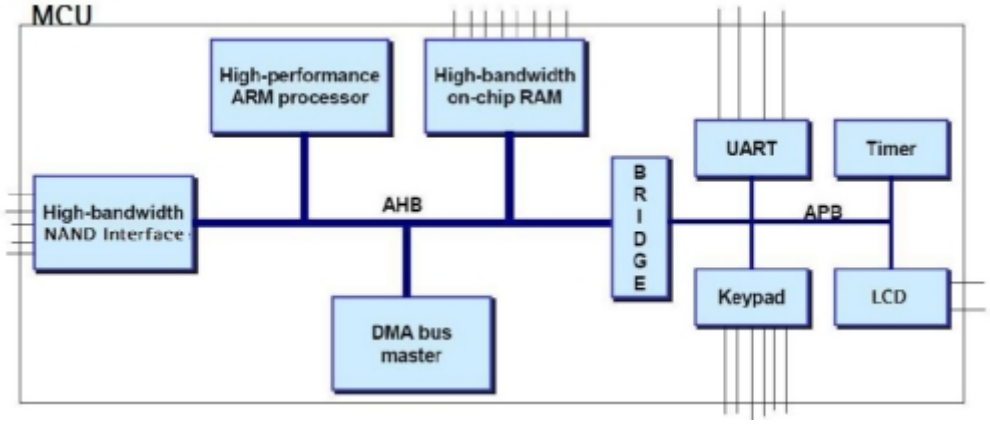
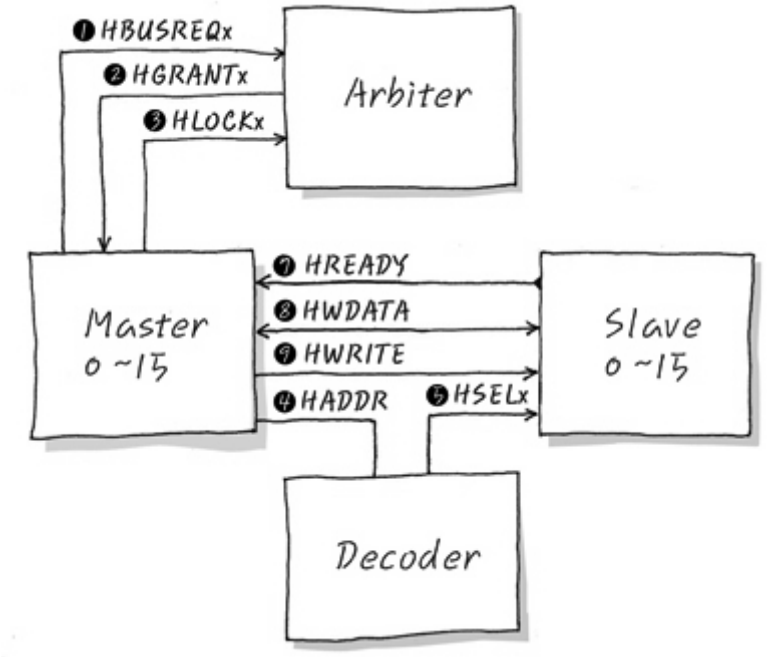
친절한 임베디드 시스템 개발자 되기 강좌 : AMBA

SoC를 보니, IP를 이용해서 Chip을 만들어 내는 구나.. 하네요. 그러면 Chip 내부에 IP만 늘어 놓으면 되느냐. 하면 안되겠죠. 집안에 TV랑 Home Theater를 사다 놓고, 따로따로 늘어만 놓으면 같이 동작하는 게 아니니까, 최소한 전선으로 연결은 해야겠죠. SoC내부에 IP들을 전선으로 연결하는 방법은 Bus를 이용하는 것이지요. - Bus도 뭐 여러 Device가 공유하는 전선이니까 별반 다를 건 없겠지요 - 이 Bus들을 어떻게 연결하고, IP끼리 서로 어떻게 통신 할 것이냐를 약속한 것이 AMBA에요. AMBA (Advanced Microcontroller Bus Architecture)는 영어로 한마디 정의를 하자면SoC Target On-chip bus protocol라고 할 수 있겠네요.역시나 ARM에서 주도하는 Bus 규격이지요. 아무래도 ARM을 SoC에 CPU로 채택하면, ARM의 성능을 최대화 하는 게 좋기 때문에 ARM社가 세상에 Open해 놓은 규격이라 할 수 있겠사옵니다. MCU내부에서 보게 되면, Bus의 통신 방식을 잘 이해 할 수 있고, 지켜줄 수 있는 것이 필요하게 되는데, 이것을 Bus Interface라고 부르고요, 이 Bus Interface는 Bus위에 Data를 어떻게 전송할 거냐, 어떻게 받을 거냐를 잘 control해주는 interface에요. AMBA에서는 이런 Bus Interface가 3가지 종류로 나뉘어요. 그것이 바로, AHB, ASB, APB인데요. AHB는 Advanced High Performance Bus이고요, ASB는 Advanced System Bus이고요, APB는 Advanced Peripheral Bus에요. 헥헥. AMBA는 이렇듯 3가지 Bus 규약 (Protocol)을 약속해 놓았구요. 흔히들 AMBA는 3가지로 구성된다는 말은 이런 뜻인 거죠. AMBA가 3가지로 구성되어 있다고 해서 Bus하나에 모든 Protocol을 지원하는 건 아니고, 어떤 Bus interface에 물려 있느냐에 따라 다른 거죠. 예를 들어, AHB Bus라고 함은 특별 난 게 아니고, AHB Protocol을 지원하는 Bus interface에 물린 Bus라고 해석하시면 됩니다. 뭐 일단 많이 사용되는 Schematic을 하나 보면 좀 더 나으리라는 신념에, 유명한 그림 하나를 선보이겠습니다 .

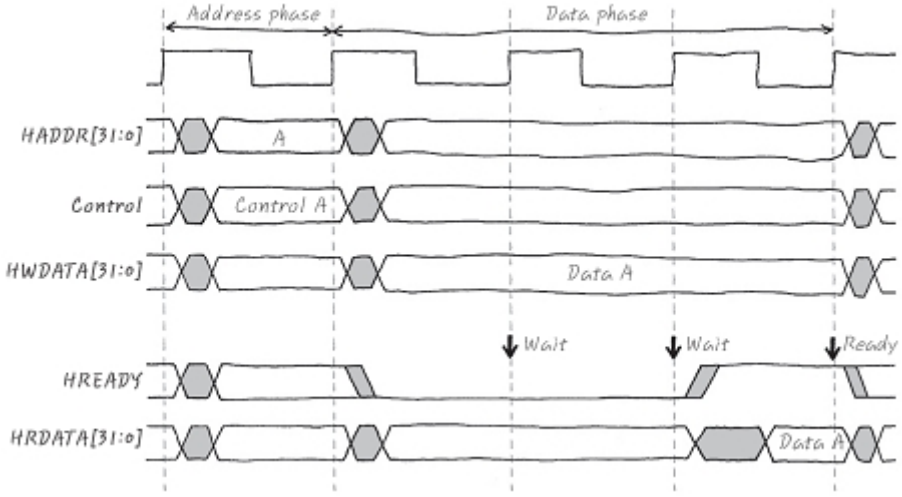


이건 MCU의 내부에 여러 가지 IP들이 AHB, APB Bus로 연결되어 있는 그림이에요. AHB는 이름이 high performance니까, Burst Mode (이빠이 한방에 팍 쏘는)의 Data 전송에 이용되고요, APB는 나름 빠르지 않은 전송속도를 필요로 하는 주변장치들 (Peripheral)등에 연결되어 있어요. 중간에 Bridge는 AHB와 APB의 속도하고, Protocol이 틀리니까, 그걸 연결해 주는 다리와 같은 역할이고요. 그러니까 220V하고 110V를 변환해 주는 트랜스와 비슷한 거라고 보심 틀림없지요. 여기에서 잠깐, 왜 AHB, APB로 회로를 나누느냐 하면, 효율성 문제이지요. AHB로 연결된 녀석들은 APB보다 훨씬 빠르니까, AHB로 모든 IP를 연결해 두게 되면, 빠른 녀석들 끼리 서로 통신을 하다가 중간에 저속의 IP에게 뭔가를 부탁하고 시키게 되었을 때, 저속의 IP가 응답을 해줄 때까지, 고속의 IP들이 Bus를 쓰지 못하고 놀게 되지요. 그러니까, 빠른 넘들 끼리는 빠른 넘들 끼리 놀게 해주고, 그 와중에 느린 넘에게 일을 시킬 때는 Bridge에게 날려놓고. 여기에 왜 ASB는 안 보이느냐? 원래의 AMBA는 ASB와 APB 두 개의 Protocol만 있었지요. 그래서 AHB자리에 ASB가 들어가야 정상인 거에요. 하지만, AMBA도 진화해야 하는 까닭에, Multiplex Bus 기반의 AHB를 발표하였지요. 그러다 보니, ASB가 들어가야 할 자리에 AHB가 떡 하니 자리 잡은 거지요. ASB로도 충분하면 ASB가 있어도 되어요. AHB가 Multiplex Bus기반이라는 말은 Multiplex Bus가 주소라인과 제어라인 그리고 데이터라인까지 모두 공유를 하고, 처음에 주소를 먼저 쏴준 후 데이터를 드르륵 쏴주는 형식의 Burst 데이터 전송 기법을 사용하여, 더욱 좋은 성능을 낼 수 있게 해

주었지요. ASB는 주소, 제어, 데이터 라인이 모두 서로 분리되어 있고, 양방향 Bus 구조이고요, Address와 Data를 번갈아서 쏴주도록 Design되었기 때문에 Burst로 Data를 전송하기 어려운 측면이 있어요. 이렇게 AHB, ASB만 그려 놓으면 다냐? 그건 또 아니죠. 이런 AMBA Bus system을 구성하는 요소에는 IP, Decoder가 있어야겠고요. 각 IP는 Master도 되고, Slave도 됩니다. Master라는 녀석은 Slave에게 Read나 Write 요청을 할 수가 있고요, Slave는 그 요청을 받아 들여서 Read/ Write를 직접 하게 되고요, 이런 명령에 대해서 성공/실패/Wait 등의 상태를 master에게 다시 report하게 되지요. Master는 Data를 Bus에 흘리고자 하는 주체인 게고요, Slave는 그걸 받아서 일을 해야 하는 노예죠. 자, 그러면 Master랑 Slave만 있으면 서로 통신이 가능하느냐! 하면, 그건 또 아니죠. 아~ 자꾸 자꾸 아니래. Master와 Slave는 복잡한 Bus System에 의해서 얽혀 있어요. 그러다 보니, 이런 Bus System을 정리할 신호등이 필요하게 된 거죠. 그게 아비터 (Arbter)라는 넘 인데요. 이 넘은 Bus를 누가 쓸 건지에 대해서 결정권을 가지고 있지요. Arbiter한테 허락 못 받으면 Bus 못써요. 여기서부터 좀 어렵고, 지루할 수 있는데, 이걸 잘 봐두면 모든 Device control은 이것과 같다고 생각하면 되니까, 잘 봐두세요. Memory Control이나, Peripheral Device의 Control은 모두 이와 비슷하다 구요



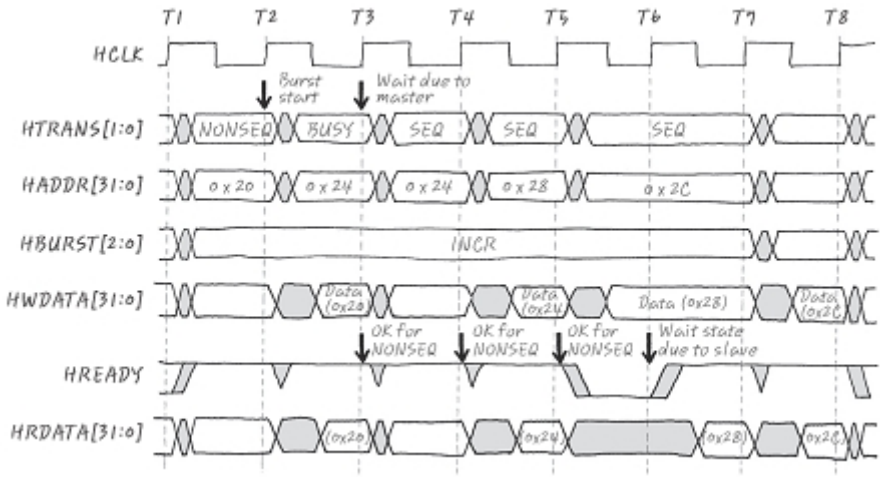
Arbiter가 대빵 자리를 차지하고 있고요 Master와 Slave는 AMBA System에서 16개까지 붙을 수 있습니다요. 그래서 Master 0~15, Slave 0~15 이렇게 달려 있고요. 여기에 Master와 Slave의 길을 제대로 뚫어주는 Decoder가 있습니다요. Decoder는 접근하려는 주소를 보고서, Slave에 Chip Selection과 비스므리한 HSEL 신호를 날려주지요. 야~ 너 누가 너한테 Data를 준단다~ 하고 말이죠. 각 신호의 뒤에 붙은 x는 0~15번을 나타내요. 여러 개의 신호 선이 있다는 뜻이죠. 자, 그러면, 순서를 한번 볼까요? ① 뭔가 하고 싶은 Master는 Arbiter에게 나.. 버스 써도 될까..? 하고 HBUSREQ 신호를 날립니다. ② 이때, Arbiter는 다른 넘이 쓰고 있으면 허가를 안 해주고, 써도 되면 허가를 해주지요. 그 신호가 HGRANT에요. 너 써.. 하고 선심을 쓰죠. ③ 그러면 Master는 내가 이제 쓴다~ 하고 Arbiter에게 HLOCKx 신호를 날려줍니다. 이제부터는 Arbiter가 누군가가 Bus를 쓰고 있군 하면서 메모를 해 놓는 거죠. 동시에 Master들이 서로 쓰겠다고 날리면, 보통 Arbiter의 Policy에 따라 다른데 Priority가 높은 Master에게 Bus 사용 허가권을 날려주지요. ⑤ 허가를 얻은 Master는 접근하고자 하는 Slave의 주소에 HADDR을 날립니다. 그러면 Decoder가 그걸 보고서 HSELx 신호를 해당 Slave에 날려줍니다. 해당 Slave는 동작 활성화가 되는거져. ⑥ 그런 다음에 Write하겠다는 HWRITE 신호를 High로 Slave에 날려요. (Low면 READ라는 의미에욧!) ⑦ 그런 후에, Slave는 HREADY신호를 이용해서 READY가 되면 READY 되었다고 알려줘요. ⑧ 그런 후에 HWDATA 선에다가 Data를 날려주는 거지요. (HRDATA Bus를 통해서 READ도 할 수 있거등요 참~ 쉽죠? 잉) 뭐, 간단하죠? 이게 Bus 동작의 전모에요. 자, 그럼 이 내용을 좀더 유식한 그림으로 함 볼까요? 어디 가서 잘난 척은 해야죠. 앞에서 Arbiter가 동작하는 순간은 이미 결판 났다고 하고요, ⑥ ~ ⑧ 사이의 Timing Diagram들을 좀 보시죠. 이렇게 Bus쓰는 동안에 Data를 주고 받는 과정을 또 유식한 말로 Transaction이라 불러요.



으하핫. 보시면 아시겠지만, Address를 날리는 과정을 Address Phase라고 부르고, Data Phase라고 불러요. 가만히 보면 Address Phase와 Data Phase라는 뭐 있어 보이는 말이 위에 있네요. Address Phase가 Slave에게 주소를 날리는 과정이고요, Data Phase가 Data가 왔다 갔다 하는 과정인 게죠. 근데 여기에 보면 wait이라는 용어가 나옵니다. wait은 아무래도 Slave가 Master보다 느린 Device인 경우가 많으니까, 그런 경우에 나 아직 준비 안되었어요... 하고 HREAY 신호를 Low로 깔아 버리는 거죠. 이게 wait state라는 건데요, wait sate동안에는 Slave가 준비 동작을 하는 거에요. Master야 기다려라~ 그거죠. 그래서 HREADY가 High인 동안에만 Data를 쓰거나 읽는 거죠 머. 위의 그림에서는 control 신호가 HWRITE 신호라고 봐야겠네요? 여기에 덧붙여서, Master와 Slave사이에 control 신호에는 HTRANS[1:0], HBURST [2:0] 등의 control pin이 더 있는데요, 이 녀석들은 Burst Transfer를 어떤 식으로 할 거냐를 알려줘요. 일단은 HTRANS 신호가 어떤 것들을 표현 하는지 볼까요?

HTRANS [1:0]	Type	Description
00	IDLE	Master가 데이터 전송이 필요 없을 나타냄 Slave는 언제나 IDLE에 대해 OKAY response
01	BUSY	Burst 전송 중간에 master가 다음 사이클을 위한 준비가 안되어 IDLE상태를 만들고 싶을 때 사용 Slave는 언제나 BUSY에 대해 OKAY response
10	NONSEQ	단일 데이터 전송이나 Burst 전송을 시작할 때 사용 (전에 전송한 것과 관련 없음)
11	SEQ	Burst 전송시 NONSEQ의 다음부터 SEQ로 동작 (전에 전송한 것과 관련 있음)

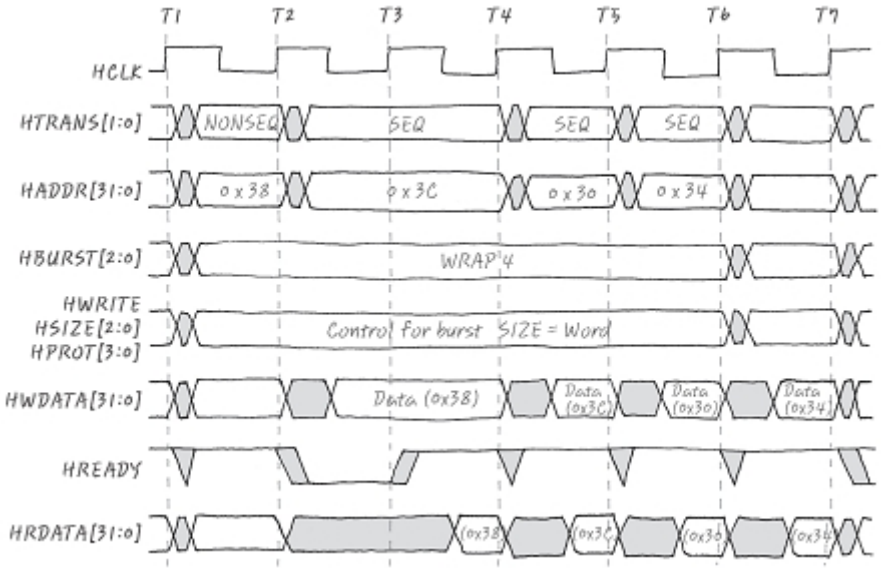
어랏 이것만 봐서는 먼 소린지 잘 모르겠지요. 연속된 주소의 Data를 주르르륵 전송할 때 아~주 유용하답니다. 이런걸 Burst Transfer라고 부르고요! 또, Diagram 하나 볼게요. Data를 보낼 때, HADDR이 0x20에서부터 0x2C까지 연속되지요? 그러면 HADDR이 시작 되는 0x20에서 NONSEQ와 주소를 날려주시고요, 한번 Busy 때려주시고요. 그 다음부터는 wait이 아닌 경우에는 SEQ와 주소가 자동으로 늘어나면서 계속 연달아서 날려 주시는 거죠. wait은 Slave의 사정이니까 wait이 걸리면 기다려 주는 게 Master의 매너죠. 아래에 표시되어 있는 Busy는 Slave쪽에서의 Wait state와 마찬가지로, Master가 준비 안되었을 때 Busy를 날려주고 잠시 기다려라라고 알려줄 수 있는 신호죠.



Burst Transfer의 장점은 무엇이냐! 원래는 하나의 Address phase + 하나의 Data Phase로서 하나의 Transaction이 끝나는 게 정석인데, 하나 이상의 연속된 Data를 쫘르륵 쓸 수 있다는 장점이 있지요. 자동으로 늘어나는 주소와 함께.. - 보통의 NOR같은 메모리의 경우에 Burst mode라는 게 있는데 이걸 이용하게 되면, 연속된 주소라면 Address의 하위 Address pin만 계속 변한다는 특징이 있으니까, 상위 주소는 그대로 두고, 하위 Address pin만 깔짝 깔짝 움직여 주면, Data가 연속적으로 딸려 올라온다는 장점이 있지요. 뭐, 여하튼 그림 보니까 INCR이라는 HBURST[2:0]라는 신호도 있네요? 요 녀석은 BURST에도 여러 가지 종류가 있다! 뭐 그런 거에요. 이런Burst의 종류에는 머머 있나 함 보시죠.

HBURST [2:0]	Type	Description
000	SINGLE	Single transfer
001	INCR	Incrementing burst of unspecified length
010	WRAP 4	4-beat wrapping burst
011	INCR 4	4-beat increment burst
100	WRAP 8	8-beat wrapping burst
101	INCR 8	8-beat increment burst
110	WRAP 16	16-beat wrapping burst
111	INCR 16	16-beat increment burst

오, 꽤나 여러 가지 구색을 갖춰두었네요. 가만히 보니까 INCR이랑 WRAP계열 두 가지로 나뉘는 거 같죠? 각각의 특징이나 한번씩 살펴 보시져 머. WRAP 계열은 뒤에 붙어 있는 숫자가 Address Boundary에요. 그러니까, 예를 들어서, WRAP4의 경우에는 주소가 4의 배수로 Boundary를 이루고요, 그 범위 안에서 Burst를 할 수 있는 거에요. 그리고 INCR도 마찬가지로 뒤에 붙어 있는 숫자가 Address Boundary인데요, 요 안에서만 Burst 될 수 있는 거지요. 뭐, Address boundary라는 어려운 말로 하긴 했는데, 그 숫자 만큼씩 burst로 쓴다고 보시면 되요. 그런데, 문제는 HSIZE를 보니 Word네요. 그러면 Word 4개씩이 Burst 라고 보시면 되요. Word 4개씩이면 32bit ARM에서는4byte*4=16byte= 0x10byte고요,0x30 보다 크거나 같고, 0x40보다 작은 size=2> 0x30~0x40의 범위가 되겠네요. 일단은 4 beat wrapping burst를 보면,



주소를 잘 보세요. 0x38 0x3C 다음에 0x40으로 안가고, 0x30, 0x34로 돌아왔죠. 이런 식으로 주소가 wrapping되어 그 내부에서만 burst로 사용됩니다. 요게 WRAP 4이고요, 만약에 Wrap type의 Burst를 쓰게 되면 Boudary를 넘어가지 못하고 다시 처음으로 돌아오니까, Random 주소에서부터 시작하면 곤란한 경향이 있지요. 그러니까 대부분 Boundary 처음에서부터 Wrap 크기 만큼 Burst를 사용하는 걸 권장 합니다요. 실은 WRAP type의 Burst를 사용하는 System을 보지 못해서, 어디에 유용 할 까는 저에게는 영원한 숙제라고나 할까요. 후~ 그리고, Word 단위의 Burst라면 Address의 A[1:0]=00 이어야 하는 건 아시겠지요? 마찬가지로 half word (2byte)라면 A[0] = 0 이어야겠죠. A[1:0] = 10 이라면 4byte align 아니니까, burst 안되겠죠? 뭐 그럼 INCR4는요? INCR은 무작정~ 주소 increase이고요, Burst 개수에만 관계 된답니다. 0x30, 0x34, 0x38, 0x3C 이렇게 Data를 쓰는 거지요. 아 멀리도 왔네요. 재미있지도 않은 많은 그림들을 왜 이렇게 많이 보면서 왔느냐! 우리가 배운 것은 Timing Diagram보는 법하고요, Device들의 동작원리가 여기에 다 숨어 있어요. 나중에 보면 알겠지만, 대부분의 Device의 동작원리는 여기서 벗어나지 않아요. 다~ 똑같아요 똑같아. 참고로, AHB, ASB, APB외에 AXI라는 AMBA 3.0 Spec도 나왔어요. Advanced eXtensible Interface)라고 부르는데요, AXI는 Burst 기반으로 이루어져있고, Write Response channel이 추가되어 있고요. Read / Write가 동시에 가능해요. 시작 주소만으로도 Burst Transfer가 가능하게 되었지요. 고속 동작이 가능하도록 설계된 이 Bus는 ARM11이상의 Core를 사용하는 MCU의 backbone bus로 사용되고 있어요. 사족으로 이런 AMBA System덕에 SoC할 때 IP를 갖다 끼우기만 하면 계속 SoC를 확장할 수 있는 구조로 만들 수 있겠죠!