Let us here present the first 80x80 bit architecture processor and operating system all in one:

Systolic Array Architecture - SAA

combination of performance and programability done with the

2D DATAPATH ARCHITECTURE

NO CONTROL UNIT

NO ALU

PETER-PAUL TROENDLE

US

APRIL 16 2017, PST 7:09 PM

COMPUTING SCIENCE

BELLEVUE, WA

USA

http://www.xlyns.com/Muxcodes-Programing.html

Here an architecture beyond the FPGA because it extends its ability by making:

runtime synthesis, auto routing of the elements, auto allocation and auto binding.

With this the FPGA can no longer be ignored

for the server acceleration.

This is the preliminary design document, the previous report might be easier to read, please check out:

www.xlyns.com/One.pdf/
Preamble: in Reference to Isaac Asimov

FIRST LAW ON ROBOTICS should be converted to:

“NO MECHANICAL MEANS SHOULD BE ALLOWED
TO BE USED OR EXIST IF IT CAN BE FALLING
IN HANDS OF THE UNINTELLIGENT”

SECOND LAW ON ROBOTICS:

“ALL MECHANICAL MEANS SHOULD BE
PROPORTIONAL TO THE INTELLIGENCE MEANS”

THIRD LAW (L. Wittgenstein):

“THE INTELLIGENCE IS TO FIND THE TRUTH
- CHECKING FOR COMPLETENESS
- MODELING CONSISTENTLY”

AND .. LAW 0

“A GOOD LAW IS A ZERO-LOSS GAME AND
LAWS THAT ARE NOT UNIVERSAL ARE IRRELEVANT”

THEN LAW ∞

“LAW 0 APPLIES TO ALL WITH CONSCIOUSNESS
AND CONSCIENCE”
## Title

Preamble: in Reference to Isaac Asimov

I/ NULL

II/ The Binary Complemented Number (extends the binary to include NULL)

III/ The Universal Number System - Unary

IIII/ The Constructor - placeholder

IIIIII/ Handling data – Weaver transport

IIIIIII/ Structuring data – The waterline principle

IIIIIIII/ Routing data – The collider

IIIIIIII/ Bitslice design – for collision avoidance

IIIIIIII/ Software instructions - exercise

IIIIIIII/ Software instructions - solution

Conclusion

Software implementation - Architecture

Why Azure

APPENDIX - Software instructions outline

Design Status

APPENDIX - Software example of an ethernet stack

Specifications
When there is nothing it does not exist, when there is nothing and its existence is defined by a name then it is the NULL, when there is something and its existence is defined by a name then its a variable because it can have anything, Including NULL because that is when the box is empty and “the content is expected” or undefined. What is important for the reader is to understand that when there is a NULL it indicates a fetch process is needed. When a human needs food it will engage in actions to fulfill that, similarly computers actions are to fulfill Queries. In fact a question is in some sense the same as NULL that opens to actions to get the data or answer. THIS= “” is the same as THIS? where the answer is expected, THIS=”THAT” would be its completion per ex.

What distinguishes code from data is code is data triggering actions that will return the answer filling the void.

[ CONVENTION : NULL is ]

When a wire carries a signal a second wire signals that it is busy so all other incoming from that side will be able to engage in deviation and the name of the second wire is the NULL. When a signal is disconnected it does not exist and it is not even NULL, when it is there connected it is set:

In Figure 1 we see a wire connected and NULL=1 signals that state irrespective of if any data is arriving or not, it is indicating an “empty” box ready for filling.

![Figure 1](image_url)

Note: the connected = NULL and disconnected = NULL is just a convention, could be the other way around, we just choose it because SQL that created the concept (Edgar F. Codd) has the empty column with NULL when connected and awaiting incoming data. We chose the same convention to be backwards compatible to SQL:

[ CONVENTION 0 : NULL is active low ]
Here why we use NULL: appending numbers of different sizes \((A,B)\), in effect the signals “find” their way using the NULL guide-rails, see Figure 2:

To be able to handle the NULL the binary numbers present the inherent limitation due to the lack of a 3rd symbol, one or zero requiring 1 bit, could only accommodate that using “reserved” sequences. In fact around 20% of all code seems to be dealing with the description of that concept of NULL where a length of data, its' format, type or even the start and finish tag on a web page use the NULL because it is there to say ok here is data and there is other data and anything in-between to dissociate is that “NULL” that is not even a space, see Figure 3, rather a separator:

Here we go even further and it is to consider even the For-Loop or any counter also to use the NULL because it is that marker that indicates when the counter has found its' goal. There the same misconception as for length in space exists where if one counts to 5 or N, conflicting conventions use N or N+1, so “count to 5 elements” can be 4 or 5 or even 6

current conventions: 01234 4 non exclusive, 12345 5 non exclusive, 123456 6 exclusive etc...
suggested method : 12345 NULL that is a distance of 5 “numbered” elements : - - - - - NULL
For the description of it a complementary bit is added similar to how signals in CMOS are complemented to make them more robust, in fact in ASIC every digital transistor is double because there is the N- doped size and it mirrored P+ doped size, Complementary Metal Oxide Semiconductor are the base of most digital circuits, so it uses a P transistor coupled to a N transistor to create the 1 bit transistor. Placing a few transistors in clever ways allow for the standard logic gates: AND, OR, NOT, EXOR etc.. and even local storage that are small feedback loops keeping the data and referred to as Flip Flops, that are ASIC 1 bit memory able to keep data for a clock cycle and refresh at each clock edge allowing for clock driven edge triggered digital sequential operations. In some sense that clock the “event” or NULL because it connects the Flip Flop (FF) memory to the inward signal and outside that edge when NON-NULL is simply storing it and makes synchronous clock triggered systems possible that are the base of all digital circuits running code.

For those in the 70's that remembered what was announced when all used BASIC or FORTH and C came along, well it was said to be the “language with the parenthesis” and on our left we can see some typical indentation, and it is from top to bottom executed sequentially once compiled into assembler.

When executing that same data in parallel the indentations stay the same its just from left to right then and we use that complementary bit stream so we have no “reserved” or “prohibited” bits and any bits any size go. In fact those bits that do the indentation also make the control “following the data flow so no central control unit is needed and for the first time we have code that are finding their way on their own.
II/ The Binary Complemented Number (extends the binary to include NULL)

Here how we connect data to one another:

say we have a stream with 5Ah and another with A5h (hexadecimal) see Figure 4:

now if we want to connect 5A together in one side and A5 on the other side
the separator is what we use in human language: 5A, A5 and for the data stream 010101010100101 we do not want an extra bit sequence to separate because it would “cripple” the binary numbers, per ex, if we choose F=111h and use it it would reduce the numbers in binary, see Figure 5,

because 1111 would be prohibited. Prohibitions and Exceptions are the result of poor design that also open the door to errors and in our example it is apparent that if we had 5Bh instead of 5Ah it would fail, see Figure 6 here would be several different interpretations of where the separator really is.

And that is why all digital systems use a fixed size format. This document is following the previous work where is shown how systolic arrays allow any ALU operations AND, OR, NOT, ADD, SUB, MULt, DIV and even SQRT (see bottom page of http://www.xlyns.com/Systolic-Array-1bit-ALU.html) with the feature that it can handle streams of operations that all can be done in parallel (that is what a systolic array does) and to make that possible a concept was needed to define in a stream where one data begins and where it ends. Here the suggestion to use Binary Complemented Numbers BCN. We separate 5Ah and A5h : 5A,A5 this way, see Figure 7:

that way toggling the areas where the data is complemented from areas where it is not make the data stream look more like a streamed “list” of data, allowing for any sizes, see Figure 8:
Grammar for Tree Structure

[ CONVENTION 1 ]
1 single control bit within data is in 2 forms: (or up and ) or dow see Figure 9:

[ CONVENTION 2 ]
2 single control bits that are in contradiction, 2 forms: . dot or , separator see Figure 10:

[ CONVENTION 3 ]
data used as name/descriptor or Alias, preceeds an indentation (see Figure 11 top):
data used as receipient content/digits or Shape finished it (see Figure 11 bottom):

Figure 12 shows how a falling edge or rising distinguishes both forms in these examples:

This allows JSON, XML to pass objects in that form to the FPGA that now has that grammar
III/ The Universal Number System - Unary

Very bluntly now we have data in a Tree Structure (see previous chapter putting it in JSON/XML/BCN allowing Tables, lists, objects and any data) the binary numbers become a real burden in parallel computation just to go from one point to another the binary number forces a fixed format and overheads, see Figure 13: in this example each “step” would need 8x4 bit gates to check if it is arrived at “1000” and they have to be “placed” on that path if one wants to count.

So here the unary was considered that just counts the steps using the “unary” units to move, see Figure 14:

[ CONVENTION 4 ]

Length is expressed by unit length – one can use it to select nodes of a tree with no bother if the amounts of sub element are of different sizes, see Figure 15:
CONVENTION 5] the unary numbers start with 0
- most binary numbers start with MSB = 1 see Figure 16:
and zero is left as empty, the calculations handle “empty” as 0 of value
- the ASCII code or other codes that are non numeric
would need to start with an added 1

The unary are convenient for addressing and this way one can distinguish such an address from data

CONVENTION 6] in trigger mode the NULL applies to the whole group selected using the waterline

In Figure 17 it is called by a link C.

\[ \{ C.B : C.A \} = \{ “X” : “Y” \} \]
Note: A, B, C, X, Y are normally in binary
that can be also in unary, see Figure 18:
The unary allows to move along tree nodes describing the path and we can create a decimal tree, a binary tree or hybrid and “counting” would be “traveling” through, hence conversions are no longer an issue.

In Figure 19 a same distance of 5 would have different unary values on different number systems, even irregular ones, yet it will still be \(5 = 5^{th}\) node and that number can be in any system expressed as a counting tree – binary is just one case, we wont restrict to that special case.

[CONVENTION 7] The depth is expressed by unit depth

Several NULL climb up the branch see Figure 20
The reason we extend the numeration system is to make use of the extended degrees of freedom that are given by the 2D Transport, when in normal systems they are constricted to a single wire that only go from one point to the other. In mechanics advanced mobility is achieved with usage of the degrees of freedom on he joints, that multiply the span, similarly the lever can exert more power if it is placed appropriately.

The unary can be compared to “small telescopic” arms that can go from anywhere to anywhere else and accommodate any size of data in any structure – that is the reason we choose it to allow universal machines that can run all different codes that are out there and the beauty is it is build on a scalable architecture meaning no limits anymore to anything. At this point I hope this makes it clear enough why it is a priority to build this and here (see Figure 21 and further) a suggested plan:

\[
\begin{align*}
Y &= \text{data in binary} \\
0001001110 &+ 1 \\
0001001111 &+ 1 \\
0001001111 &+ 000000 \\
0001001111 &+ 1000000 \\
\end{align*}
\]

\[1 \Rightarrow Y[0] \]

\[10000 \Rightarrow Y[4:0] \]

\[0 \Rightarrow Y[7:6] \]

\[0 \Rightarrow Y[15:12] \]

\[etc.. \]

\[K \text{ returns the weight bin test result} \]

\[A,B \text{ means that B does not wait on A: “A or B”} \]

\[A,B \text{ means that B waits on A: “A and then B”} \]

\[a \text{ list is applied concurrently using a sequence is related to a trigger} \]
One very important feature of the NULL is its character as "envelope" and when the streams go in the transport they could collide so it is build that they would avoid other streams by checks if there are foreign "envelopes" and the NULL is used as guide-rail for others to avoid wrecking on other streams.

III/ The Constructor - placeholder

The "envelope" of data when left empty is then what will "request" a fetch for the data to fill it.

For the case there is not the luxury of "named" data and it is raw one can use placeholders

\{Link : " . . . . "\} 

and when they hit the raw data they "scoop" the bits, each thread gets 1 bit, so we call that the placeholders that search for the data in \{“Link”: “raw data”\}

For the Link. shown in Figure 22 it is a NULL that uses "Link" to search for the data

![Figure 22](image)

and when it finds \{“Link”: “data here”\} it then fills the data.

![Figure 23](image)

[ CONVENTION 8 ] Placeholders are NULLS instead of data.
III/ Handling data – Weaver transport

The concept used in the transport that allows us to get rid of the ALU and Control unit is to have a self driven data that carries a tag that will search for its counterpart, so if one tell it to get “ID” it will and if then one tell it to get “NEXTID” it is going there resovling the link ID.NEXTID. and the . Is used to push the tag (“ID” or “NEXTID”) in the transport. Basically it works with identifying, searching that uses only one gate: XNOR! when A matches B = 1 and we use the . To define a group so if all elements match all bits of “ID.” it “stills” the NULL and the following one is activated, hence the matching signal has to cascade through the whole length of the link and when found it unlocks the vertical release, see Figure 24 with ID = 10 and NEXTID = 11:

on the right hand side the unresolved ones are then fed back in the carousel memory while the ones with a solution go one level higher to the higher carousel.

[ CONVENTION 9 ] The matched descriptors for a query add the found data to its stream
CONVENTION 10 The NULL finishes its action when hitting the waterlines end. See Figure 25:

Here some, see Figure 26: note: ↑↓, . are in BCN ↑ is passed as 01 -.
↓ is passed as 10 -.

descriptor names are under the line with falling edge digits of data are above the line with rising edge these tags from the waterline allow to have the weaver skip the data and focus on the names.

CONVENTION 11 Foreign data or data leaving the memory converts its tree position in unary “reading the waterline” it allows for the return path and we name it variable “UN”.

Just to say here is shown 12 elements structured and in normal flows they would use 12x64bits = 768 wires so it is clear that this is now far reduced and is feasible now. The process is also unpenetrable because processors are physically orthogonal (see weaver) and subsequent “deletions” or “adds” of data cannot take place.

The idea is to let things flourish naturally – without deleting the previous things or renewing links, basically one can get a copy from anything and then filter, process or handle and that instance will not touch the original data and that is the end of the “braking” computing where “hidden” data is added. Here to add data a port access to the prompt (2 lines BCN) is given.

CONVENTION 12 The user prompt is given the upper lines of the Transport. This allows new processes to start instantly and real time is possible.
When removing sequential data from memory it is truncated because that is what sequential does, it cuts data regardless of its content so feeding it to the transport or routing the results back is needing an interface doing that. It is really simple as “stacks” yet hard to visualize because it is as if the data was subjected to “artificial gravity”

[CONVENTION 13] Data with no authorization or Null-rail are considered as free space where the other data sacks up.

In Figure 27 the collider gets the remainder from the carousel (green) and if the whole carousel line is free it adds the following data left in the collider upper part or additional inwards stream from the memory (blue).

![Figure 27](image)

Basically we here use the carousel getting new data on it that it appends so we can pass from one stage to another.

Note: for larger data it is blocking, the carousel once reaching the “free” is filled, blocking. Because memory is sequential hence blocks downstream. Solution: infinite large memory = non blocking.
III/ Bitslice design – for collision avoidance

This is what it looks like sorry for the whole theory it is in fact real simple to build (see chapter I in the beginning) and use (see next chapter)

In Figure 28 we see in red the relevant signals are made to turn and in blue they can carry on, in Figure 29 is shown how the slice is done showing that it is likely to be reproduced on a large scale.
Here incoming 2 stream from left and 1 from top, see Figure 30. The space between A0 and B0 is removed in the collide stage. This is how the data is added to queries.

In Figure 31 the situation of parallel queries is shown and we see the nullwave broadening allowing the data to be also available to the downstream queries. Without this queries would be forced to have placeholders and the principle here is to let the operations unfold the needs of size not the user.
Software instructions - exercise

Let's see:

UN.ID

UN.ID.NN.OO } 3 parallel commands enter on the left, see Figure 31

UN.ID.AA.01...

On this JSON:

unary of the root position

(=namespace)

Figure 31
Software instructions - solution

Let's see:

UN.ID
UN.ID.NN.OO  \{ 3 parallel commands
UN.ID.AA.01,...  \}

On this JSON:

\[
\begin{align*}
\text{UN} & \text{ ID} \\
\text{UN} & \text{ID} \downarrow \\
\text{UN} & \text{ID} \downarrow \\
\text{UN} & \text{ID} \downarrow \\
\text{UN} & \text{ID} \downarrow \\
\text{UN} & \text{ID} \downarrow \\
\text{UN} & \text{ID} \downarrow \\
\text{UN} & \text{ID} \downarrow \\
\text{UN} & \text{ID} \downarrow \\
\text{UN} & \text{ID} \downarrow \\
\text{UN} & \text{ID} \downarrow \\
\text{UN} & \text{ID} \downarrow \\
\text{UN} & \text{ID} \downarrow \\
\text{UN} & \text{ID} \downarrow \\
\text{UN} & \text{ID} \downarrow \\
\text{UN} & \text{ID} \downarrow \\
\text{UN} & \text{ID} \downarrow \\
\text{UN} & \text{ID} \downarrow \\
\text{UN} & \text{ID} \downarrow \\
\text{UN} & \text{ID} \downarrow \\
\text{UN} & \text{ID} \downarrow \\
\text{UN} & \text{ID} \downarrow \\
\text{UN} & \text{ID} \downarrow \\
\text{UN} & \text{ID} \downarrow \\
\text{UN} & \text{ID} \downarrow \\
\end{align*}
\]

unary of the root position  

(=namespace)

Similar to the top query UN.ID the 2 others are done and each query gets back the result that is then appended in completion, see Figure 32
Mr. Pintaske Jürgen that worked for RCA on the COSMA 1802 architecture (a variant of the 0x86 that resulted in the 68x DSP architectures) asked us to provide a demo for FORTH (see previous documents) and we were only able to give 4 bits wide due to FPGA wirings and his goal was 16 bits wide and would have taken $16 \times 16 = 256$ slices that were too full (bitsliced add, mult, sub, div, plus all logic boolean). Additionally was his pertinent question if we have done the shift left and shift right, and that became a burden because our thought of having a 4 x 4 with Rollover to 4 blocks would imply having the carry taken care of, now the shifts was complicating the features at such extend, so the idea of making an all Systolic Array Architecture emerged and here with IN.AA.01... we proven the point of being agile enough even to define a group within bits and extract that group accurately defining the start and finish precisely: “start on the 2nd bit unary 01, take 2nd and 3rd ..”

Thanks to J.Pintaske for his request and mentoring in addition to N. Beaumont and Microsoft.
Software implementation - Architecture

First the user calls the program on C1, see Figure 33

Then parsing the wireshark TCP raw data on C1 (RAM → OAM Organised Access Memory), and links to be resolved are collected in C2, see Figure 34
When the links are resolved first on itself C1, see Figure 35, if incomplete on M0 it fills the archive C-1

The C1 OAM then merges with the resolved links C2 into C12, see Figure 36
Then the LUT function footprints are loaded into C4, see Figure 37.

The gathered data is processed in T5 using the LUT and the result is returned to M0, see Figure 38.

The prompt can call the TCP packet prepared to send and a trigger on the response to capture the ACK SEQ and increment and send back, then receiving the HTTP back and appending the FPGA data and send back to AZURE. To have the FPGA directly accessible via web page or database, makes real time processing on large scale feasible.
Server programming for all

A 100$ FPGA can do 2808 bits width @ 125Mhz and if we can have it to the weaver it would yield each clock many complex operations in the least possible time and it does have 400 pins and inside 700Gbs any access would be resolved concurrently, The FPGA would compete with server real time abilities such as for Games etc.. so each user could send the game data real time in form of JSON or XML and resolve the calculus (in the games the figures running against each other would bump really).

The FPGA makes it only possible to run one logic defined program, that runs fast but the setup time that uses synthesis and a lot of conception time is blocking the normal use of it. So here a way to bypass the synthesis by self routing code elements, it is now possible to have any program that can be defined by logic and most programs can be converted very simply to that, all needed is the equivalent boolean logic operation for each assembly code or other code where one can simply ignore all the parts of them that deal with control because here its greatly simplified and uses the NISC no instruction set architecture
APPENDIX - Software instructions outline

1. **constructor**
   ```
   { "URG": {"."} }
   ```

2. **simple access**
   ```
   { "data": { ip.datagram } }
   ```

3. **expression search**
   ```
   { "user data": {""}, "event":{foreign.http.datagram."x0="} }
   ```

4. **functions**
   ```
   { "CRC": conn.phy.Crc.{"NewAlias"} }
   ```

5. **conditional fetch**
   ```
   { "ACKNum":{""}, "event": foreign.tcp.datagram.(ACK)?SEQNum+1}
   ```

**Design Status**

**Engineering Resources Used**

We use here Vivado 2016.4 so there the Block Editor, Simulation works and also in 64bit without the issues
Laptop with USB plug, USB to USB micro cable, Power supply 12Vdc 3A, Vivado - xilinx.com free download,
Nexys-Video Board, Digilent Inc, WA 320$ academic pricing, the Nexys-Video board is a complete, ready to
use digital circuit development platform based on the latest Artix-7™ Field Programmable Gate Array from
Xilinx. With its large, high-capacity FPGA XC7A200T-1SBG484C, 33’650 slices with 6 i/p LUT 8 Flip Flops
and 13Mb Block RAM, the whole system running at 400MHz, 2.5nS from memory read to memory write

**Design for Test**

Due to the synthesis methods being not open source the simulation tool vendors may have heuristic
approaches, we measured that they had inaccurate edges, so here we use an inner oscilloscope RAM
oversampling test point over time allowing for real time monitoring of internal processes for testability.

**Devices under Test (so far 128bits – goal 2048)**: M0 UART, M1 TCP, C0, C1, C2, C3, C4, Collider, Weaver
APPENDIX - Software example of an ethernet stack

{"conn":
{"16bitcascadingsum": { "................ : + }},
{"arp":
{"datagram": [
{"Hardware Type": "................" },
{"Protocol Type": "................" },
{"Hardware address length": "........" },
{"Protocol address length": "........" },
{"Operation code": "........" },
{"Source hardware address": "00000000000000000000000099999999" },
{"Source protocol address": ".........................." },
{"Target hardware address": "," },
{"Target protocol address": ".........................." },
"comment": "}
}],
{"http":
{"datagram": [
{"deleted"},
{"user data": {""}, "event":{foreign.http.datagram."x0="}}
],
"comment": "}
"POST /parse_simple.php HTTP/1.1 Accept: text/html, application/xhtml+xml, image/jxr, */* Referer: http://xlyns.azurewebsites.net/FPGA-feeder0.html Accept-Language: en-US User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.79 Safari/537.36 Edge/14.14393 Content-Type: application/x-www-form-urlencoded Accept-Encoding: gzip, deflate Host: xlyns.azurewebsites.net Content-Length: 7 Connection: Keep-Alive Cache-Control: no-cache Cookie: ARRAffinity=9b78f19010cde6d04cb8246999dcbb4f2744e93b3b758d64c2432bf18d4d16e78 x0="}
],
{"tcp":
{"datagram": [
{"Source Port": {"................"},
"event": uart.tcp.datagram.[ Source Port ]},
{"Destination Port": {"................"},
"event": uart.tcp.datagram.[ Destination Port ]},
{"Sequence Number": ".........................." },
{"event": foreign.tcp.datagram.(ACK)?[ACK Number]+1 },
{"Acknowledgment Number": ".........................." },
{"event": foreign.tcp.datagram.(ACK)?[SEQ Number]+1 },
{"Data Offset": {"0101"} }, // 5 32bits words are in the header
{"Reserved": {"000000"} },
{"URG": {"."} }
}
Appendix Programming example: Ethernet Stack (continued)

{ "ACK": { ".”} , "event": foreign.tcp.datagram.(ACK)? "1" : "0"},
{ "PSH": { "0"} },
{ "RST": { "0"} },
{ "SYN": { ".”} , "event": "(SYN)?[Sequence Number]=[Sequence Number]+1"},
{ "FIN": { "0"} },            // the Time to live closes anyway
{ "Window": { "0100000000000000"} }, // 16384
{ "Checksum": { "0000000000000000"}, "event": "Checksum()" },
{ "Urgent Pointer": { "0000000000000000"} },
{ "Option": { "000000000000000000000000"} },
{ "Padding": { "00000000"} },
{ "data": http.datagram }
}),
{"Checksum": !(16bitcascadingsum(16bitcascadingsum(datagram.ip.[[ Protocol ],[ Source Address ], [ Destination Address ] , ByteLength( datagram.tcp ), datagram.tcp ] ))),
{ "comment": "" // SPECIFICATION RFC 793 TCP Datagram
// TCP Header Format
// 0                   1                   2                   3
// 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
// ++++++++++++++++++++++++++++++++---------------------------+++++++
// |          Source Port          |       Destination Port        |
// +-------------------------------+-------------------------------+
// |                        Sequence Number                        |
// +-------------------------------------------------------------------+
// |                    Acknowledgment Number                      |
// +-------------------------------------------------------------------+
// |  Data |           |U|A|P|R|S|F|                               |
// | Offset| Reserved |R|C|S|Y|I|            Window             |
// |       |           |G|K|H|T|N|N|                               |
// +-------------------------------+-------------------------------+
// |                   Checksum            |         Urgent Pointer        |
// +-------------------------------+-------------------------------+
// |                    Options                    |    Padding    |
// +-------------------------------+-------------------------------+
// |                             data                              |
// +-------------------------------+-------------------------------+
//                            TCP Header Format "}
Appendix Programming example: Ethernet Stack  (continued)

```json
{
  "ip": {
    "datagram": {
      "Version": "4",
      "IHL": "",
      "Type of Service": "",
      "Total Length": "",
      "Identification": "",
      "Flags": "",
      "Fragment Offset": "",
      "Time to Live": "",
      "Protocol": "00000110",
      "Header Checksum": "",
      "Source Address": "",
      "Destination Address": "",
      "Option": "",
      "Padding": "",
      "data": tcp.datagram
    },
    "Checksum": !(16bitcascadingsum(16bitcascadingsum(datagram.

      [ Version, IHL, Type of Service, Total Length, Identification,
      Flags, Fragment Offset, Time To Live, Protocol,
      Source Address, Destination Address ]))",
    "comment": "// SPECIFICATION RFC 791 IP Datagram
// 3.1. Internet Header Format
// A summary of the contents of the internet header follows:
//          0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
//    +----------+-----------+-----------+-----------+-
//    |Version| IHL |Type of Service| Total Length |
//    +----------+-----------+-----------+-----------+-
//    |Identification|Flags|Fragment Offset|
//    +----------+-----------+-----------+-----------+-
//    |Time to Live|Protocol|Header Checksum|
//    +----------+-----------+-----------+-----------+-
//    |Source Address|
//    +----------+-----------+-----------+-----------+-
//    |Destination Address|
//    +----------+-----------+-----------+-----------+-
//    |Options|Padding|
//    +----------+-----------+-----------+-----------+-
// Example Internet Datagram Header
// Figure 4.
"}
```
The current program shown above is describing far more operations than what usually 30 to 40 files in C or other current programming language do that are not even parallel and here the “form” requests data with as many threads in parallel as the width allows so its real fast and practically real time.
**Features and Specifications**

### Specifications

The state of the art allows a clear conscience and to enjoy it

<table>
<thead>
<tr>
<th>RISC/MIPS</th>
<th>FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use XML JSON</td>
<td>XML and JSON user BCN interface</td>
</tr>
<tr>
<td>Float, Long, Double</td>
<td>User definition : placeholders for any format</td>
</tr>
<tr>
<td>Fixed size 8/16/32/64 bits</td>
<td>Scalable 1 to n bits in 1 to m streams</td>
</tr>
<tr>
<td>ALU and Control unit</td>
<td>XML/JSON form, Event triggers</td>
</tr>
<tr>
<td>Incompatible programming languages</td>
<td>Universal, can run any assembler</td>
</tr>
<tr>
<td>Sequential processes are easy to “tap”</td>
<td>Concurrent unpermeable processes</td>
</tr>
<tr>
<td>Binary</td>
<td>Binary or User defined numeration (unary)</td>
</tr>
<tr>
<td>Non deterministic duration</td>
<td>Real time possible for RAM &gt; 2048</td>
</tr>
<tr>
<td>Chipscope/Oscilloscope/Test</td>
<td>100% coverage test with intern oscilloscope RAM</td>
</tr>
<tr>
<td>80% of the code dealing with control issues</td>
<td>Code auto routing, binding and allocation</td>
</tr>
<tr>
<td>Sequential</td>
<td>Parallel</td>
</tr>
</tbody>
</table>

2017 All rights reserved by the author