요점정리

<https://www.so-logic.net/documents/knowledge/tutorial/Basic_HLS_Tutorial/sec2_developing_custom_ip_core_using_hls.html>

[**https://www.xilinx.com/support/documentation-navigation/design-hubs.html**](https://www.xilinx.com/support/documentation-navigation/design-hubs.html)

**Vivado & HLS 관련 모든 자료가 다 있음**

**Xilinx HLS 사용법**

* Run C Simulation -> soution1/csim 생성(debugging도 가능)
* Active Solution -> soution1/syn 생성
* C/RTL Co-simulation -> soution1/sim 생성(testbench 포함 RTL 검증 simulation)
* Export IP 실행, Configuration에서 Vendor name과 Display Name을 꼭 적어둔다.

Vivado에서 IP repository를 등록하고 IP를 불러서 사용함

* Project Setting -> IP->Repository Manager에서 D:/HLS생성 폴더이름/solution1/impl/ip

을 등록함

* IP Integrator/Create Block Design에서 IP search를 하면 HLS에서 생성한 IP가 검색되고 이를 불러들임
* Pin을 선택하고 right button를 누르고 make external port를 선택
* Auto Connection을 눌러 자동 연결을 하고 수동 연결도 필요하면 한다.
* 회로도 완성 후 Zinq Processor를 더블 클릭하여 I/O, Clock등을 설정해준다.
* Address Editor에서 메모리 크기를 설정한다.
* 메뉴에서 Validate를 실행하여 검증한 후
* Source Tap에서 Design을 선택하여 right button를 누르고 Create HDL Wrapper..를 선택, 실행함
* Synthesis > Implementation > Generate Bitstream
* Open Implementation Design을 실행하여 Placement&Routing 결과를 확인하고

Timing window에서 Setup&Hold time이 critical 한지 확인할 수가 있다. 이 때 Setup&Hold time이 마이너스 값이면 동작하지 않는다.

Netlist에서 HLS에서 생성된 블록을 선택하여 오른쪽 마우스로 색을 변경하여 Placement&Routing 결과에서 어디에 위치하는 지 확인 할 수가 있고 Project Summary에서 어느 정도의 resource를 사용했는 지 확인 가능하다.

* File > Export > Export Hardware를 실행하고 Include Bitstream을 체크하고 OK한다.
* File > launch SDK를 실행하여 SDK을 연다.
* Application Project를 실행하여 Project를 생성하는데 Finish를 누르지 말고 Next를 눌러서 Helloworld를 선택하면 Helloworld.c에 main()가 생성된다.
* Main()내에 C code를 작성한다.
* Terminal tab을 선택하고 노란색 Terminal Setting 메뉴를 선택하여 Serial 선택 board rate를 115200을 선택한다
* Xilinx Tools/Program FPGA를 선택하여 FPGA를 굽는다.
* Debug As/Launch on Hardware 를 실행한다(GDB Jtag으로 설정하는 것임) Debug mode로 들어감
* Debug를 누르면 실행이됨.

\*\* Src/lscript.ld에는 Stack Size, Heap Size를 변경할 수가 있고 기본적인 메모리 주소가 있다

\*\* <참조>PL내에 있는 Xilinx IP(예: VDMA 등)을 사용하기 위해서는 Lib아래에 있는 Include 파일을 Include 해야 된다.

ug998-vivado-intro-fpga-design-hls.pdf p35. Figure 4-4

When using HLS, any pointer access on function parameters implies either a variable or an

external memory. p40.

Option 정의: p.132

Interface Synthesis : p.85

**Coding 방법(ug902-vivado-high-level-synthesis\_2018.pdf; Ch. 3 High-Level Synthesis Coding Styles)**

1. System call 함수는 ifdef \_\_SYNTHESIS\_\_로 처리하여 합성이 안되도록 막아준다
2. Dynamic memory allocation은 고정변수로 처리한다
3. 변수이 Bit수를 알고 있을 경우 변수선언시 지정하여 준다.

typedef int12 din\_t;

typedef int24 dout\_t;

**Loop**

1. Loop count값이 정해져 있지 않고 입력을 받을 경우 pipeline으로 구현시 latency값을 알 수가 없기도 하고 풀어헤쳐서 로직으로 구현하는데에도 어려움이 있다.

이럴 경우 최대 Loop 횟수값으로 정하고 조건문을 삽입하여 동일하게 동작하도록 한다.(p316)

1. 해당 Loop를 PIPELINE으로 처리하면 하위의 모든 Loop는 unroll되어 모두 병렬, 하드웨어로 처리된다. 따라서 Loop Hierarchy가 있을 경우에는 어느 Loop를 pipeline으로 처리하느냐가 로직 count및 처리시간에 중대한 영향을 미친다.
2. Nested Loop의 경우 내부 루프를 실행하기전에 latency time이 발생하므로 내부루프까지 풀어서 하나의 루프로 만들어서 내부적으로 설계한다.
3. Sequential 루프의 경우 일반적으로는 가능한 병렬처리가 되도록 컴파일하고 구현하지만 Sequential 루프의 반복횟수가 다르기떄문에 병렬로 설계하지 않는다. 하지만 각 루프를 함수로 코딩하여 불러서 사용한다면 병렬로 구현하게 된다.
4. 종속성 루프인 경우(루프내의 실행 결과에 따라 루프릐 다음 스템을 실행할 지가 결정되는 경우;p319)는 파이프라인으로 구현될 수 없다

**Array**

1. 메모리를 많이 사용하는 경우 **Project Settings > Simulation > Linker flags**에서메모리 용량을 늘릴 수가 있다.
2. Multiple 메모리 access인 경우에는 파이프라인을 수행할 수 없다. 이 경우에는 C coding을 수정하여 파이프라인을 수행할 수가 있다. P324~325
3. 순서대로 read/write하면 FIFO로 구현한다

#pragma HLS INTERFACE ap\_fifo port=d\_i

#pragma HLS INTERFACE ap\_fifo port=d\_o

1. Array 초기화; static int coeff[8] = {-2, 8, -4, 10, 14, 10, -4, 8, -2};
2. P344

<https://www.youtube.com/watch?v=BUVbKonhc2w>

<https://www.xilinx.com/html_docs/xilinx2018_1/sdsoc_doc/tgc1517252127858.html>

<https://www.xilinx.com/html_docs/xilinx2018_1/sdsoc_doc/clu1517269835481.html>

<https://www.xilinx.com/sdsoc>

SDK 설명\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<https://www.youtube.com/watch?v=Ijrct2e7qLw>