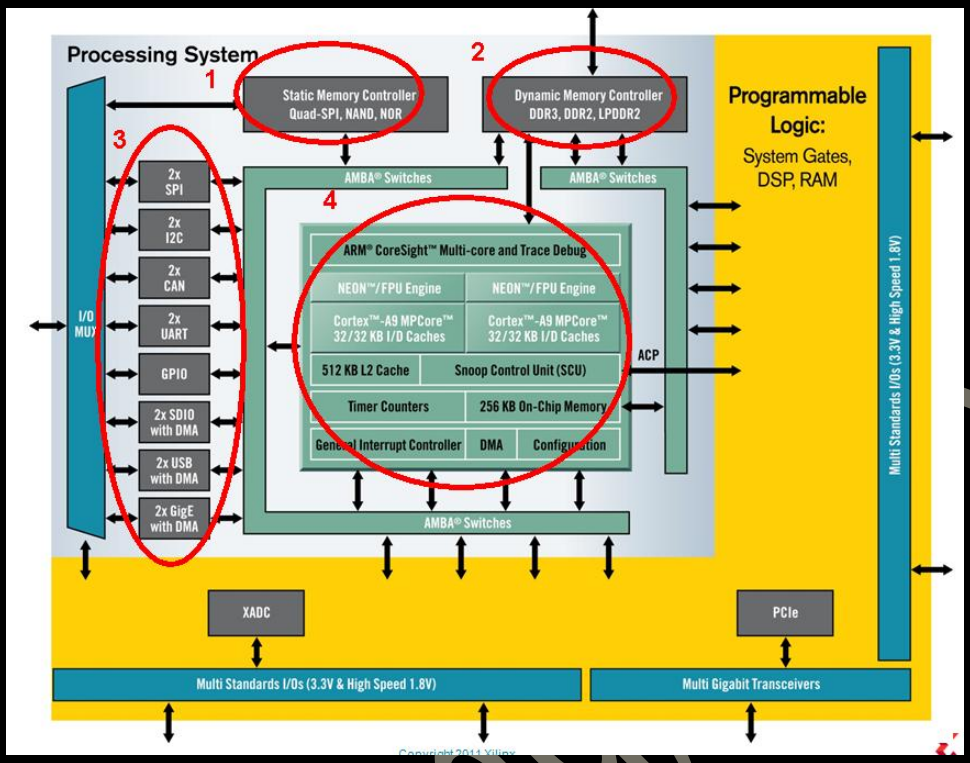
사실 **ARM** 과 주변 장치가 내장된 디바이스는 징크가 처음이 아닙니다.

하여간 무쟈게 종류도 많고 가격도 지극히 저렴 합니다.

하지만 이런 디바이스는 한번 기능이 고정 되면, **ASIC** 으로 만들게 되면, 더 이상의 설계변경은 없습니다.

하지만 징크는 다릅니다.

다음 그림을 잠깐 보시면 아시겠지만 기존 **ARM Processor** 와 임베디드 시스템에서 자주 사용되는 주변장치가 내장되어 있습니다.



FPGA 임베디드 프로세서를 사용해 분들은 알고 있으시겠지만 이 칩에 들어가 있는 프로세서 블록과 기존에 자일링스에서 제공하는 임베디드 시스템과는 어떤 차이가 있을까요?

기존 FPGA 의 로직을 포함하고 있어서 확정된 ARM 프로세서와 주변장치 이외의 기능을 손쉽게 추가하고 변경시킬 수 있습니다.

칩크는 현재 모두 4 개의 디바이스로 구성되어 있습니다.

Z7010, Z7020, Z7030, Z7045

예전 자일링스 디바이스를 보면 디바이스 파트 명에 포함된 숫자가 일반적으로 로직셀의 크기를 의미했습니다. 하지만 칩크의 경우는 로직셀의 크기를 표시하는 방법이 약간 다릅니다.

이들 구성에 대해서는 뒤에서 차츰 설명하도록 하겠습니다

현재 전세계 임베디드 시장의 규모는 모두 **$12.7B** 정도로 추정하고 있으며 자동차 부터 우주 항공까지 그 범위는 매우 광범위 합니다.

다양한 시장의 요구사항을 맞추기 위해서 임베디드 시스템은 점점 복잡해지는 기술적인 스펙을 만족시켜야 합니다.

"기술적인 스펙을 맞춘다" 라는 것이 말이 쉽지 적은 비용으로 짧은 시간에 이런 스펙을 맞춘다는 것은 사실 쉽지 않습니다.

스펙을 이해하고 스펙에 맞는 블록도를 그리게 되면 아마도 "하나의" 프로세서 또는 "하나의" ***ASIC*** 또는 "하나의" ***FPGA*** 로 이런 스펙을 커버하는 것이 불가능 하지는 않지만 늘 부족 합니다.

그래서 요즘 임베디드 시스템을 보면 프로세서와 ***ASIC*** 또는 프로세서와 ***FPGA*** 와 같이 **2** 가지 이상의 디바이스를 조합함으로써 기술적인 스펙을 맞추려고 합니다. (굳이 제 말씀에 동의하지 않아도 됩니다. ^^)

The Need	The Limitations
Higher Performance	**Microprocessors have insufficient signal processing**
Lower Cost	**Multiple chip implementations are too expensive**
Lower Power	**Multiple chip implementations use too much power**
Smaller Form Factor	**Multiple chip implementations take up too much room**
Greater Flexibility	**ASICs/ASSPs cannot adapt to rapid changes in requirements and provide competitive differentiation**

실제 임베디스 시스템을 설계하기 위해서는 주어진 설계 규격을 잘 살펴봐야 합니다. 혼자서 다 할 수 없기 때문에 선배님, 후배님 도움을 받아 입출력 조건 및 소프트웨어가 처리해야 하는 업무량을 잘 살펴보면서 찬찬히 설계를 해야 하지만....실제로는 그렇지 않죠.

개발기간이 짧고 업무지식이 서로 다른 동료와 같이 일하다보면 이런 저런 사고치는 사람도 많습니다. 제일 꼼꼼한 사람이 이런 사람 뒤치닥거리 하다 보면 어딘가 회로도를 그리는데도 실수를 하고 설계스펙을 잘못이해 하거나 기구물 규격을 잘못이해해서 커넥터가 들어가지 않는 경우도 있습니다.

하긴 전원 설계 잘못해서 보드에서 불나는 경우도 종종 있습니다. (불이야~~~~)

전통적인 연구실은 이렇게 꼼꼼하게 원칙대로 일하기 보다는 선배들이 했던 작품에서 조금씩 조금씩 변형시켜 기능을 완성하게 됩니다. ㅋㅋ 창의력과는 거리가 서쪽이 동쪽에서 먼 것 처럼 멀게 느껴집니다.

썰을 너무 풀었네요

The Need	The Limitations
Higher Performance	**Microprocessors have insufficient signal processing**
Lower Cost	**Multiple chip implementations are too expensive**
Lower Power	**Multiple chip implementations use too much power**
Smaller Form Factor	**Multiple chip implementations take up too much room**
Greater Flexibility	**ASICs/ASSPs cannot adapt to rapid changes in requirements and provide competitive differentiation**

자 다시하넌 앞에서 그림에서 살펴보면 실제 임베디스 시스템은 주어진 기능을 최소한의 하드웨어와 소프트웨어 환경의 조합으로 값싸게, 신뢰성 있게 만드는 것이 중요 합니다.

커피 자판기 만드는 것이 아니라면 "High Performance"는 늘 고민 거리가 됩니다. 프로세서를 좋은 것을 사용해야 하는데 그러자니 한참 비싸고 PCB 를 조립하는 것도 만만치 않죠. 그리고 프로세서 순차실행이라는 치명적인 (?) 특성이 있습니다.

이게 좋은 점도 있지만 "High Performnace" 측면에스는 순차실행 보다는 병렬 실행이 좋죠. 요즘 왠간한 32 비트 프로세서는 약간의 병렬 실행을 하기 위한 기능들이 추가 되어 있지만 … 제대로 사용하기에는 워낙 까다로운 일이 아닙니다.

따라서 이런 "High Performance"를 만족시키기 위해서는 하나의 프로세서로는 부족하고 프로세서 옆에 조그만 ASSP 나 FPGA 를 추가해서 만족시킵니다.

음~~~ "High Performance"를 만족했나 생각해 보니, 아이고 프로세서 옆에 덩그러니 프로세서 만한 덩치를 가진 놈이 붙어 있으니 PCB 의 모양 흔히 말하는 '가다'가 나오지 않습니다.

매끈하게 만들려고 했지만 운명은 그렇게 호락호락 하지 않네요.

그리고 영어로 말하는 'side effect' 한글로 번역하면 '부작용' 이 몇가지 눈에 확 들어 옵니다.

젠장 PCB 커진 것은 이해하는데 전원 설계하는 것이 녹녹치 않습니다. 깔끔하게 5V 하나 쓰면 좋겠지만 그런 반도체를 찾는게 더 어렵습니다. 전력도 생각보다 많이 사용하게 되지요. 그리고 ASSP 나 FPGA 하나 추가했는데 줄줄이 흥부 새끼들 마냥 따라서 붙여야 하는 부품들도 한두가지가 아닙니다. 하나 못해 컨덴서 하나라도 붙여야 하니까요

그래서

마지막으로

깔끔하게 마무리 하는 결론으로

ASIC 이 있습니다.

"모르면 용감하다"는 얘기가 있습니다.

너무 점잖게 얘기 했군요

"무식하면 용감하다"

ASIC 이라는 것이 일단 성공적으로 나오게 되면 너무 너무 행복합니다.

가격 저렴하죠, 패키지 하나에 몽땅 들어가 있으니 **PCB** 깔끔하죠. 전기 덜 먹죠~~

그런데 이게 성공하기 하기에는 상당히 많은 '오까네'와 '시간'이 필요 합니다.

게다가 하나의 패키지에 추가해야하는 기능이 많아지면 많아질 수록 ***ASIC*** 공정이 얇아지는데 그 비용이 지수함수 마냥 켜져요.

그냥 '억'대가 됩니다. '억' 옆에 '**0**'이 하나 더 붙기도 하고 떨어지기도 하지만 하여간 생각보다 '오까네' 많이 듭니다.

그리고 제조 공정상 오늘 ***ASIC*** 을 맡기면 담주에 칩이 나오지 않아요.

상황에 따라 다르지만 **8** 주, **12** 주, **16** 주 정도,

감이 잘 안옵니까**?**

흔히 계절이 한번 바뀌야 칩이 나오는 경우가 많다…이 얘기 입니다.

ㅋㅋㅋ

하여간 성공하면 대**~~~**박.

그런데

설계가 바뀌야 한다.

내부 사정에 따라 다르겠지만 처음 ***ASIC*** 을 만들때 들인 '오까네'와 '시간'과 똑같지는 않지만 상당한 수준의 희생이 불가피 합니다.

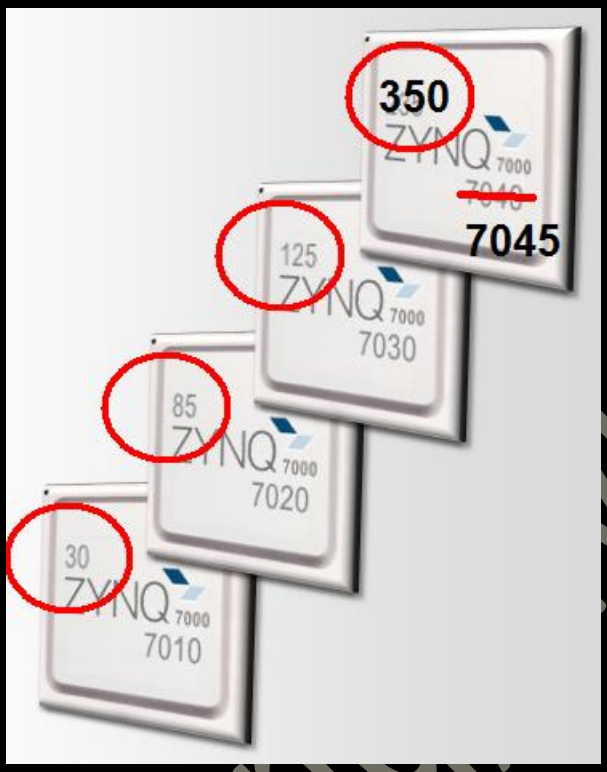
근데 실수라도 하면

그때는 옷벗고 나와야죠... 맘 좋은 사장이나 돈많은 회사라면 좀 더 버티겠지만.

	ASIC	ASSP	2 Chip Solution
Performance	+	+	■
Power	+	+	-
Unit Cost	+	+	-
TCO	■	+	+
Risk	-	+	+
TTM	-	+	+
Flexibility	-	-	+
Scalability	-	■	+

+ positive, - negative, ■ neutral

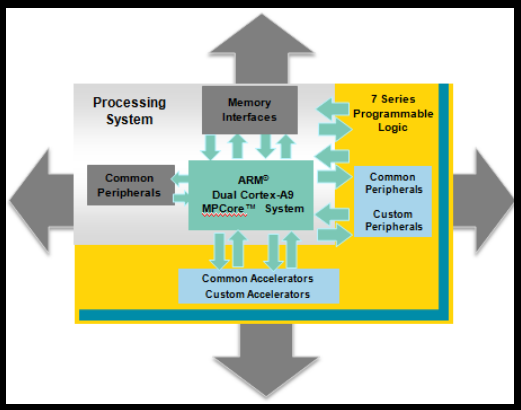
그래서 프로세서 하나로 하든, "2 Chip Solution"으로 하든 "ASIC"으로 하든 임베디스 시스템을 만들 때는 장단점이 '학~~실하게' 드러납니다.



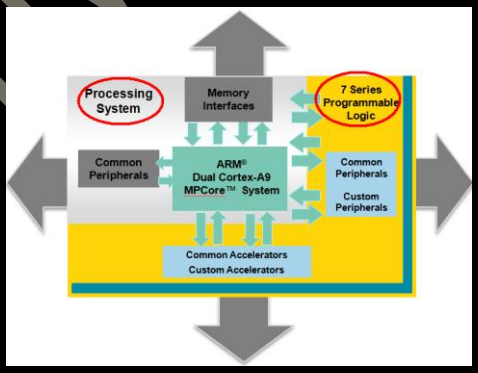
이런 이유로 나온 반도체 칩이 바로 "Zynq" 입니다. 모두 4 종류가 있고요 7010,20, 30, 45 라는 이름으로 불리 옵니다.

동그라미 안쪽에 있는 것이 바로 로직셀의 크기 입니다. 30K, 85K, 125K, 350K 로직 셀들이 들어 있습니다.

다음 그림은 징크라는 디바이스의 특징을 가장 간략하게 표시한 블록도 입니다.



화살표들이 뾰족뾰족하게 튀어 나오고 있지만 다 다름 이유가 있어요. 한번 쳐들아가 보겠습니다. 일단 2 개의 단어가 눈에 보이네요.



뭐죠? 옆에 있는 그림 처럼 Processing System 과 7 Series Programmable Logic 단어가 있습니다.

이것을 줄여서 PS, PL 이라고 합니다. 앞으로 계속 많이 나오는 단어이니 꼭 기억해 주시기 바랍니다.

그럼 PS 와 PL 은 뭔가?

먼저 PS 에는 아래와 같은 기능들이 들어 있습니다.

- **Complete ARM®-based Processing System**
  - Dual ARM Cortex™-A9 MPCore™, processor centric
  - Integrated memory controllers & peripherals
  - Fully autonomous to the Programmable Logic

애게~~ 이게 뭐예요 하면 솔직한 좀 서운하고 차례대로 설명 하겠습니다. 첫번째 "Dual ARM …" 의 의미는 징크 PS 블록에 CortexA9 이 2 개 들어 있다는 얘기죠. 두번째 "Integrated…"는 징크 PS 블록에 메모리 콘트롤러와 각종 주변 장치들이 있다는 얘기 입니다. 세번째 말이 좀 골때리는 말인데요. 이 부분은 고객 여러분들이 잘 못느끼는 부분일 겁니다.

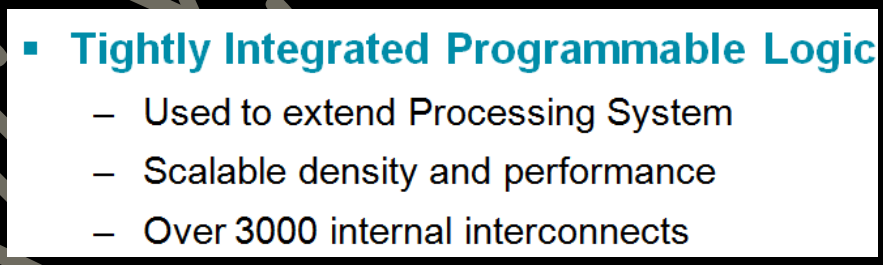
이말이 무슨 말이냐 하면 PS 와 PL 이 서로 독립적이라는 얘기 입니다

좀 생뚱맞은 말이긴 하지만 무슨 내용인지 살펴보면 PS 와 PL 의 전원은 서로 따로 가질 수 있습니다. 그러니까 PS_Vcc, PL_Vcc, PS_Gnd, PL_Gnd 같이 PS 와 PL 전원 핀이 따로 있다는 얘기 입니다.

이게 도대체 무슨 얘기냐 하면 PS 블록은 PL 블록에 전원을 공급하지 않더라도 따로 동작할 수 있다는 얘기지요.

예 맞습니다. 예전 자일링스에서 프로세서를 여러 방법으로 구현할 수 있었지만 결국 PL 블록, 다른 말로 FPGA 가 컨피규레이션 되어야 프로세서가 동작 했습니다.

극단적으로 징크는 PL 에 전원을 공급하지 않아도 PS 는 동작할 수 있습니다.



자, 다음으로 살펴볼 내용은 PL 에 대한 특징인데 기본적으로 PL 은 "Extend Processing System"으로 사용된다는 얘기 입니다.

이게 또 무슨 골때리는 얘기냐 하면 원래 프로세서는 입출력 규격이 정해져 있지 않습니까? 그렇죠. 프로세서는 뭐, 메모리 콘트롤러는 몇개, GPIO 는 몇개 등등 이미 어디다 사용할지가 대략적으로 정해진 상태에서 각종 주변 장치의 갯수와 최대 입출력 속도가 거의 정해져 있습니다.

이런 상태에서는 시스템의 스펙에 변화가 생겼을 때 소프트웨어로 처리할 수 있으면 좋을 텐데 그렇지 않는 경우가 많다 이겁니다.

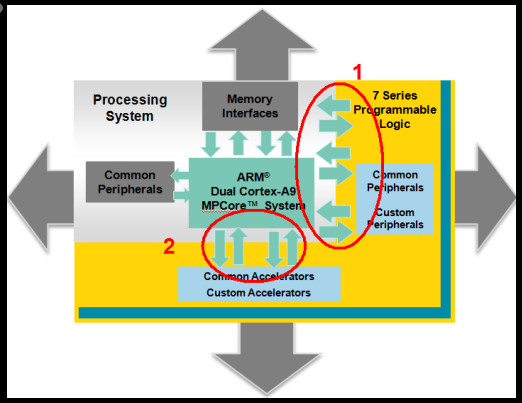
그런데 여기에는 약간의 머피의 법칙이 작동을 하게 됩니다. 무슨 얘기냐 하면 "아~ 그 기능을 필요없게 되었어요. 함수를 처리하는 시간이 줄어들었으니까 좀더 낮은 스펙의 프로세서를 사용해도 문제가 없겠습니다"

이런 얘기는 거의 들어보지 못했습니다.

거의 모든 상황에서 기술적인 스펙은 증가하게 되고 필요로 하는 주변장치는 조금씩 늘어나게 됩니다.

어떻게 하지? 난감하네..

이렇게 골때리는 순간에 징크는 매우 큰 힘을 발휘 합니다. 아까 봤던 그림 다시한번 더 불러보겠습니다. 이 그림 자일링스에서 심심해서 마구 그린 그림이 아니예요.



하여간 옆에 있는 그림을 보면 PS 블록과 PL 블록이 화살표로 연결되어 있지 않습니까?

이 화살표로 표시된 부분이 바로 PS 와 PL 을 연결하는 부분인데 1 번 표시된 화살표가 2 번 표시된 화살표보다 조금 두껍게 설계되었습니다. 결론부터 풀어보자면 1 번은 64/32 비트 데이터 폭을 가지고 있고요 2 번은 32 비트의 데이터 폭을 가지고 있습니다.

화살표 갯수와 방향도 의미가 있습니다. 화살표 갯수와 초등학교만 졸업해도 알 수 있듯이 데이터 포트가 그만큼 있다는 얘기죠, 1 번에 6 개, 2 번에 4 개

화살표의 방향은 마스터가 누구냐르 말해주고 있습니다. 데이터 입출력 방향을 말하는 것이 아닙니다.

- **Flexible Array of I/O**
  - Wide range of external multi-standard I/O
  - High performance integrated serial transceivers
  - Analog-to-Digital Converter inputs

자 마지막으로 징크의 또 다른 특징을 살펴보면 "Wide range of external … I/O" 라는 얘기나 나옵니다.

자일링스가 Virtex6 패밀리에서 좀 고생한 측면이 뭐냐 하면 3.3V LVTTL, LVCMOS 와 같은 IO Standard 를 지원하지 않았어요.

Spartan6 패밀리에서는 지원했지만 고성능 FPGA 인 Virtex6 의 경우에는 좀 만만치 않는 과제 였습니다.

자일링스를 싸랑하는 고객분들이야 너그러이 이해해 주셔셔 다행이였지만 고생좀 적지 않게 했죠.

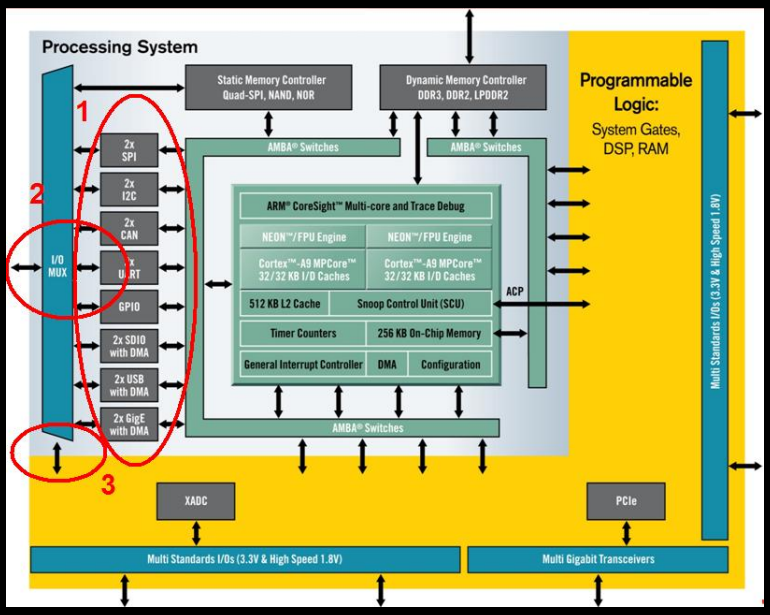
징크는 3.3V 를 공식적으로 지원 합니다.

게다가 Serdes 도 하드코어로 들어가 있습니다. 비록 7030, 7045 에만 있지만요. 그리고 12 비트 1MBPS/Sec, Dual input ADC 가 내장되어 있네요.

요즘 FPGA 를 사용하는데 Serdes 는 기본중에 기본 기능이 되었는데요 특별히 징크에서는 패키에 따라 6.5Gbps, 12.5Gbps 두 종류를 지원 합니다.

자이제 좀더 내공을 쌓는 의미에서 PS 블럭을 좀 더 자세히 살펴보겠습니다.

옆에 그림은 자주 나오네요. 쩝… 중요하니까 그렇죠.



먼저 1 번을 보겠습니다.

음…. 임베디스 시스템에서 자주 사용하는 주변장치가 쭉 늘어서 있네요.

SPI, I2C, CAN, UART, GPIO, SDIO, USB2.0, GigE 가 각각 2 개씩 들어 있습니다.

예 모두 2 개씩 들어 있습니다.

게다가 SDIO, USB2.0 GigE 에는 DMA 까지 들어 있습니다.

DMA 가 지원된다는 것은 무슨 의미인가? 음…이 주변장치 장치를 통해서 메모리 억세스가 무쟈게 많다는 것을 의미 합니다.

뭐 당근 그럴만도 하죠.

그런데 DMA 에도 족보가 있다는 사실을 아십니까?

일반적으로 우리가 알고 있는 DMA 는 Simple DMA 입니다. Simple DMA 는 Source, Destination, Length 를 정해주죠. 그리고 Start 신호를 주면 DMA 는 주어진 만큼 복사해 두죠.

그런데…이렇게 간단한 DMA 기능으로는 약간 아쉬운 점이 없지 않아 있습니다.

음…조금은 복잡한 얘긴데…이런 DMA 를 Simple DMA 라고 불러요. Simple DMA 가 있으면 Complex DMA 가 있나? 이렇게 생각하는게 인지상정이죠.

사실 Simple DMA 가 처리하기에는 약간 "에~~~매한 기능들이 있어요" 이런 얘매한 영역이 어디에 있느냐 ? 주로 이더넷을 처리할 떄 발생 합니다. 왜냐하면 ~~~

이더넷은 프로토콜 특성상 데이터를 패킷 형태로 주고 받지 않습니까? 패킷형태라고 하는 것은 데이터의 시작과 끝이 있고 시작과 끝 사이에 데이터 즉 페이로드 (payload)가 있는 형태 입니다. 그런데 이더넷 프로토콜에 의해서 데이터를 주고 받을 때 하나의 패킷으로만 이루어지지 않고 여러개의 패킷으로 쪼개서 주고 받게 되죠.

그러니까 패킷이 여러개가 들어오기는 하지만 그걸 나중에 한 덩어리로 조립을 해야 제대로 된 데이터를 만들 수 있습니다.

이럴 때 패키마다 Src, Dest, Length 를 지정하려면 프로세서가 꽤 많이 DMA 내부에 있는 레지터스를 억세스해야 합니다.

이런 문제를 방지하기 위해 DMA 가 스스로 Src, Dest, Length 를 읽어 갈 수 있도록 일종의 자료구조론에 의거한 쌍방향 선형 리스트 구조를 메모리에 적어 놔야 하는데…

그럴려면 보드 디스크립터라는 것을 만들어야 하고 링구조를 만들고 … 쩝, 무슨 얘기 하는지 잘 모르겠죠… 하여간 이런걸 거치면 좀더 효율적인 DMA 를 만들 수 있습니다. 이 DMA 를 우리는 Scatter Gather DMA 라고 합니다.

그러니까 Ethernet, USB, SD Controller 에는 이런 SG DMA 가 있다는 얘기죠.

이런 얘기를 들으면 "아이고 이걸 어떻게 만들어…"라고 할 수 있지만 사실 이런 거는 자일링스에서 대부분 디바이스 드라이버를 제공하고 있습니다.

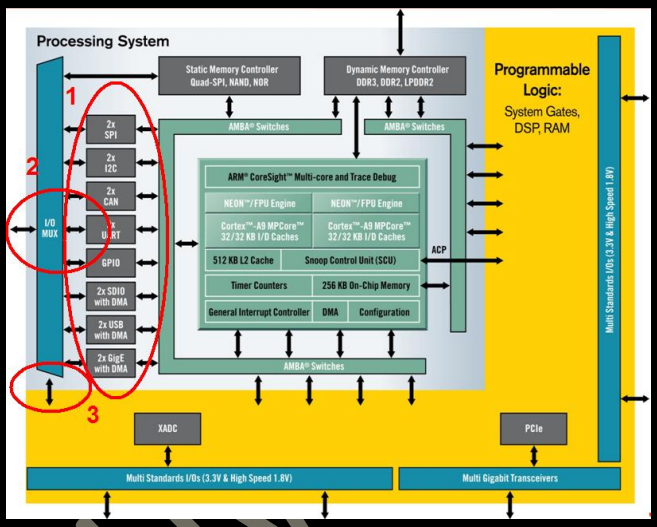
구구절절한 설명은 일딴 접어두고 다음 얘기를 진행 하도록 하겠습니다.

Ethernet 의 경우에는 현재 10Mbps, 100Mbps, 1000Mbps 모두 지원 합니다.

USB2.0 의 Full, High speed 모두 지원하고요 OTG 라는 프로토콜도 지원하고 있습니다. 또 USB2.0 은 디바이스 모드나 호스트 모두 모두 동작 합니다.

나머지 주변장치에 대해서는 다음에 기회가 되는대로 설명 하겠습니다.

이제 두번째 항목을 알아 보겠습니다.



옆에 그림에서 2 번을 자세히 보면 IO MUX 라고 표현되어 있습니다.

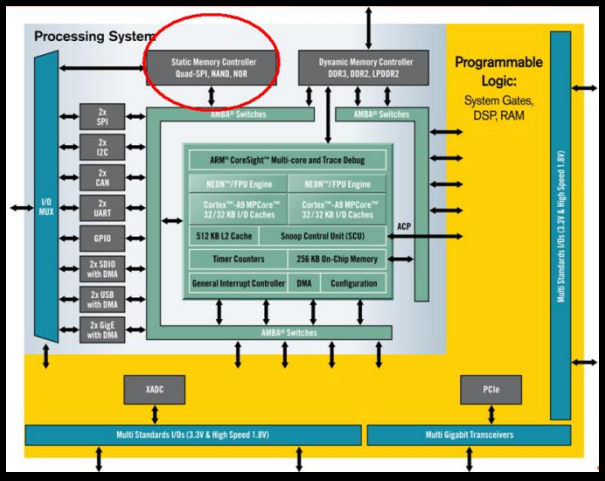
IO MUX 는 모두 54 개의 IO 를 가지고 있는데, 징크가 가지고 있는 주변장치의 모든 IO 갯수를 합친 것 보다는 IO 숫자가 적기 때문에 MUX 를 사용한 것 입니다.

예를 들어 Ethernet 같은 경우에는 RGMII 인터페이스를 사용하는데 이 신호는 대충 머리속에서 계산 해보면 (RxD, TxD, RxC, TxC, CRS, DV,…4 + 4+ 1 + 1 + 1 + + ➔ 12 개 보다는 크죠) 12 개 대충 후려쳐도 그정도 되는데 Ethernet 이 2 개 있으니까 24 개 되죠. 그리고 32Bit GPIO.

음 이것만 해도 모두 56 개가 되는데 어려면 당근 54 개의 IO 와 연결하려면 선택적으로 연결해야 합니다. 그래서… MUX 가 사용되었습니다.

다음으로 3 번을 그림을 보면 이해할 수 있는 것은 결국에 설계자가 54 개의 IO 를 넘어서는 IO 를 사용한다 치자면 어떻게 할거냐?

이럴 때는 3 번으로 퉁친, EMIO 라는 포토를 통해 PL 블록에 있는 IO 를 사용한다는 말씀입니다.



자 이제 다음 그림으로 넘어 갑니다. 같은 그림을 계속 우려먹어서 좀 그렇지만 하여간 징크는 모두 3 가지 종류의 플레시 메모리 콘트롤러를 지원 합니다.

이게 뭐냐하면 꼭 외워두세요

NOR 플레시 메모리야 지금까지 BPI 컨피규레이션 모드를 사용했던 FPGA 설계자 분들이 가장 많이 사용한 부품이죠.

NOR 플레시 메모리 데이터 쉬트를 보면 알겠지만 이게 핀수가 꽤 많아요. 그래서 FPGA 개발자들이 조금은 불편해 했던 디바이스 입니다. 예를 들어보면 데이터 비트수가 16 개이고 어드레스 핀이 보통 23 개 정도 되면 16 + 23 + RD + CS 등등 FPGA 에 연결되는 놈만 보더라도 41 개… 8 비트 데이터를 사용한다고 해도 33 개.

그러면 54 개의 IO MUX 에서 한 2/3 정도 사용하죠.

그래서 아주 아주 특별한 경우가 아니라면 그냥 NOR 는 스킵

두번째 NAND 는 사실 NOR 보다는 조금 낮습니다. 왜냐하면 NAND 는 어드레스와 데이터 핀을 서로 같은 핀을 사용하거든요. ALE (Address Latch Enable) 신호가 있어서 현재 어드레스가 나오고 있다, 데이터가 나오고 있다 등을 구분해 줍니다.

그래도 적은 핀은 아니죠.

그러면 가장 추천 받는 플레시 메모리는 뭐냐?

당근 QSPI 입니다.

SPI 인터페이스는 굳이 여기서 설명하지 않더라도 Q 의 의미만 알면 대충 그림이 그려 지죠.

이 Q 는 Quad 를 의미 합니다. '4' 죠. 그래서 4 개의 SPI 플레시 메모리를 하나로 만들어서 데이터 비트수가 4 개이고 SPI 인터페이스를 가지는 플레시 메모리를 연결할 수 있다는 것 입니다.

속도도 제법 좋고요 무엇보다 핀 수가 적어요. 그래서 징크를 사용하는 개발자는 QSPI 를 가장 좋아 한답니다.

그런데 이렇게 질문하면 조금 곤란하죠~~~

앞에 나온 그림을 정말 그림책으로만 이해하지 못했다, 이렇게 말할 수 있습니다. 다음과 같이 차례대로 근거를 제시해 보겠습니다.

NOR, NAND, QSPI 는 모두 어디로 연결되어 있습니까?

IO MUX 맞습니다.

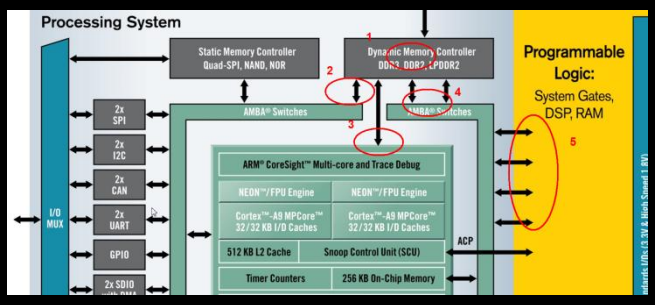
IO MUX 의 **IO** 갯수는 몇개 입니까?

54 개 맞습니다.

만일 **3** 개 플레시 메모리 콘트롤러를 모두 사용한다면 **IO** 갯수가 **54** 개 넘을까요 안넘을까요?

자 이제 징크의 가장 큰 특징중 하나를 보여주는 또다른 부분을 자세히 뜯어 보겠습니다.

옆에 있는 그림에서 자세히 살펴 보면 메모리 콘트롤러를 볼 수 있습니다.



일반적으로 임베디스 시스템은 프로세서가 메모리 콘트롤러의 마스터로 동작 합니다. 여기서도 그런 부분이 보이고 있죠.

1 번과 3 번을 연결한 화살표 보이시죠. 이 화살표가 프로세서와 메모리 콘트롤러가 서로 연결된 것을 보여 줍니다. 즉 프로세서에서 메모리를 억세스할 수 있다는 얘기 입니다.

일단 메모리 콘트롤러를 살펴보면 이것이 지원하는 메모리의 종류가 모두 3 종류임을 알 수 있습니다.

DDR2, DDR3, LPDDR2, LPDDR2 는 Low Power DDR2 라는 얘기죠.

DDR2 는 1.8V, DDR3 는 1.5V, LPDDR2 는 1.2V 를 주로 사용 합니다.

하여간 징크는 3 종류의 DRAM 중 하나를 선택해 사용할 수 있습니다. 그런데 이 메모리의 콘트롤러의 특징을 나타내는 가장 중요한 부분이 있는데 이 부분이 2, 3, 4 번으로 표시되어 있네요

아까도 잠깐 말씀드렸지만 일반적으로 임베디스 시스템의 메모리의 마스터는 프로세서 입니다. 그런데 ~~~ 그런데~~~ 좀 돈 될만한 임베디스 시스템은 메모리의 마스터가 하나만 있지 않죠. 대표적인 예가 DMA 입니다.

음 ~~ 근데 이 그림에서는 2 번 화살표와 연결된 ARMBA-SWITCH 에는 이미 USB, Ethenret, SD/SDIO 를 위한 DMA 이미 포함 되어 있습니다.

즉 USB, Ethernet, SD/SDIO 는 이미 메모리를 읽고 쓰기 위한 하드웨어 준비가 된 상태를 의미 합니다.

이 메모리 콘트롤러의 가장 큰 특징은 4 번 입니다.

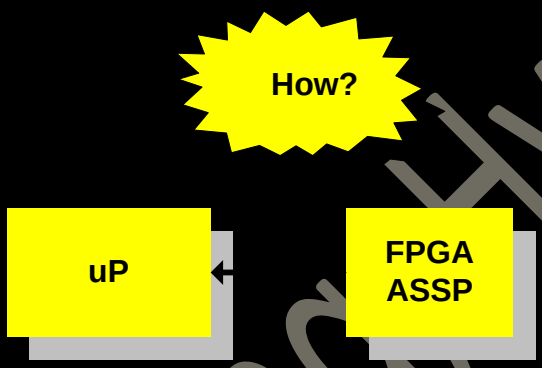
4 번에 연결된 AMBA SWITCH 를 보면 5 번과 같이 PL, Programmable Logic, 과 연결되어 있는 것을 확인할 수 있습니다.

근데 이게 가장 중요한 부분중에 하나이고요 ,,, 이 부분을 사용한다는 것이 바로 징크를 제대로 사용한다고 감히 말 할 수 있다니까요.

일반적으로 임베디스 시스템은 하나의 프로세서로 모든 처리를 다 하는 것이 불가능 하기 때문에 프로세서 옆에 FPGA 나 ASSP 를 하나씩 달고 다닙니다.

문제는 이 2 개의 칩간에 데이터를 어떻게 주고 받을 것이냐? 이것이 문젠데? 정말 이 부분은 가볍게 생각되는 부분이 아니죠.

그동안 상당히 많은 개발자들을 힘들게 하고 많은 시간을 들여서 겨우 겨우 흔히 말하는 인터페이스를 뚫어 놓게 된 것 입니다.



그림으로 보면 간단해 보이지만 저 화살표의 굵기가 두꺼워지면, 다른 말로 bandwidth 가 높아지면, 그냥 그냥 간단히 PCB 에서 패턴만 연결한다고 끝날 일은 아니죠.

그런데 이 고민을 한방에 날려버린 기능이 바로 5 번 이옵니다.

먼저 프로토콜은 뭐냐 AXI 입니다.

굵기는 어떠냐 32 비트 또는 64 비트 입니다.

최대 동작 속도는 얼마냐? 이것은 PL 부분을 어떻게 설계하느냐에 따르지만 제가 본 레퍼런스 디자인은 약 150Mhz 로 설계되어 있었습니다.

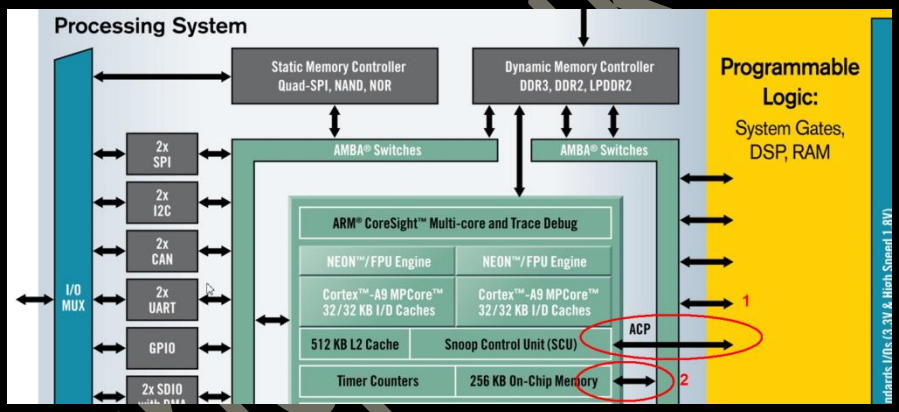
그러면 64 비트 150 ➔ 9.6 G bps 로 데이터를 전송할 수 있는 포트가 모두 4 개라는 얘기죠. 음 최대 메모리 bandwidth 와 비교해보면, 먼저 메모리의 bandwidth 는 32bit * 533 * 2 == 34 Gbps 가 이론적인 bandwidth 이고 클럭에 약 80%정도 데이터가 실린다고 가정하면 34 * 0.8 == 27G 정되 되네요.

아까 포트당 9.6G 라고 했는데 이것을 10G 라고 퉁치고 하면 한 3 개 포트를 사용해야 최대 메모리 bandwidth 를 다 사용한다고 할 수 있네요.

Programmable Logic 입장에서 이런 고속 인터페이스가 모두 4 개 지원되니 개발자들의 부담이 많이 덜어진 것이 사실 입니다.

구체적인 타이밍이나 프로토콜에 대해서는 추후에 차근 차근 보도록 하죠.

자 이제 다음 화살표를 살펴 보겠습니다. 일단 ACP 라고 되어 있는 부분이 있는데 1 번이라고 하는 부분요.



이거 처음에 확실히 이해하는데 조금 시간이 걸렸습니다.

뭐 지금이라고 해서 그닥 정확히 이해하지는 못했는데요 하여간 데이터 쉬트에서 얘기는 하는대로 풀어 보면 이렇게 된다는 것 입니다. 일단 ACP 를 이해하려면 Snoop 의 기능을 알아야 합니다.

스누프가 탄생한 이유는 프로세서가 두개라는 원죄를 안고 있기 때문 입니다. 좀더 얘기를 하면 두개의 프로세서가 모두 캐시를 가지고 있죠. 두 프로세서가 같은 메모리 번지를 가지고 있을 때 한쪽 프로세서가 캐시의 내용을 바꿨다면 다른 한쪽에 있는 캐시의 내용도 당근 바꿔야 하겠죠.

이걸 누가 하느냐, 스누프 콘트롤러가 이런 일을 맞고 있는데 이 스누프가 이일만 하는 것이 아니라 PL 부분과도 이런 비스므리한 일을 ACP 포트를 통해서 한다는 말씀 입니다.

하여간...

좀더 나중에 자세히 살펴보고 정확히 이해 하지 못하는 부분을 애써서 설명하지는 않겠습니다. 쩝 약간 부끄...

다음 2 번으로 표시된 부분을 조금은 흥미로운 기능을 제시할 수 있습니다.

예를 들어보면 프로세서와 로직, 여기서는 PL 부분인데 앞으로는 계속 로직 이라는 표현을 사용하도록 하죠, 은 데이터를 수시로 교환할 필요가 있습니다. 그렇게 하는 할 수 있는 방법은 제법 다양하지만 여기서는 대표적인 예 2 가지만 설명하겠습니다.

첫번째 예가 바로 메모리 콘트롤러를 사용해 징크 외부에 연결한 DDR2, DDR3, LPDDR2 를 이용하는 것 입니다. 늘 설명했던 것 처럼 크게 노력을 기울이지 않고 많은 양의 데이터를 손쉽게 주고 받을 수 있습니다.

두번째 예가 바로 256KBYTE 의 OCM, On Chip Memory,를 이용하는 것 입니다.

2 가지 방법의 가장 큰 차이점은 외부 메모리를 사용 하느냐 내부 메모리 사용 하느냐에 있습니다.

두번째 차이점은 레이턴시에 있습니다. 무슨 말씀이냐면 외부 메모리를 사용하면 데이터를 주고 받는데 필요한 기본 클럭 수가 상당히 크죠. 하지만 OCM 을 거치게 되면 외부 메모리를 거치는 것 보다는 상대적으로 클럭 수가 적습니다.

하지만 주고 받기 위한 데이터의 양은 상대적으로 OCM 이 적죠. 이 두가지 조합을 잘 이해해야 하겠습니다.

일단 이렇게 징크 내부에 있는 여러 하드웨어 구조에 대해서 살펴봤고요 다음 부터는 좀더 내부 구조를 자세히 해부해 보도록 하겠습니다.

Xilinx, Zynq, Hyuk