CONFIGURATION BASED ARCHITECTURE FOR HIGH SPEED AND GENERAL PURPOSE PROTOCOL PROCESSING

Dake Liu, Ulf Nordqvist, and Christer Svensson
Dept. of Physics and measurement, Linköping University,
58183, Linköping, Sweden, Phone +46 13 28 {1256, or 5816, or 1223}
Email: {dake, ulfnor, christer}@ifm.liu.se

Abstract - A novel configuration based general purpose protocol processor is proposed. It can perform much faster protocol processing compared to general purpose processors. As it is configuration based, different protocols can be configured for different applications. The configuration makes compatibility possible, it also makes very fast protocol processing on the fly. The proposed architecture can be used as a platform or an accelerator for network based applications.

Key words: Protocol processor, configurable logic, Parallel Processing, SoC.

BACKGROUND

Networking has been developing very fast. More and more protocols are emerging for different applications. Higher process performances are requested by applications. Requirements could be recognized as:

1. Multiple ports and multiple Gigabits per second real time framing and de-framing.
2. To pre-process as much protocol jobs as possible before memory accesses.
3. A general, simple, fast, and flexible architecture for different kinds of protocols.
4. A built in protocol learning and automatic configuration capability.
5. Low power, high speed, and memory (size and access) efficient architecture.

Two kinds of protocol processors are available on the market now, one is a specific single protocol limited ASIC (we call it SPASIC in this paper), another is the processor based general-purpose CPU (we call it GPCPU in this paper).

None of them can fit the requirements for future computer communications. The first one, SPASIC, is used only for one protocol or a few specific protocols included in the design. Naturally, it does not support future protocols. The second one, GPCPU, can not work at very high speed because of the general architecture. As a redundant and speed limited architecture, it is not the best for a relatively stable and control-extensive flow. From another point of view, the protocol processor must be compacted because it is often used as a pre-processor and as a small part in a certain kind of application. Therefore, the redundant architecture is not suitable for embedded or integrated solutions.

Most solutions available now use a specific circuit to process the protocol flow, and use a GPCPU for switching, routing, and other applications. Because of the limited SPASIC architecture, future flexibility is limited. For multiple applications, more SPASIC cores are integrated to cover more protocols. This makes the system redundant.

We need to recognize the protocol of the in coming package and then configure the processor to fit the protocol because the system might be used in the variable environments. Therefore, a new architecture is strongly requested, which is as fast as a SPASIC, as flexible as a GPCPU, and as simple as possible.

PROBLEMS CAN BE SOLVED BY OUR SOLUTION

The system proposed is a new architecture for control-extensive processes, for example, protocol processing. One example is to take the data package from Manchester decoder as inputs and perform fast pre-process for different level of protocols, for example, from Ethernet to IP and even up to TCP on the fly.

We can solve all problems mentioned above by introducing the *super pipeline serial processor SPSP*. It executes the protocol processing based on a booted and predefined configuration. Since the control is based on the configuration instead of software programs, SPSP can process protocols in real speed, e.g. Gbit Ethernet. After booting, the configuration HW can be shut down, which gives possibilities of low power. Following this way, the application, for example, IP Telephone, or IP switching can be separated from the protocol framing and de-framing. The advantages are:

1. Framing and de-framing is performed in a separate core, it acts as a platform or an accelerator and makes more application integration possible.
2. Separated SPSP as stand-alone machines working at high speed with a standard implementation.
3. All functional blocks inside SPSP are self-contained and configured, therefore the adaptation to long-term unpredictable future is possible.

4. The protocol can be recognized by this solution and a correct configuration can be booted to SPSP after the recognition process. We define this feature as a self-learning and self-adaptation for any product used for different environment (home RF for example).

The architecture executes protocol processing based on both pre-configured setting and a real time control program. The pre-configured setting processes the protocol in every cycle inside each field of a data frame. The real time control program only works on the higher level such as branch decisions, macro selections, and job hand over. Thus, the processing speed can be much higher because there is no program (which is slow in principle) involved in sub level processing. By planning the configuration, the architecture can supply as good flexibility as a GPCPU gives.

APPLICATION OVER VIEW

The motivation is to make a platform for all possible network applications. Part of the possible applications supported by the platform can be listed:

1. Fast framing, de-framing for the Internet switching: G-bits Ethernet source, and destination address extraction, fast IP DA and SA extraction etc.
2. Predict the memory allocation: relax memory traffic, payload reordering, etc.
3. Fast queue and priority check for the real time network applications.
4. For certain applications the products recognize the protocol of the coming data, and boot the protocol configuration after learning.
5. The user can boot different protocols for different applications.
6. For fast prototyping or SoC integration.

ARCHITECTURE

We introduce a new architecture that can work towards the physical limits of CMOS [3]. It can be implemented using conventional ASIC design flow, and can be configured by program to suit different kinds of protocol applications. The proposed architecture is divided into two parts. The first, which is the key part namely Super Pipeline Serial Processor (SPSP). Serial here does not mean bit serial, it is a byte or word based serial architecture. The second part is a normal micro controller, μC, it supports the SPSP configuration, the interface between SPSP and the application, and the real time high level job control. The SPSP can work much faster than the micro controller can.

The proposed architecture executes the protocol process based on both programs and pre-set configurations. The program only controls macro jobs, which are based on the frame rate instead of the byte rate. The pre-set configuration controls real time protocol processing at high speed with relatively fixed control and working mode. Therefore, the program control induced speed limit is completely eliminated.

The proposed architecture is configured for a specific protocol before the protocol process. The configuration is performed by writing coefficients and control codes into control registers in a Function Page, FP. All Function Pages are scheduled in the order, in which the protocol is processed in sequence.

For implementation convenience, data coming into every functional page is pipelined. Functional pages are connected one by one following the job schedule. Each FP manages its process in its own sub field. For example, the FP for CRC manages only the CRC check on the fly. Another example, the FP for header matching only matches the protocol header for its synchronization.

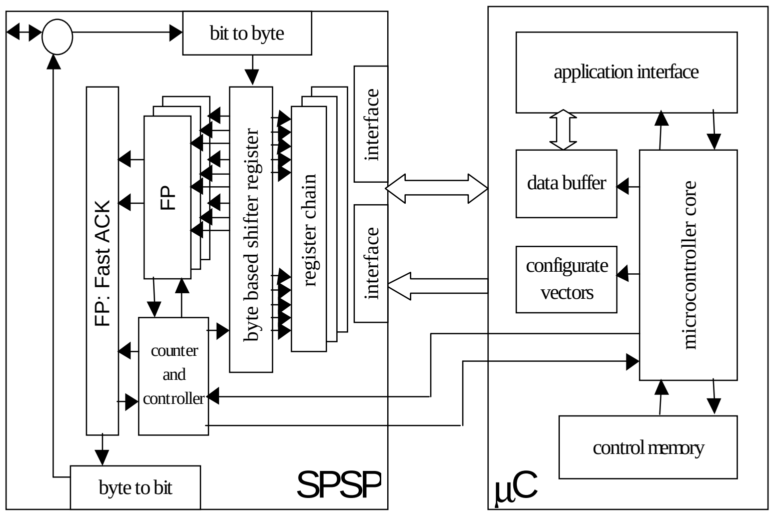

Fig. 1. Systemblock diagram

The System block diagram is given in Fig. 1. The left part is SPSP and the right part is μC for configuration, applications and for supporting applications. Different protocols can be executed according to the configuration given by μC. The μC performs the service support. Which is divided into three parts. The first part is booting, including the boot of configurations for all FP and programs in the counter and controller. The second part is SPSP monitoring, including check SPSP executing status, receive and transmit payload data, and send interactive control. The third part is to coordinate SPSP with the application hardware. The configuration is performed during the power on boot. When the protocol of the incoming data is unknown, the booting is performed for the protocol recognition first and secondly, the normal configuration according to the result of the recognition is booted. The SPSP top level architecture is given in Fig. 2. Following functions will be allocated as FP's in the SPSP given in the above figure:

1. **Matching**: It finds the synchronization point by recognizing the preamble.
2. **Error checking**: Check errors according to the coding of the protocol.
3. **The field extraction**: It extracts fields and accelerates processes further.
4. **Level hierarchy transparent process**: The HW can make levels of network hierarchies transparent. The upper level payload can be extracted.
5. **Payload management**: To measure the length of the payload and to validate the correctness of the data. Then allocate the data into a suitable position.
6. **Application interfacing**: Before data allocation, check the application, find the possibility to send data to the application on the fly.
7. **Fast acknowledgement**: The acknowledgement can be compiled in an easy and fast way according to extracted fields.

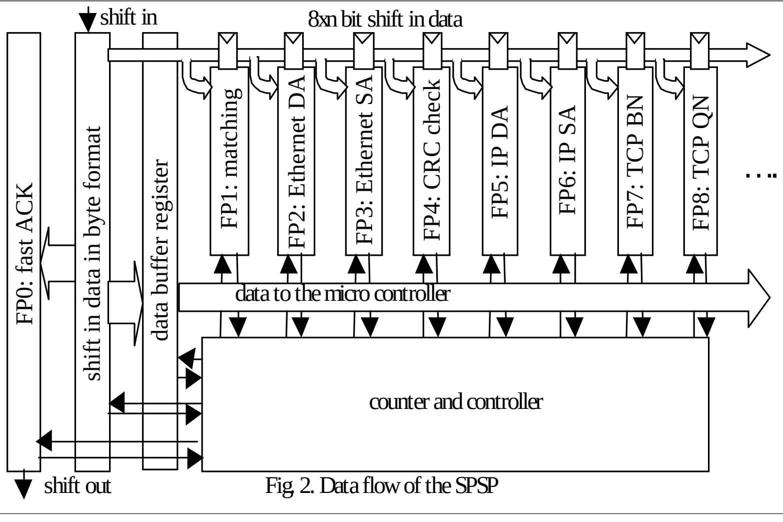

Fig 2. Data flow of the SPSP

Fast ACK as an important function is performed on the fly in SPSP. Necessary messages such as DA and SA are kept for building up the fast

ACK. The FP for ACK is allocated in between the shift-in and shift-out. The fast ACK packet can get TCP ACK, IP address and LAN address, for example Ethernet address from the buffer.

The data flow is given in Fig. 2. The data coming from the physical level has been converted to byte level format and data rate has been one eighth of the bit rate. Control signals (single pins) are handover start-finish strobes from the counter and controller. Control signals coming to the counter and controller gives timing status. Shift in and out are 8 bits input-output data of SPSP. Other width of data busses can be configured.

Functional Pages

Simple FP implementation can be done by custom design. Complicated FP will be implemented using synthesis. The flags are outputs from the sythesised logic using the configuration, the incoming data, and the control conditions as inputs. As an example, the matching unit use configuration registers to save the header pattern. When the shifted input data matches the pattern at a certain clock, a matching flag is given as Y_match = & (data, configuration_register).

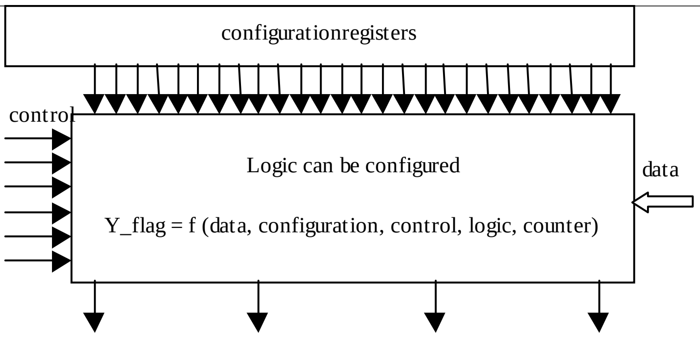

Fig. 3. FP structure

The active period of a FP is decided by its function. Most FP’s are active only part of the time. Some FP’s are active all the time during a frame process, e.g. the CRC check.

Counter and Controller

The counter and controller is a counter based state machine (FSM) adapted by the configuration. A complete configuration will be written into a register file. Each one or few lines in the register file are configured for the control of a FP.

There are two levels of controls performed in the ”counter and controller”. The upper level control is specified as the handover process. The lower level control supports only the counting status. The upper level control is a kind of interactive control. The lower level control is not interactive because the FP uses the status as a control reference without giving feedback. The super pipeline is scheduled inside each FP. The control of the super pipeline is given by the lower level control from the ”counter and controller”. Status of the state machine is configured according to the recognized protocol. A (group of) control vector for a specific FP is selected (addressed) by the counter. Therefore, the control procedure is scheduled following the configuration. The super pipeline data path performs the protocol jobs in N+Δ cycles. Here N is the number of bytes (or words, according to which protocol is used) and Δ is the number of cycles used for hand over one job from one FP to another FP.

The control is scheduled in the following way:

1. Start a FP
2. Let the FP run itself and
3. Monitoring flags coming from all active FP’s.
4. Make new control decision according to flags.
5. Monitor the control interface between the micro controller and the SPSP.
6. Change the control procedure if the micro controller gives a new request.
7. Inform the micro controller to access the available data.
8. Responde to the micro controller to accept data.
9. Send the accepted data to a FP responsible for the acknowlegement.

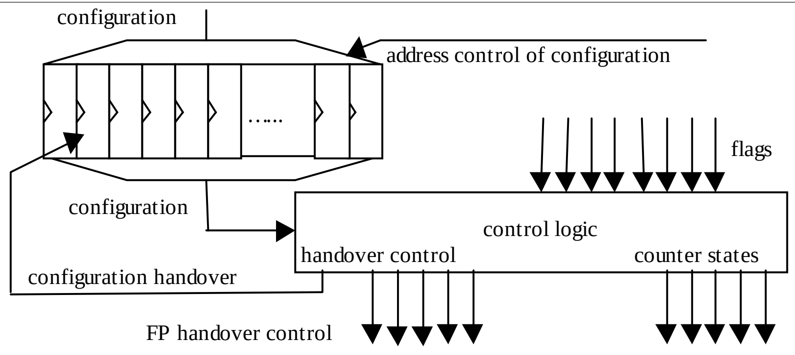

Fig. 4. Counter and controller

CONCLUSION:

We have described a configuration based SPSP architecture as a platform for network applications. The architecture implements the infrastructure of an accelarator which gives the necessary framing and de-framing, and fast acknowledgement. Most protocol processes can be supported by SPSP architecture because of the flexible configuration. The configuration based architecture can also support protocol recognition based on predefined protocol preambles. As SPSP is a specific architecture for protocol processes, it can accelarate protocol processing on the fly for high speed applications.

ACKNOWLEDGEMENT:

Authers thanks to the useful discussions with Dr. Kenny Ranerup Switchcore, Sweden. The research is supported by the Center for Industrial Technology at Linköping University (CENIIT) and Exellence Center in Computer Science and Systems Engineering in Linköping (ECSEL).

REFERENCES:

[1]. Andrew S. Taanenbaum, Computer Networks, 3rd Edition, Printice Hall PRT, ISBN 0-13-349945-6, 1996.

[2]. Jayant Kadambi et al, Gigabit Ethernet, Printice Hall PRT, ISBN 0-13-913286-4, 1998.

[3]. Anders Edman, and Björn Rudberg, SDH 10Gb/s regenerator frame in 0.6μm CMOS, 97 IEEE ISSCC, pp 156-157, 1997.