IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 9, 2019, accepted February 24, 2019, date of publication March 5, 2019, date of current version March 29, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2903126

Towards Designing Asynchronous Microprocessors: From Specification to Tape-Out

ZAHEER TABASSAM[1], SYED RAMEEZ NAQVI[1], TALLHA AKRAM[1], MUSAED ALHUSSEIN[2], KHURSHEED AURANGZEB[2], AND SAJJAD ALI HAIDER[1]

[1]Department of Electrical and Computer Engineering, COMSATS University Islamabad at Wah, Wah Cantonment 47040, Pakistan
[2]Computer Engineering Department, College of Computer and Information Sciences, King Saud University, Riyadh 11543, Saudi Arabia

Corresponding author: Syed Rameez Naqvi (rameeznaqvi@ciitwah.edu.pk)

The authors extend their appreciation to the Deanship of Scientific Research at King Saud University for funding this work through research group NO (RG-1438-034). This work is also supported in part by the Pakistan Science Foundation under Grant PSF/Res/P-CIIT/Engg (159).

ABSTRACT Proceeding miniaturization in the VLSI circuits continues to pose challenges to the conventionally used synchronous design style in microprocessors. These include the distribution of clock in the GHz range, robustness to delay variations, reduction in electromagnetic interference, and energy conservation, to name a few. The asynchronous logic has been known for its ability to address the aforementioned challenges by means of the closed-loop handshake protocols, instead of notorious clock signals. Because of these advantages, there have been numerous attempts on building general and special purpose microprocessors during the last three decades. Still, however, the number of asynchronous processors commercially available is scarce, mainly due to an insufficient electronic design and automation tools support, an ambiguous design flow and testing mechanisms for asynchronous logic and, most importantly, absence of a forum to look for relevant works, explaining the design steps and tools for such microprocessors. This paper is intended to bridge this gap by 1) reviewing the design principles of asynchronous logic, including classification, signaling conventions, and pipelining approaches; 2) presenting the complete design flow and available electronic design and automation tools; 3) developing an encyclopedia of various general and special purpose microprocessors proposed by far; and 4) presenting an evaluation of those works in terms of area on the die and performance metrics. This paper will also serve as guidelines for the asynchronous microprocessor design and implementation in all phases from specification to tape-out.

INDEX TERMS Asynchronous logic, electronic design and automation, microprocessor.

I. INTRODUCTION

While reduction in feature sizes has led the digital circuits to operate at increased clock-rates, the synchronous designs, on the other hand, face certain challenges that are difficult to overcome in the deep submicron era [1]. These include chip wide clock distribution, and susceptibility to delay variations. The former may be addressed by means of a balanced clock tree with a sufficiently low skew, however, the strong clock drivers will still pose a threat to energy requirements [2]. The asynchronous logic, which relies upon closed-loop handshakes for communication between components, naturally eliminates the need for a clock, and at the same time provides an inherent ability to adapt to uncertainties and even dynamic changes of timing parameters. Lower power dissipation, reduced electromagnetic emission, higher operating speed, and better modularity are among a few other traits associated with asynchronous logic designs [3], [4].

In spite of the advantages that asynchronous logic enjoys over its synchronous counterpart, it never flourished, and failed to catch industries' attention. The primary reason behind this predicament is insufficiently mature electronic design and automation (EDA) tools support [5]. For the same reason, the principles of asynchronous logic, the existing asynchronous systems – be them in the industry or in academics, and their design flow along with the EDA support,

The associate editor coordinating the review of this manuscript and approving it for publication was Songwen Pei.

are usually misunderstood and more often overlooked. Same is the case with asynchronous microprocessors, which have been developed during the past three decades using various design flow and tools, but neither had they managed to be among the processors of eminence, nor could they define the standard design flow for asynchronous systems.

With proceeding miniaturization, and consequently growing number of functional units on a single chip, however, the asynchronous logic once again is receiving attention of the research community [6]. We believe there is a need to comprehensively present principles of asynchronous logic, its standard design flow and available EDA support, followed by a thorough evaluation of various general and special purpose microprocessors existing in literature. This is the main contribution of the proposed work: while it serves as an encyclopedia of asynchronous principles and microprocessors, it is intended to give direction for specifying, modeling, synthesizing, and implementing all classes of asynchronous circuits and systems, and to present a quantitative evaluation of existing asynchronous microprocessors.

The rest of the manuscript is organized as follows: We begin by presenting principles of the asynchronous logic in Sect. II. Sect. III-A details the design flow and EDA support for asynchronous circuits and systems. In Sect. III-B, we present an overview of the existing processors, and their quantitative evaluation. We conclude the manuscript in Sect. IV.

II. FUNDAMENTALS OF ASYNCHRONOUS LOGIC

In what follows, we briefly review the fundamental principles of asynchronous logic, knowing which is essential for understanding and designing asynchronous circuits and systems.

A. DATA AND CONTROL PATHS

Data path refers to a part of circuit that is responsible to perform operations, such as, arithmetic and logic on data. The control circuit, on the other hand, maintains the operation sequence of data path, as well as controls the timing.

Two asynchronous circuits are connected in such a way that their data paths are directly connected to each other, while their control paths are connected to each other by means of a pair of signals, known as *request* and *acknowledge* – together called the *control* signals. The latter indicate validity and safe reception of data between sender and receiver on the data path respectively. The instances at which the control signals are asserted lead to a distinction between two delay models of asynchronous logic; namely *bounded* and *unbounded*. In the former, the control signals are automatically asserted once a presumed delay, which is usually long enough for the corresponding operation on the data path to complete, has elapsed. On the other hand, in unbounded delay models, additional steps need to be taken to know for sure the data validity and their safe reception. Data and control paths are usually separately synthesized to gate level netlists, because each of them requires a different set of methodologies and tools.

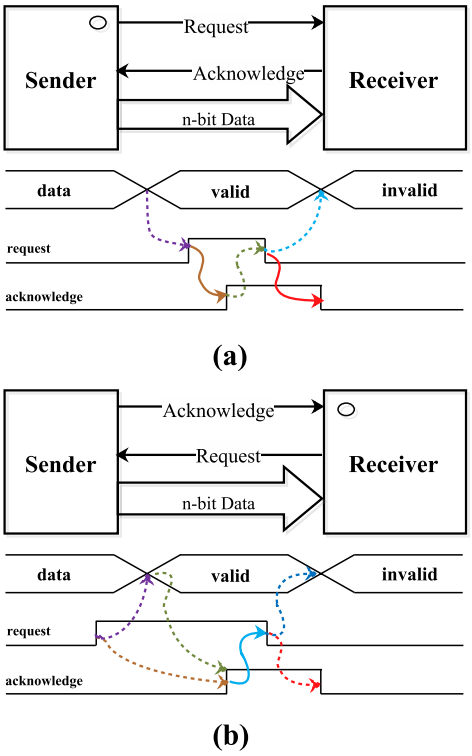


FIGURE 1. (a) Push channel, (b) pull channel.

B. HANDSHAKING CONCEPT IN ASYNCHRONOUS DESIGN

A channel is a point to point, unidirectional communication link that connects two asynchronous circuits. Usually there are three signals comprising a channel: request, data and acknowledge; the request signal may be encoded into the data bus in some cases. The sender places some data on the data bus, and indicates their validity to the receiver using a control signal. The receiver, on the other hand, receives the data, consumes them, and indicates its availability for receiving the subsequent data item, using the other control signal. This request-acknowledge activity, to transfer a data item, is termed as a *handshake*. The communication may be initiated by the sender; in which case the channel is known as a *push* channel, whereas, in a *pull* channel, the receiver initiates the communication by asserting the request signal fig. 1 depicts each type of channel with its block diagram and corresponding waveform.

C. CLASSIFICATION OF ASYNCHRONOUS CIRCUITS

Asynchronous circuits can be classified according to their delay models; the usual three classes are discussed next.

1) DELAY INSENSITIVE CIRCUITS

The class that is most robust against variations of process, voltage and temperature (PVT), is called delay insensitive (DI) [7], since it assumes arbitrary, but finite, wire and gate delays [4], [8], [9]. The receiver, in such circuits, is bound to properly acknowledge every transition by the sender, which

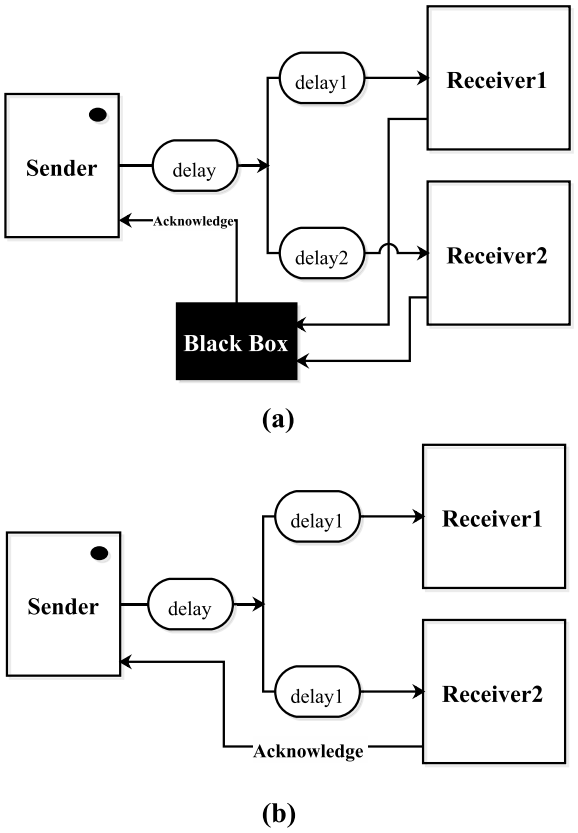


FIGURE 2. (a) Delay insensitive, (b) quasi delay insensitive.

means the next transition is allowed only when the previous data are correctly accepted and/or consumed. However, the number of asynchronous circuits that may be made DI, is very small [10]. A DI circuit is illustrated in fig. 2(a): the acknowledge signal is asserted once each of the two receivers has issued its own acknowledgment; indicating its availability for accepting the next one. The black box introduced, is responsible for *joining* the two acknowledgments (waiting for all the receivers to respond), since delay1≠delay2, and hence, their acknowledgments may arrive at different times as well.

2) QUASI DELAY INSENSITIVE CIRCUITS

DI circuits with isochronic forks are said to fall in quasi delay insensitive (QDI) class. This class of asynchronous circuits compromises the delay insensitivity property, by assuming that in isochronic forks, all the receivers receive the signal at the same time, as presented in fig. 2(b). That is, the input delay of each receiver is identical, so only one acknowledgment from any of them ensures the completeness to the sender. Isochronic forks, if not carefully implemented, may cause a hazardous effect in the circuits [11].

3) SPEED INDEPENDENT CIRCUITS

Speed Independent (SI) class assumes arbitrary, but finite, gate delays, and zero wire delays. This class is similar to

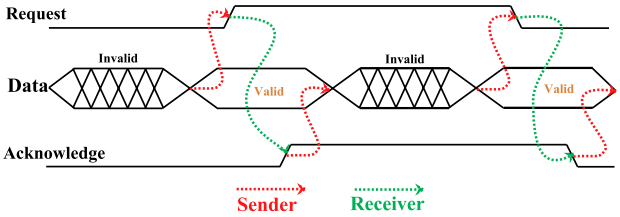


FIGURE 3. The 2-phase bundled data signaling.

synchronous style: it assumes that before the *req* signal is asserted, the data have to be stable at the receiver side; similar to synchronous approach, where the clock edge occurs sufficiently later than the data becoming valid and stable. So to achieve this, in asynchronous environment, there has to be an appropriate delay, by means of a buffer or inverter chain, in the *req* path. However, these circuits, by doing this, lose their robustness against PVT variations.

D. SIGNALING CONVENTIONS

In asynchronous designs, the local controller, instead of a global clock, governs the data movement on a channel [12]. The control signals follow some predefined pattern for accurate operation, where the latter is specifically known as signaling; this is discussed next.

1) 4-PHASE SIGNALING

To complete one handshake cycle, or to exchange one message, the 4-phase signaling protocol uses four transitions, two by each of the *req* and *ack* signals [13]. The waveforms illustrated in fig. 1 are examples of 4-phase bundled data[1] protocol. As may be seen in the waveform, the transition to high level indicates any valid event, while the transition to zero changes the phase that resets the communication – giving this scheme another name, *Return-to-Zero* (RTZ) protocol.

2) 2-PHASE SIGNALING

In 2-phase signaling, transition of request signal from zero to one, as well as, from one to zero, indicate validity of the data, as illustrated in fig. 3. Since there is no, unnecessary, resetting phase involved, this type of signaling is also termed as *Non-Return to Zero* (NRZ). Naturally, this type of signaling will lead to faster circuits, besides being more energy efficient due to fewer number of transitions required per data transfer.

In comparison to 2-phase signaling, the advantages that 4-phase signaling enjoys include increased robustness to delay variations, since the RTZ phase provides sufficient safety margin, and the circuits are relatively simpler to design. For instance, a level controlled latch can be directly driven by using control signals of the 4-phase protocol: one level switches it to opaque, while, the other makes it transparent. The 2-phase signaling, on the other hand, requires some additional logic to make the latch functional as required.

[1] II-E1

E. DATA REPRESENTATION

The purpose of communication is to transfer meaningful information in the form of data. The predefined suitable representation of data is also called encoding, on which two parties agree. Generally, in asynchronous logic, the data may be encoded in one of the two ways: 1) *single rail encoding* where one bit of data takes one line, 2) *M-of-N encoding* in which single bit of data takes multiple lines. Next, we discussed these two encoding schemes.

1) SINGLE RAIL ENCODING

In single rail encoding as mentioned above, each wire carries one bit of data [14]. The control signals use separate rails, and are said to be bundled with the data signals – hence the name *bundled data encoding*. In bundled data encoding, the control signals may adopt either of the two signaling conventions discussed above. In that case, the suitable prefix, 4-phase or 2-phase (whichever adopted), is placed before name of the encoding scheme. Because of their simplicity, these schemes are widely used, where the cost, in terms of area on the die, is approximately the same as synchronous equivalents [15].

2) M-OF-N ENCODING

This type of encoding is used within the DI class of asynchronous circuits, where N wires carry $log_2(N)$ bits of information (N is in a power of 2), and there is an explicit wire to carry the acknowledgment [12]. The dual rail encoding [16] is a special case of *1-of-N* encoding, with $N = 2$. Each bit is encoded using two rails: true and false. Level 0 is represented by logic '1' on the false rail, while '1' on the true rail is used to represent level 1. A '0' simultaneously on both rails indicates 'no valid data'. The two rails are mutually exclusive, so at a time, only one is allowed to make a transition. The new data validity at the receiver side is detected by transition, since no explicit *request* signal is available. The *completion detection* circuit performs this task. An example of dual rail encoding, and completion detection logic for each type of protocol, are illustrated in fig. 4 and 5 respectively.

Note in the completion detection mechanism, it is important to detect the RTZ phase on all lines as well, which cannot be handled by an AND gate. The reason behind this deficiency in AND gates is the fact that a low on only one input will cause a low on the output. Therefore, a component that waits for all the inputs to go low before it could deassert its output, should replace the AND gate. Muller C-element (MC) [17] is one such component, which has been the primitive for asynchronous logic since its inception.

From *1-of-N* encoding class, one hot encoding represents n bit data by 2^n lines. It is different from the dual rail codes for $n = 2$, in that it uses a 4-bit unique code to represent the 2-bit data, unlike the dual rail codes, which would encode each bit using two lines. This difference is presented in Table 1. Although the area overhead for the two equivalents remains the same, the fewer number of transitions in the *1-of-N* encoding makes it more energy efficient, and therefore the preferred method. With *1-of-2* encoding, the 4-phase protocol

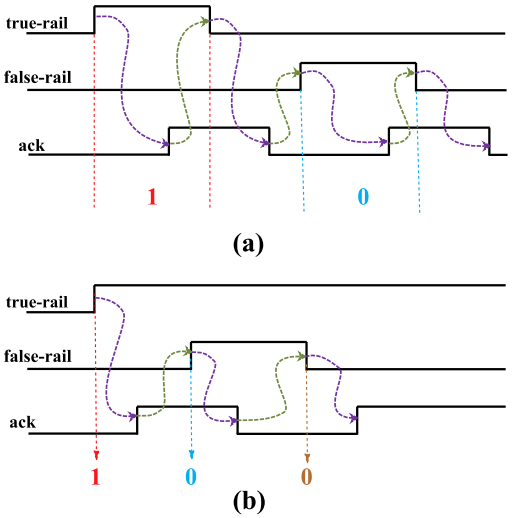


FIGURE 4. Dual rail encoding (a) 4-phase, (b) 2-phase.

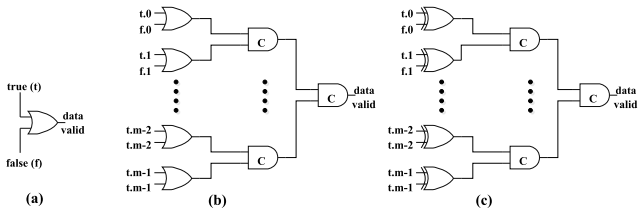


FIGURE 5. Completion detection logics (a) 4-phase signaling with 1-bit message, (b) 4-phase signaling with m-bit message, (c) 2-phase signaling with m-bit message.

TABLE 1. 2-bit Representation using dual rail, and 1-of-4 code.

Message	Dual Rail Code				1-of-4 Code
	true1	false1	true0	false0	
00	0	1	0	1	0001
01	0	1	1	0	0010
10	1	0	0	1	0100
11	1	0	1	0	1000

is known as *Null Convention Logic* (NCL) [18], [19], where the RTZ phase is called a spacer or an empty word, used to separate two code words. The other difference is usage of the majority or threshold gates [20] for completion detection, in comparison to MCs used in dual rail codes.

Level Encoded Dual Rail (LEDR) is another important dual rail encoding scheme [21]. In these codes, the data bit is the first bit in the codeword, followed by a 1-bit phase, which keeps alternating between *odd* and *even* for each codeword. Therefore, the two consecutive codewords are always different in their phases, making it possible to distinguish identical data items without the need of having a spacer in between – resulting in more energy efficient codes.

F. ASYNCHRONOUS PIPELINE IMPLEMENTATIONS

Efficient asynchronous circuits are usually built as pipelines, which increase the overall throughput by distributing the task among several function units operating in parallel on different data values. There are several types of asynchronous pipelines, micropipelines [22], mousetrap [23], GasP [24],

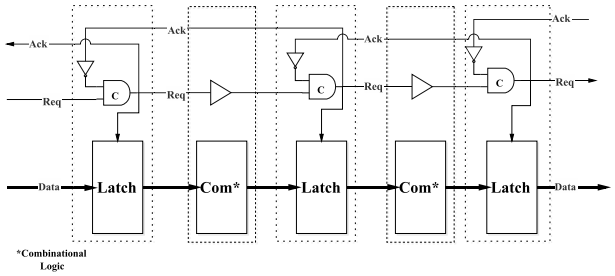


FIGURE 6. 4-phase bundled data pipeline.

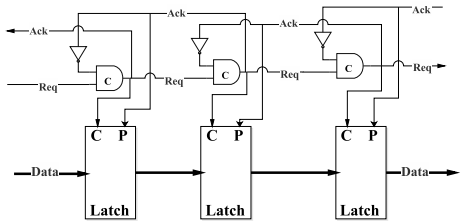


FIGURE 7. 2-phase bundled data pipeline.

QDI [25], [26], asP* [27], wave [28], [29], surfing [30], and RAMP [31]; all of them have a common Muller pipeline as their backbone though. The Muller pipeline is a simple arrangement of MCs, such that each of them forms a single stage. The output of each stage (MC) serves two purposes: 1) it becomes the input req to the successor stage, 2) it is sent as the inverted ack to the predecessor stage. The first stage receives its input req from the sender, and generates ack in return. Similarly the last stage generates the output req to the receiver, and receives the ack in return. The Muller pipeline is a mechanism that relays handshakes [4]. The pipeline is said to be empty when all the MCs are initialized to zero. At this point in time, the left-environment (also called sender or producer) can initiate the handshake by asserting req. While this transition ripples through the pipeline to the right-environment (also called receiver or consumer), due to the symmetry, each stage sends the acknowledge to the previous stage. Now in case the producer is faster than the consumer, it may deassert its req which should traverse the entire pipeline up to the last stage and get blocked, waiting for the receiver to consume the token by asserting the ack. Sooner or later, a time may come when all the stages get blocked because of the slow nature of the receiver. A fully filled pipeline has an interesting characteristic, i.e., alternating stages will always store opposite values. Singh and Nowick [23] made use of this feature to build the mousetrap pipeline.

The 4-phase bundled data pipeline with datapath is illustrated in fig. 6. In a completely filled pipeline, one can observe that only half of the pipeline stages store data, since each pair of successive MCs hold alternating logic. This pipeline configuration is just like a Master-Slave setup in synchronous designs [4].

The 2-phase bundled data pipeline, also known as micropipelines, was proposed by Sutherland [22]. As may be observed in fig. 7, the control path is identical to the Muller pipeline, with a slightly different signal interpretation, which makes it follow the 2-phase handshaking.

III. ASYNCHRONOUS PROCESSORS, LANGUAGES AND DESIGN TOOLS

A. TOOLS AND LANGUAGES

1) TANGRAM

Tangram [32], is a tool based on the dedicated programming language (a CSP based VLSI programming language) with transparent silicon compiler. In a Tangram program, a programmer can define whether commands are executed concurrently or sequentially. The Tangram program first translates into handshake circuits, as an intermediate state, prior to the VLSI circuit layout. The intermediate translator, known as tangram compiler, performs syntax-directed translation into handshake circuits [33], where the compiler contains the handshake circuits translation rule for each tangram program. At the next stage, transparent silicon compile or handshake circuit compiler performs two tasks: component substitution, and layout generation. In the former, the handshake components are implemented into standard cell library, and in the layout generation phase, commercial CAD tools are used. Some asynchronous chips programmed in Tangram are [34]–[38].

2) CHP: COMMUNICATING HARDWARE PROCESSES

CHP [39] is a programming language for fine grain distributed computation. Usually, a CHP program consists of the parallel configuration of several concurrent processes, where inside each process, the code is mostly sequential. These processes do not share variables; the latter are local to each process, but they may be passed to other processes as communication channel messages. The procedures and functions are also used as local variables. Integer (*int*), boolean (*bool*) and symbol are three generic variable types. For structuring data, two mechanisms named *array* and *record* are used, where the latter may contain many variables, each having its own type.

The process graph in CHP is made with a set of processes as vertices, and communication channels as edges. Initially, a process is declared and then instantiated. A process may be of two types: a meta-process or a simple process, where the former contains a number of sub-processes, and label *meta* identifies this type. On the other hand, label *chp* identifies a simple process.

Synthesizing a QDI system comprises few steps. At first, the system is described as a sequential CHP program, which is decomposed into a fine grain CHP process. This step is known as process decomposition, after which a CHP program is transformed into handshake expansion (HSE). Finally, the HSE code is transformed into a productive rule set (PSR) program. CHPsim locates and detects a deadlock, estimates the performance, and debugs the system, as well as provides syntactic and runtime checks, where the main and interesting

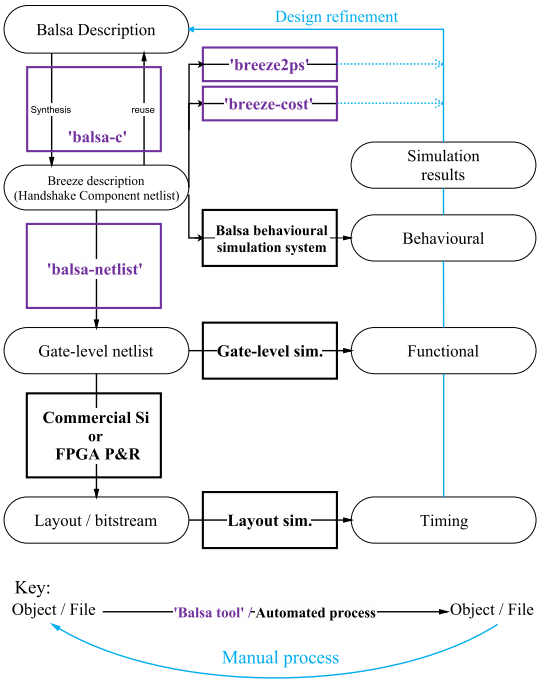


FIGURE 8. Balsa design flow: reprinted from [46].

function of the CHPsim is co-simulation. Many projects have been synthesized with CHP, including [40]–[45].

3) BALSA

Balsa [46], [47], is a language for describing asynchronous hardware system, as well as, it is an asynchronous circuit synthesis system that generates gate level netlists from Balsa high-level description language. Balsa design flow, shown in fig. 8, demonstrates the overall working.

Balsa contains a number of tools from which some of the important ones are listed below.

- *balsa-c*: Balsa language compiler, intermediate language *breeze* is the output of *balsa-c* compiler
- *balsa-netlist*: from a Breeze description it produces an appropriate netlist of the target CAD/technology framework.
- *breeze2ps*: postscript file of the handshake circuit graph is produced by this tool.
- *breeze-cost*: circuit area cost estimation tool.
- *balsa-md*: makefiles generating tool.
- *balsa-mgr*: for *balsa-md*, a graphical front-end with project management facilities.
- *blasa-make-test*: for a Balsa description it automatically generates test harness.
- *breeze-sim*: at the handshake component level the preferred simulator.
- *breeze-sim-control*: for the simulation and visualization environment a graphical front-end.

Balsa adopts a syntax directed translation method to yield communicating handshake components, where the compilation approach is transparent and similar to Philips Tangram

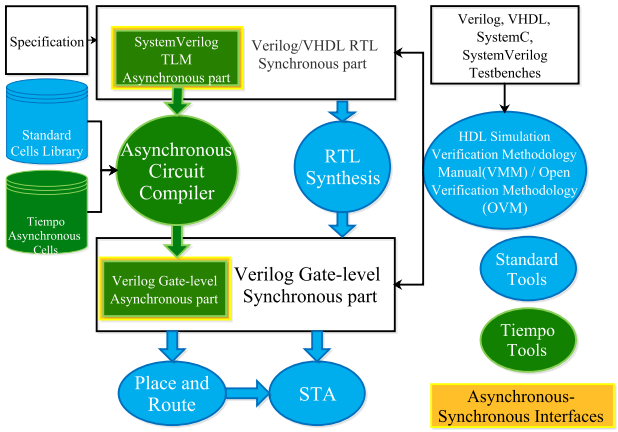


FIGURE 9. Tiempo asynchronous/synchronous circuit design flow: reprinted from [48].

system [32]. The set of ≈45 handshake components are listed in [47], which are connected by channels, on which the communications take place.

4) ASYNCHRONOUS CIRCUIT COMPILER

Asynchronous Circuit Compiler (ACC) is the first fully automated synthesis tool for asynchronous and delay-insensitive circuits. It is used in Tiempo [48] asynchronous circuit design flow [49] as shown in the fig. 9. The input of ACC is a description written in Transaction Level Modeling (TLM) using SystemVerilog [50] description language. Such a format gives logical integration of Tiempo clockless technology into verification platforms such as Mentor Graphics Questa^{TM}, Cadence NCsim^{TM} and Synopsys VCS^{TM}. It produces output at gate level netlist in Verilog description language. In addition to standard cell libraries, ACC uses Tiempo asynchronous cells for circuit mapping. The gate-level netlist representation generated using standard back-end and electrical simulation tools can be placed-and-routed and simulated respectively. As ACC is made interoperable and compliant to standard design flows, it can be integrated/used with any tool based on industry standard. As an example, TAM16 [51] IP core is designed by using Tiempo fully asynchronous and delay insensitive technology.

5) PETRIFY

Petrify [52], [53] is a tool for synthesis of Petri Nets (PN) and asynchronous circuits. It manipulates concurrent specification as well as optimized asynchronous control circuits. Petrify generates bi-similar and simpler PNs or a Signal Transition Graph (STG) from the originally described PN or STG (fig. 10). Furthermore, it transforms a specification using token flow analysis of the initial PN which in turn yields a Transition System (TS). In an initial TS, the same label transitions are counted as one event. The condition required to obtain pure. unique, free and non-redundant PN is that the TS is transition re-labeled and transformed. By using design gate library, Petrify generates asynchronous controller net list while the input-output behavior remains unchanged.

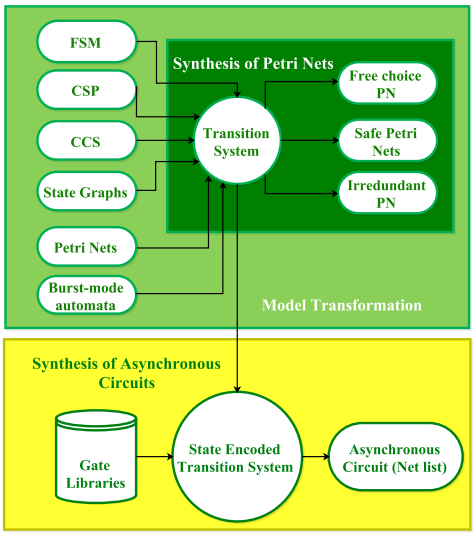


FIGURE 10. Petrify framework: reprinted from [52].

By solving complete state problem, it performs state assignment when asynchronous circuits are synthesized, and generates speed independent circuits [54].

6) OTHER TOOLS

In the literature, there are a number of other similar tools including Teak [55], Occam [56], LARD [57], DESI [58], VSTGL [59], Workcraft [60], VERISYN [61], Pipefitter [62], CHAINworks [63] and TiDE [64].

B. PROCESSORS

Processors that are dependent on a global clock are synchronous where the clock regulates processing. The global clock in such processors may become problematic, in particular, when the processing environment is more complex. The main issues faced in a synchronous processor are the clock skew and the clock circuit itself. The later can dissipate an enormous amount of power because it's always running. A alternative choice among the research community is to consider asynchronous designs. In an asynchronous design. each functional unit communicates with other by using a local clock or more technically using handshaking. Such a design choice delivers simplified interfacing and average case performance as compared to the worst case performance in synchronous design. In asynchronous designs, the clock delay is larger than the delay of slowest component. Asynchronous processors are efficient in power dissipation as only the required part of the circuit is alive. In this section, different asynchronous processor designs are explored.

1) CALTECH ASYNCHRONOUS MICROPROCESSOR

Caltech Asynchronous Microprocessor (CAM) [40] is a 16-bit RISC type architecture with 16 general purpose registers, an ALU, four buses, and two adders. The two adders are used for memory addresses calculation and program counter calculation, respectively. The CAM use 4-phase handshaking protocols with dual rail-data transfer. The estimated performance of the CAM processor was approximately 15 MIPS at 7V when realized with 2μm Scalable CMOS version at room temperature and 30MIPS at 12V when a chip is cooled with liquid nitrogen. The performance was estimated to be 26MIPS at 10V@105mA when realized with HP 1.6μm SCMOS. The processor is realized using Harvard architecture and its chip consists of 2000 transistors.

2) FULLY ASYNCHRONOUS MICROPROCESSOR

Fully Asynchronous Microprocessor (FAM) [65] is a 32-bit RISC like architecture with 4-stage pipeline. Its data path includes 32-bit ALU, 32 registers (each 32-bit wide), 32-bit barrel shifter, multiplier and an adder. The 4-stage pipeline uses ALU and register file and includes operations instruction fetch, memory access, instruction decode and instruction execute. It consists of two types of blocks: a computation block and an interconnection block. The computation block includes adder, shifter and register while the interconnection block includes combinational logic, pipeline register and a data latch. The instruction set of FAM microprocessor has 18 instructions, uses 4-phase handshaking protocol with dual rail-data transfer and is based on CMOS technology with approximately 71000 transistors. Its design is based on Differential Cascade Voltage-Switch-Logic (DCVSL) for completion detection of combinational logic with performance measured 300MIPS in 0.5 micron CMOS.

3) NON-SYNCHRONOUS RISC PROCESSOR

Non-Synchronous RISC (NSR) [66] is a 16-bit load/store architecture with 16 general purpose registers and contains 5-state pipeline. The 5-stage pipeline include units for instruction fetch, instruction decode, execute unit, memory interface and register file. In NSR, stalling caused by a slower instruction is covered by adding self-timed FIFO queues between concurrent units. Each block accepts data from other blocks for processing and sends the result by means of FIFO queues. An instruction that does not need a particular pipeline stage is never passed through that stage. For example, if an instruction does not use the memory, it is never passed through the memory interface pipeline stage. The self-timed concurrent blocks in NSR design communicates using 2-phase bundled data protocol. For a prototype NSR, seven Actel Field Programmable Gate Arrays (FPGAs) are used. To test any unit of NSR processor, each unit request is blocked by using a switch to hide the request and acknowledge signals from other units. The best case performance of NSR is estimated to be 1.3MIPS.

4) COUNTERFLOW PIPELINE PROCESSOR ARCHITECTURE

Counterflow Pipeline Processor (CFPP) [67] architecture (fig. 11) is realized using SPARC instruction set and is based on the idea that instruction flows in one direction and its result on other within a pipeline. The CFPP have multi-stage pipeline design in which program counter is at the bottom

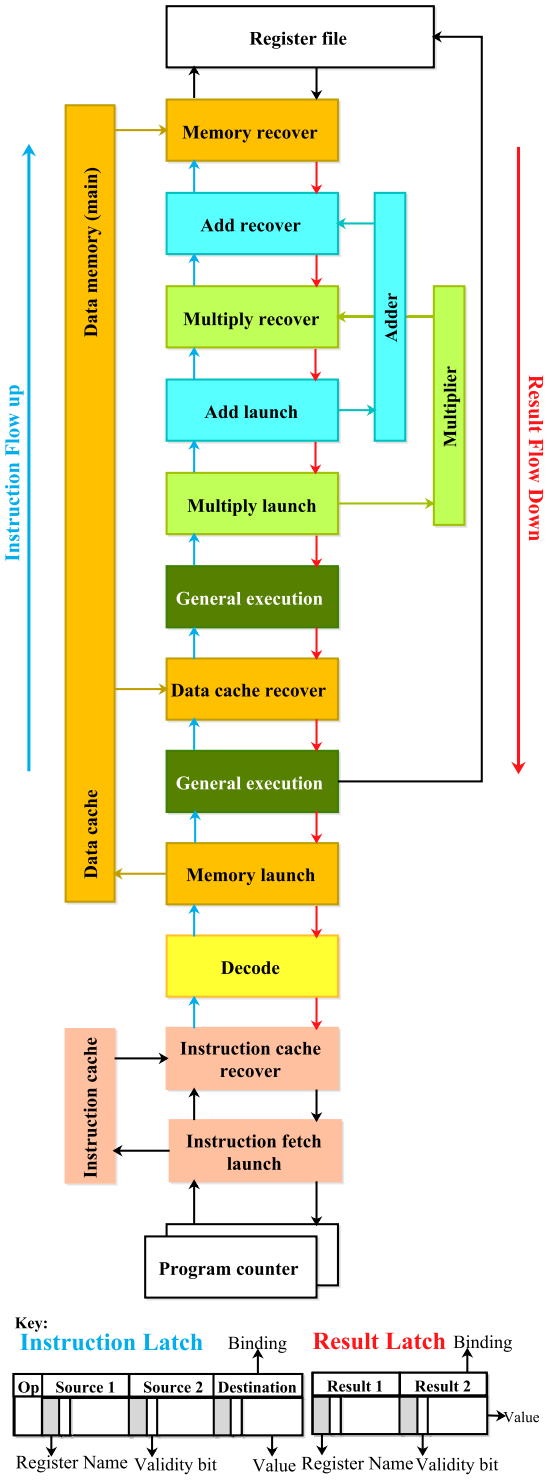


FIGURE 11. Counter flow pipeline processor: reprinted from [67].

while the register file is placed at top (of pipeline). An instruction flows up for execution and stalls when the upper pipeline stage cannot accept a new instruction. An instruction may also stall if it reaches execution stage while at the same time the upper stage include a stalled instruction. Such situation may be avoided if there is a gap of arbitrary size in a pipeline which leaves some stages empty (without any instruction).

An instruction include opcode, source and destination bindings as shown in the fig. 11. Each binding contain three fields register name, validity bit and a value. When an instruction reaches execution stage, the new data value is loaded to a destination binding and marked valid after the execution. Similarly, when an instruction reaches top pipeline stage, the data value from destination binding is loaded to a specified location of a register file. Afterwords, the destination binding flows downward in the pipeline on the result of a subsequent instruction.

There are two bindings in a result pipeline. If a subsequent instruction needs source binding with its register name matching the register name of result binding, it garner the value from register pipeline to instruction pipeline, just like bypassing or forwarding. On the other hand, if the executed instruction destination binding matches with the result binding, then the result binding garner the value from destination binding. With any register that match, the new instruction source binding updates with most recent values. Another situation arises when an instruction yet to execute with result binding match with destination binding, the result in binding is killed. As the result binding is not valid for later instructions, all the rules just described guarantee the correct result binding for instructions. A multi-result binding on different stages of a pipeline at the same is similar to register renaming.

Issues like trap and incorrect branch predictions are resolved by the architecture effectively. Trap caused by any instruction on any stage generates trap-result bound to a result pipeline (not to a destination binding). The instruction that causes a trap is set to invalid that may proceed to next pipeline stage but will have no effect on register file or result pipeline. The trap-result is interpreted by stage responsible for program counter control which changes program counter to a suitable trap handler. The incorrect branch predictions are similarly (to traps) handled while the program counter control starts execution from the proper path.

In CFPP architecture, non-identical stages perform different processing: one stage, for example, performs multiplication while the other performs addition. This architecture may use siding which performs execution of long computation delays instruction. This implies, when multiplication instruction reaches multi-launch, it stalls till all operands required become valid before launch. Multiplication is shifted to siding multiplier and execution result is recovered in later multi-recover stage. Other sidings such as adder and memory sidings are also available in this architecture.

Non-identical stages weakens the performance of architecture: if a store instruction, for example, is dependent on multiplication instruction then the compiler must reschedule independent instructions between them. This way, when the multiplication instruction propagates to eight stage, the store instruction would propagate to fourth stage. When the multiplication stores the result in 10th stage, there will be five empty stages between multi-recover and the memory launch. The result from multiplication enters pipeline and propagates through five stages to reach awaiting store

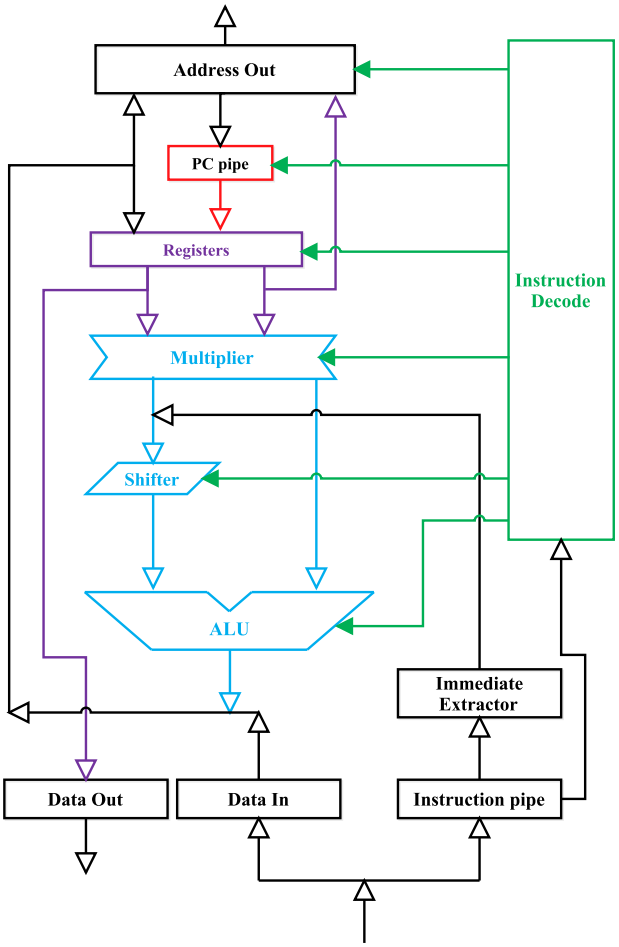


FIGURE 12. AMULET1 organization: reprinted from [68].

instruction which affects the throughput. It had long stage pipeline which requires an excessive amount of area. This version was not implemented on hardware.

5) AMULET1

AMULET1 [68] is an asynchronous version of ARM processor and is object code compatible with ARM6 (32-bit) processor. It consists of functional units address interface, register bank (with 30 general purpose registers each 32-bit wide), execution unit and data interface (fig. 12). Each functional unit in AMULET1 works concurrently and independently. To avoid control hazards, coloring mechanism is used [69]. In this mechanism, a color bit is used to represent the state of the processor as well as the same color bit is allocated to an instruction fetch at a particular moment. Whenever the instruction and processor color bits mismatches, the instruction is discarded. The processor color bit changes on the termination of the instruction stream. This architecture uses register locking mechanism, 2-phase single rail protocol for communication and bounded delay timing model and operate on fundamental mode of operation.

The AMULET1 processor is fabricated using two CMOS processes where 1μm process at ES2 gives the performance of 20.5K Dhrystone (@5V and 152mW, 77MIPS/W) while 0.7 μm process at GEC Plessey Semiconductor gives the performance of 40K Dhrystone @ 5V [69]. It does not give best performance compare to its synchronous version ARM6 but gives a clear way for asynchronous implementation.

6) TITAC: DESIGN OF A QUASI-DELAY-INSENSITIVE MICROPROCESSOR

TITAC [70] is a non-pipelined asynchronous implementation of 8-bit Von Neumann microprocessor. The processor is organized as a control section and data path section. The control section contains two independent controllers (controller 1 and controller 2) where the first controller is hard wired controlled. The other controller is microprogrammed which controls outside chip storage. Either controller can be selected to control data path section. TITAC microprocessor is based on quasi-delay insensitive timing model and uses 4-phase communication protocol. It is realized using 1μm CMOS and uses $\approx$22068 transistors with the estimated performance measured 11.2MIPS and 1.8MIPS, respectively, for controller 1 and controller 2.

7) THE GALLIUM ARSENIDE ASYNCHRONOUS MICROPROCESSOR

The Gallium Arsenide (GaAs) Asynchronous Microprocessor [71] with a 16-bit RISC pipeline architecture is the modified implementation of Caltech Asynchronous Microprocessor using GaAs Technology. The processor data path includes program counter, 16 general purpose registers, an ALU and memory unit for load/store operation execution. All data path gates, completion detection circuit and buffers, except NAND gates, are Direct Coupled-FET Logic (DCFL). The performance of the processor measured is 50MIPS/W.

8) FRED ARCHITECTURE

Fred [72] is a self-timed decoupled pipeline computer architecture based on micro-pipelining and roughly based on NSR III-B3. It uses most of the Motorola 88100 instruction set. Fred organization include dispatch unit, register file (32 general purpose registers) and execution unit as shown in the fig. 13. Dispatch unit is the main control unit that controls program counter, instruction fetch and sends instructions to other functional units. It issues instructions and monitors data hazards after satisfying data dependencies by resolving register destination conflict. Execution unit has five functional units (arithmetic, logic, control, memory and branch unit) where a distributor is responsible for sending an instruction to appropriate unit. The result of each functional unit is written back to register directly or by a register (R1) queue. In Fred architecture, direct result forward to other functional units is not allowed due to complexity.

Many decoupled independent processes of Fred architecture are connected via FIFO queues (dedicated paths) of arbitrary length to process various instructions at a time. As each pipeline stage passes data by communicating locally with the neighbor stage, no extra control circuitry is used for

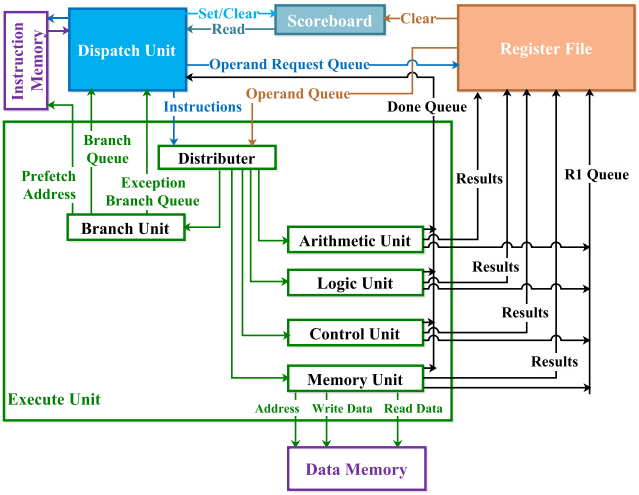


FIGURE 13. Fred block diagram: reprinted from [72].

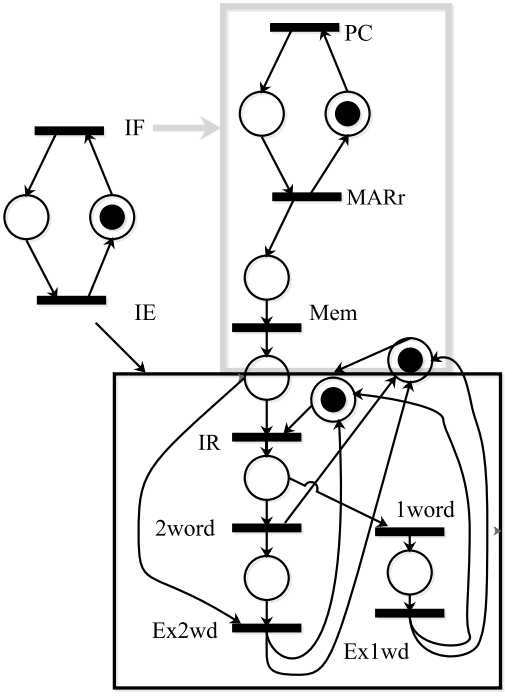
adding additional pipeline stages. Fred prototype is described in hardware description language VHDL and is fully functional. For performance measurement, different benchmark programs were run through Fred and the average performance measured was 149.67 MIPS.

9) HADES ARCHITECTURE

Hatfield Asynchronous DESign (HADES) [73], [74] is a superscalar RISC type processor with Harvard architecture. HADES is a step towards the design of an asynchronous superscalar processor. It includes four pipeline stages namely instruction fetch (fetches in groups), instruction decode (twofold operation), execution (independent functional units) and write-back stage. In the write-back stage, two register files integer and boolean are used. The condition generated by integers comparison is stored in the boolean register file for resolving branches. It addresses read-after-write and write-after-write hazards using register locking mechanism and decoupled operand forwarding. To resolve such hazards, each functional unit in execution stage have separate forwarding register. This architecture is capable of issuing single and multiple instructions: instructions are issued in order but allows their out of order completion. Furthermore, it uses 4-phase protocol for communication. A formal specification language, Communication Sequential Processes (CSP) [75], is used for designing baseline of HADES processor in which all concurrent processes communicate asynchronously. The specification language CSP and description language VHDL allows the designer to check correctness of the design and simulate them easily.

10) ASYNCHRONOUS PROCESSOR BASED ON PETRI NETS

The processor in [76] is based on Holton's [77] 3-bit simple synchronous processor design where the asynchronous version employed the same instruction set specified in synchronous implementation. It performs the operations load general register (LdGR), load accumulator (LdAcc),



Key:
IE Instruction Execute, IF Instruction Fetch.
IR Instruction register, MAR Memory address register.
Mem Memory, PC Program counter, r Read, w Write.

FIGURE 14. Asynchronous microprocessor high level specification with Petrinets: Reprinted from [76].

arithmetic operation (Arth) and store. At first stage, labeled petri nets are produced (as shown in fig. 14) where places are represented by circles and transitions are represented by bars. An abstract labeled petri net includes two places and two transitions for instruction fetch and execute mode. An instruction and word fetch results program counter (PC) increment while an instruction execute transition produces complex structure. An instruction can be one or two-word wide: on the completion of one word instruction, the processor executes next instruction while on the completion of two-word instruction, the instruction (first) word remains in the instruction register as the second word fetched contains data.

Instruction execution completes as follows: the load instruction is decomposed into decoding of instruction LdAcc and fetching the second word Accdta. Arithmetic instruction execution is completed using ALU and latching result to the accumulator. Store instruction is completed after memory address register write (loading address) and memory write.

After the completion of labeled petri nets, the designer derives the temporal relation between the transitions. The analysis shows the increment of PC is concurrent with all execution transitions. The design was improved (from version 1) by refining using decoupling ALU as arithmetic instructions do not require data from memory. In this refinement, the acknowledge is sent to Memory Address Register (MAR) after decoding. In this improved version (version 2), the ALU

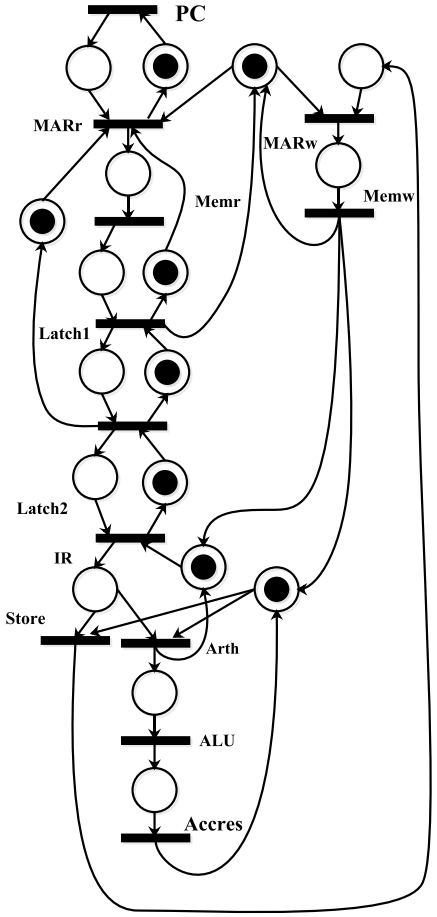


FIGURE 15. Pipelined processor model with two latches: reprinted from [76].

and Accres (latching result to accumulator) are concurrent to MARr and Memr (memory read) that produces reduction in arithmetic operation execution time. In this later version, modules are decoupled further because of low concurrency. Furthermore, instruction register is concurrent with ALU and Accres where instruction decode work concurrent with MARr and Memr. This, however, introduces deadlock and stall signal. With further refinement (version 3), the deadlock was removed by adding new register for storing fetched word and allow MAR to accept request from this transition. The stall signal allows new PC value to MAR only when previous instruction start to decode.

Further improvement was brought in version 3 (of the processor) after analyzing the temporal relation of modules in the processor. Extra latch was introduced for decoupling memory and instruction registers. The instruction register is now concurrent with MARr and Memr (version 4 of the processor). This include 4-stage micro-pipeline with extra feedback as shown in fig. 15. The performance of the designs was measured using UltraSAN. In version 4, PC-cycle takes 109.0ns and execution of Arithmetic operation takes 100.0ns. At the second stage, the labeled petri nets are translated to asynchronous circuits where the translation method employed was inspired from Patil's work [78].

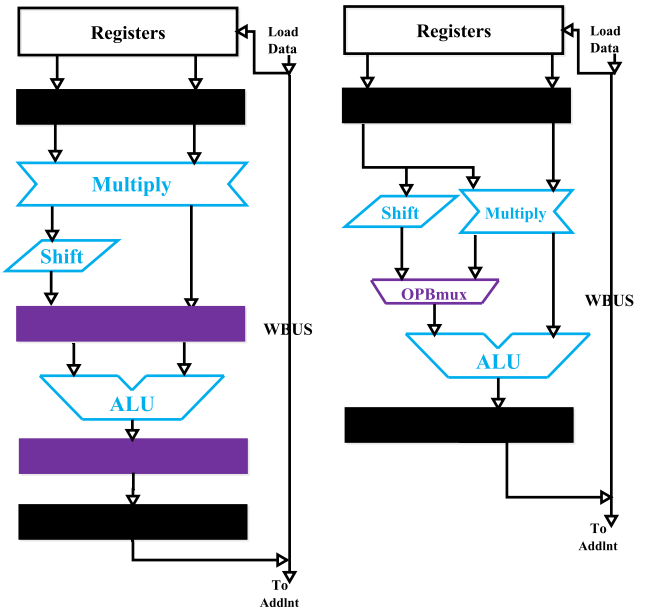


FIGURE 16. Execution pipeline of Amulet1 (left) and Amulet2 (right): Reprinted from [79].

11) Amulet2e: AN ASYNCHRONOUS EMBEDDED CONTROLLER

Amulet2e [79] is an asynchronous embedded controller powered by asynchronous ARM core. Amulet2e has asynchronous ARM core, RAM/cache of 4Kbyte and memory interface to connect with external memory. For performance gain, amulet1 was modified to amulet2 where the main change in execution pipeline is reduction of pipeline stages as shown in fig. 16. In amulet2, stall due to register locking is resolved with forwarding mechanism using the last result register (LRR) and last loaded value (LLV) techniques to bypass the register read as shown in fig. 17. To predict branches, amulet2 introduced Branch Target Cache as shown in fig. 18. Branch prediction is another performance edge compare to amulet1. In amulet1, issues raised due to sequentially pre-fetch instructions from program counter are corrected by execution pipeline.

In amulet2, HALT is introduced for power efficiency where the system resumes full performance on interrupt. The range of chip select lines, address bus and bidirectional data bus in the memory interface of amulet2e makes it more convenient than amulet1. Furthermore, amulet2e uses 4-phase bundled data protocol for communication. When all performance features are turned on, the performance measured is 42MIPS (Dhrystone 2.1 benchmark). On the downside, amulet2e is only used as a research prototype and is not suitable for commercial use.

12) ASYNCHRONOUS MIPS R3000 MICROPROCESSOR

The asynchronous version of MIPS R3000 microprocessor is known as MiniMIPS [41]. The microprocessor MiniMIPS has a 32-bit RISC CPU, memory management unit and two 4Kbyte on-chip caches (instruction cache and data cache).

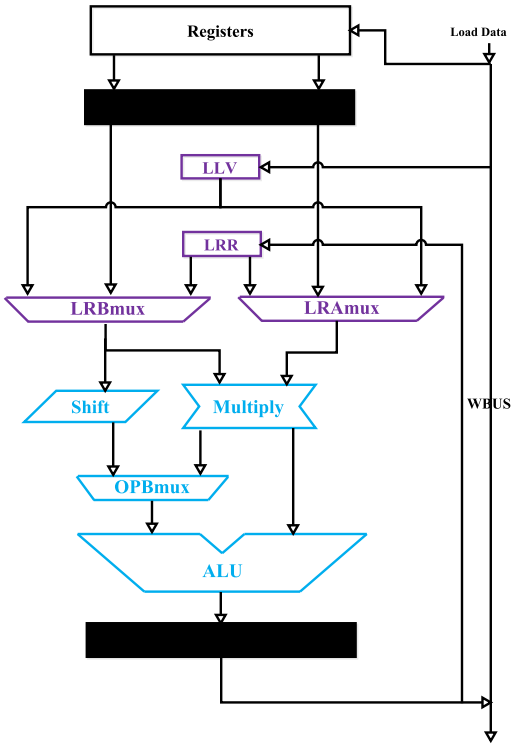


FIGURE 17. Organization of register forwarding: reprinted from [79].

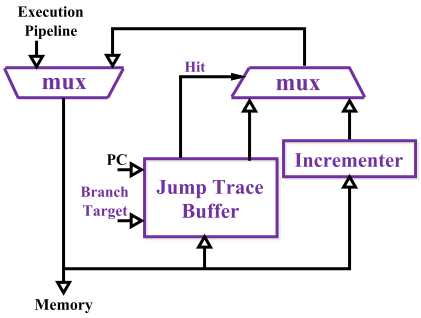


FIGURE 18. Organization of branch target cache: reprinted from [79].

It contains 32 general purpose registers (each 32-bit wide), two special purpose registers for division and multiplication and a program counter. The pipeline structure in MiniMIPS includes fetch loop and execution pipeline. The fetch loop has program counter, fetch and decode unit. The execution pipeline, on the other hand, includes execution, register and write back units. All execution units (e.g., adder and multiplier) are parallel and works concurrently. This means, multiplier's result is directly written to register and is not passed to any other execution unit or write back unit.

The MIPS R3000 microprocessor is a 3-stage pipeline architecture where as MiniMIPS is very carefully pipelined to gain performance. The main design goals are to achieve, without sacrificing low power advantage of the asynchronous design, high throughput, address the architecture issues missed in CAM, precise exception, branch delay slot, branch prediction, register bypassing and caches. MiniMIPS microprocessor operates on two modes: user and kernel mode. The design uses 4-phase handshaking protocol (dual rail or 1 of N code) and quasi-delay insensitive timing model. The performance of MiniMIPS measured is 180MIPS @ (4W and 3.3V @ 25°).

13) TITAC-2

TITAC-2 [80], an asynchronous implementation of MIPS R2000, is a 32-bit microprocessor based on scalable-delay insensitive model. It has a modified version of instruction set and include multiply/divide, delay slot of branch and privilege instructions. As the instructions encoding of TITAC-2 and MIPS R2000 are different, they are not object-code compatible. They, however, are similar in pipeline stages (both have five pipeline stages), precision exception handling, external interruption, memory protection and chip cache. The pipeline stages include instruction fetch, instruction decode, execution, memory access and write back.

TITAC-2 introduced new timing model based on scalable-delay-insensitive (SDI) model. In short circuit functions, the delay becomes K times larger than estimated delay, where K is the maximum variation ratio. The SDI model is faster than delay-insensitive (DI) or quasi-delay-insensitive. This model is used for subsystems where global interconnection uses DI model. By using Dhrystone V2.1 benchmark, the measured performance of TITAC-2 is 52.3VAXMIPS (@ 2.11W and 3.3V).

14) ASYNMPU

ASYNMPU [81], fully asynchronous CISC microprocessor, is the first implementation of CISC microprocessor that is pin-to-pin compatible with Intel 8/16-bit microprocessor. The functional units of ASYNMPU include pre-fetch, decode, control, execute unit, three ports (one read and two write ports) and its register bank has 26 registers. The execute unit includes bus interface, arithmetic logic unit, mov unit and a miscellaneous unit for handling microinstructions. As ASYNMPU is pin-to-pin compatible with Intel 8/16-bit microprocessor, the bus interface unit makes it possible to interface the external synchronous system with asynchronous processor. In Von Neuman architecture, the control (read, write data) and pre-fetch (for instruction fetch) unit may access bus interface at same time, result metastable state. In ASYNMPU, the bus interface unit has arbiter [82], [83] block for avoiding such metastable state.

The design of ASYNMPU addresses the complex feature of CISC microprocessor using asynchronous processing techniques. Among its major features are instruction set compatibility, controller-sequencer and variable instruction length handling. The ASYNMPU uses 2-phase bundled data handshake protocol and its instruction size varies from 1-6 bytes. Its performance is equivalent to an Intel 8/16-bit microprocessor (uses a 33MHz clock) and the average power dissipation calculated is 110mW. Bus interface of ASYNMPU unit in the busy state uses 11mW where as it uses 0.73mW in idle state: a distinctive feature of asynchronous implementation.

15) ECSTAC

Event Controlled Systems Temporally specified Asynchronous CPU (ECSTAC) [84], [85], is a fast asynchronous microprocessor based on event controlled system design methodology. It is a linear pipeline Harvard architecture and uses RISC like ISA with 8-bit data path and 24-bit address path. This mismatch results in a performance trade-off. ECSTAC architecture includes program counter, instruction and data cache, instruction decode FIFO, operand fetch, ACS (24-bit adder, comparator, and stack processing), ALU, order unit, register and scoreboard. The instruction decode unit is heavily pipelined to accommodate the data received from instruction cache. After instruction decode stage, the operand fetch stage fetches the operands which requires registers. The output is formed with immediate value (if any) and sent to ACS unit which performs 24-bit address offset addition. It checks the jump condition: if it is true, a signal is sent to all preceding units to invalidate instruction within it due to branch.

The instruction decode unit includes stack pointer that provides address for reading and writing from data memory. Order unit maintains instruction order of ACS and records the unit (ALU or data memory) used by the instruction. Furthermore, the order unit multiplex data bus of ALU and data memory to register bank (8-bit 16 general purpose register and flag register). It returns events from register bank to ALU and data memory. The scoreboard scheme, on the other hand, is used to prevent data hazards. It is based on a transition signaling operating under fundamental mode. Processor gives peak performance of 28 MIPS while fabricated in ES2 0.7μm DLM CMOS process.

16) TinyRISC TR4101 MICROPROCESSOR CORE

ARISC [86] is an asynchronous implementation of TR4101 embedded microprocessor core based on Harvard architecture. Its pipeline structure includes fetch, decode, flush, register read, register write, issue and execution units. The PC register holds address of token (32-bit instruction word) that is used by the fetch unit to provide a token to decode unit. After the instruction is decoded, it is sent to flush unit which checks branch condition. If the branch condition is true, the instruction pipeline is flushed. The issue unit issues instruction to relevant execution unit when the operands are ready and issues new PC value for PC-ALU.

Opaque latch controller is used on the input of each parallel execution unit for faster and power efficient instruction execution. The register locking mechanism ensures the correct register write and read which avoids data hazards. The ARISC microprocessor operates on MIPS-II/MIPS16 modes, where all units, except fetch and decode units, operate independently. The data path is designed and verified using Verilog hardware description language and Synopsys synthesis tool, respectively. The speed independent control logic design is accomplished partly by hand and using Petrify tool. Its design was simulated (can be used as well) with three different configurations: 1) separate instruction and data cache, 2) shared cache, and 3) synchronize bus interface and synchronous shared memory module @83MHz. The 4-phase bundled data protocol and normally opaque latch controller (where needed) are used for communication. High MIPS are achieved by using asynchronous configuration as configuration 1) gives performance of 74MIPS and 635MIPS/W(Stanford benchmark) and 123MIPS(Peak benchmark) with $V_{dd} = 3.3V$.

17) ASPRO-216

ASPRO [87] is a standard-cell QDI 16-Bit RISC asynchronous microprocessor based on A. Martin's method specifically design for an embedded application. It is a scalar processor where instruction issues in order and completes out of order. The fetch-decode loop includes PC-unit which sends addresses to program memory interface. The program memory interface fetches instructions either from on-chip memory or external memory. The external and program memory is 48K and 16K words each, respectively, and the instruction words are 24-bit wide. The external and program memory together with instruction decoder work in fetch-decode loop. The decoder sends information to data-path loop and acknowledge to PC-unit. At this point, if branch or unconditional branch is taken, the PC-unit sends target address, otherwise, the incremented address is sent to fetch-decode loop.

The data-path loop includes register file (16 general purpose register), bus interface and the processing units. The processing unit includes ALU, branch unit, load/store unit and custom unit (for future enhancement). The ALU has ''Min'' and ''Max'' instructions that are used for image processing, bit reversing used in FFT computation and ''Slt'' for carry and overflow testing. This architecture also has data memories (64Kbytes, byte/word addressable) where 256-word area is reserved for peripherals (accessed with dedicated instructions). ASPRO design is completed using standard cell library of 0.25μm 5 metal layer CMOS technology with automatically generated RAM and gives a performance of 140(peak)MIPS [42].

18) 80C51 MICROCONTROLLER

The microcontroller 80C51 [36] is an asynchronous implementation of 8-bit CISC type microcontroller for achieving power effectiveness. The 80C51 asynchronous microcontroller is fully bit and timing compatible with synchronous 80C51. Asynchronous 80C51 is based on the latch with latch enable signals as well as demand-driven peripherals. Its design has been described in description language Tangram [88]. Standard cell gate-level netlist is achieved from Tangram description after intermediate handshake circuit level. The handshake circuit uses 4-phase bundled data protocol. The design is processed in 0.5μm 3 metal layer CMOS which contain data RAM of 256bytes and program ROM of 16Kb. Gate level simulation of asynchronous

80C51 results 2.10MIPS(943MIPS/W) when memories are excluded, and worst-case condition is assumed.

19) AMULET3

AMULET3 [89], successor of AMULET1 and AMULET2e, is a 100 MIPS asynchronous embedded processor based Harvard architecture. It is a viable asynchronous processor for commercial use as it supports 4T version of ARM architecture and 16-bit Thumb compressed instruction set[2] for more detail on Thumb please check [90]. The AMULET3 processor has six pipeline stages that include instruction prefetch, instruction decode, execute, data memory reference, record buffer, and register result write-back stage as shown in fig. 19. Branch prediction unit in instruction pre-fetch stage supports thumb code. The decode and register read stages include logic, such as ARM and Thumb decode, and mechanisms such as register read and forwarding. It either decodes thumb critical control signals directly with thumb instruction decoder, or it first converts them to ARM equivalent instruction and then uses ARM instruction decoder. Register read and forwarding stage traces operand in the register file and search the reorder buffer if the operand is not available. The forwarding process stalls till the value become valid, where three read ports are available for AMULET3 register file.

Execution stage has adder with carry arbitrary scheme, multiplier (computes 32x32 product in approximately 20ns) and shifter. The program status register is fit logically into the execution stage. Reorder buffer stage stores result from execution pipeline stage and data memory interface. The results in reorder buffer may be orderly written-back to register file as well as used for forwarding purpose. The AMULET3 uses 4-phase communication protocol, gain high performance as compared to its predecessor and operates up to 120MIPS (Dhrystone 2.1).

20) A8051

A8051v1 [91] is a novel asynchronous pipeline architecture for CISC type embedded controller and is compatible with Intel 8051. It proposes optimized instruction execution scheme by skipping the redundancy and bubble states and uses only required stages. The A8051v1 is a multi-looping pipeline architecture and handles variable length instructions (1 to 3 bytes) for CISC type machine. It has 5-stages pipeline which include instruction fetch (pre-decode with branch predictor unit), instruction decode, operands fetch, microinstruction execution and write back unit. The instruction decode unit checks data dependency for the previous instructions. The microcontroller uses 4-phase handshake protocol, dual rail encoding and delay insensitive timing model. It was realized using 0.35μm CMOS technology while the performance measured by the designers was 75.5MIPS with 5-stage pipeline. Without the pipeline, the A8051v1 delivers with performance of 35.8MIPS.

[2]Effect on the processor is same as 32-bit ARM instruction.

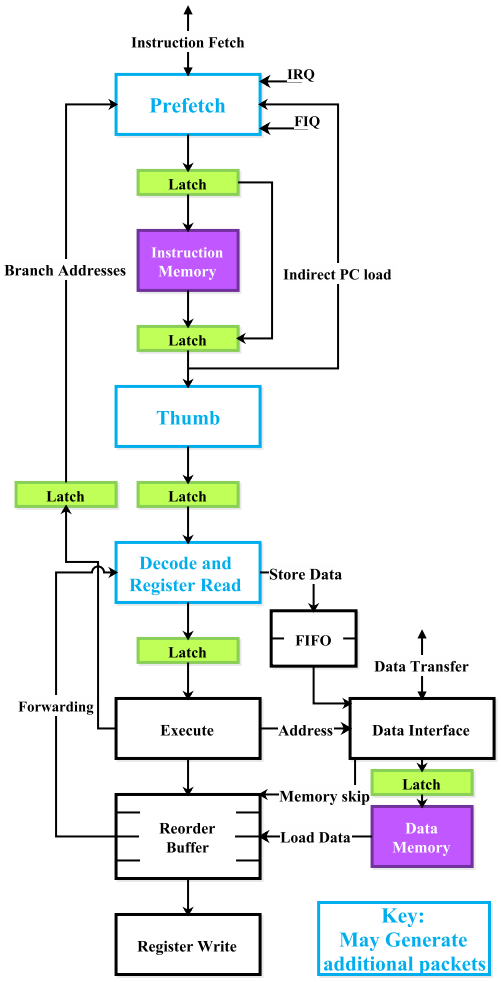


FIGURE 19. AMULET3 organization: reprinted from [89].

21) THE LUTONIUM MICROCONTROLLER

Lutonium [92] is an asynchronous implementation of 8-bit CISC type 8051 microcontroller for low ET^2 where E is average energy per instruction and T is the cycle time. Based on Harvard architecture, the 8051 microcontroller supports 255 variable length instructions, with each instruction varies from one to three bytes. Lutonium architecture, as shown in the fig. 20, includes fetch/IMem, decode, execute, branch and register units. Fetch/IMem includes fetch uni, program memory (holding code up to maximum 8kB) and switch box. Two bytes can be fetched from program memory and the switch box route the instruction to decode unit. Optimization of the fetch loop gives the average throughput of 1.37bytes/cycle.

The decode unit is decomposed into many processes (control0 and control1) which consumes dynamic energy as per requirement. Control0 decode the first byte of instruction opcode and the next two byes (if any) are decoded by control1. The decoded bytes are sent to an appropriate execution unit. Lutonium stops all switching activities in deep sleep mode and wakes up without any delay from the deep sleep mode. In the deep sleep mode, only counters operate. Lutonium uses 4-phase handshake protocol and

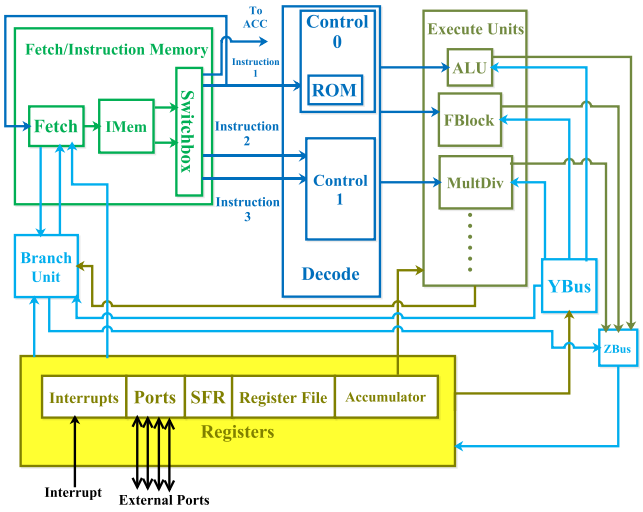


FIGURE 20. Block diagram of Lutonium: reprinted from [92].

Quasi-delay-insensitive timing model. The performance of Lutonium prototype implementation using TSMC SCN018 0.18 μm CMOS process by MOSIS at 1.8V was estimated 200MIPS (1800MIPS/W).

22) MODELLING SAMIPS

SAMIPS [93], [94], a synthesizable asynchronous MIPS processor, is based on the MIPS application architecture. The main purpose of the design was to use it as a test case in an integrated formal verification and distributed simulation environment [95]. The pipeline in SAMIPS consist of five stages: instruction fetch, instruction decode, execution, memory, and a write-back stage. In instruction decode stage, the read/write register operation is performed. The instruction decode unit, based on asynchronous design, checks six MSB and five LSB (in R-type only) to generate control signals bundled with data. The processor handles data hazards using forwarding mechanism which is based on history information recorded in DHdetection unit inside the register bank.

The execution unit includes ALU (without multiplication and division operation), a functional unit for branch test, a shifter, color matching mechanism and ForWarding mechanism (FW) unit. For control hazards, SAMIPS uses coloring mechanism that is first used in AMULET1 [69]. In this mechanism, one bit is used to represent the state of the processor as well as instructions at a particular moment. When the instruction and processor color bit mismatch, the instruction is discarded. The processor color bit changes on the termination of instruction stream. The model of SAMIPS is described in Balsa: a CSP based asynchronous hardware description language and synthesis tool with LARD [57] is used for behavioral simulation.

23) SENSOR NETWORK ASYNCHRONOUS PROCESSOR

Sensor Network Asynchronous Processor/Low Energy (SNAP/LE) [96] is an ultra low-power processor for sensor networks. It is a 16-bit data-driven RISC core processor based on ISA of SNAP [97] (MIPS architecture) and optimized for data monitoring in sensor network. It has extremely low power idle state and very fast wake-up response. The target sensor node remains idle most of the time which makes the asynchronous technique as the best choice for processors for that types of nodes. The asynchronous processor design has hardware support for commonly-occurring sensor network operations. The low power consumption of the processor maximizes the network lifetime. The SNAP core includes event queue, instruction fetch, decode, execute units, buses, register file, message FIFO's and memories (two on-chip 4KB memory banks).

The execution units include adder, logic unit, load-store unit (for memory), timer unit (for timer coprocessor interfacing), jump branch unit, a linear-feedback shift register and a shifter. Two types of buses are commonly used: a fast and a slow bus. The execution units in sensor networks are placed on a fast bus. The SNAP architecture is completely event-driven and remains idle until external event hit event queue. After completion of an event, the DONE instruction halts the processor until the next event appears on an event queue. A 4-phase protocol and Quasi-delay insensitive timing model was adopted for asynchronous circuits. The processor shows the performance of 200MIPS @ 1.8V consuming $\approx$ 218pJ/instruction while using 0.18μm TSMC process.

24) BITSNAP

BitSNAP [98] is a dynamic significance compression for a low-energy sensor network asynchronous processor based on SNAP ISA [97], [96]. It uses bit-serial data-path with dynamic significance compression for achieving low energy consumption. The processor is proposed as a logical extension of SNAP/LE [96] processor. On a bit-serial data stream, BitSNAP employed dynamic adaptive compression known as length adaptive, making the processor a length adaptive data path processor. For each word, the delimiter bit and bits prior it are only sent instead of 16-bit word which reduces switching activity. The architecture of BitSNAP is similar to SNAP/LE with some modification: dynamic significance compression and parallel bit data path conversion to serial bit data path. All execution units, register file and all data path bus split and merge units are bit-serial units operating on LAD digits. The memories, ShareI (shares input word with decode unit or data path) and fetch unit are bit parallel circuits. The BitSNAP processor uses special hardware for interfacing bit parallel data to bit serial units and interfacing LAD data to bit parallel units. In 0.18μm CMOS process, the expected speed of BitSNAP presented by designer was 6 and 54MIPS while consuming 17pJ/ins at 0.6V and 152pJ/ins at 1.8V respectively.

25) HT80C51

HT80C51 microcontroller [99], an asynchronous implementation of 80C51, is a commercial product by Handshake Solutions (Philips). It is functionally compatible with the

instruction set and peripherals of 80C51 with some unique features. These features include extremely low power, low electromagnetic emission, low supply-current peaks, zero standby power with immediately wake-up, asynchronous and optional synchronous mode. Single process HT80C51 execute an instruction in sequential phases fetch, decode, read, execute and the write phase. The structure of the microcontroller is based on Harvard architecture and the instructions are variable length (one, two or three bytes). The high-level programming language Haste, by Handshake Technology design flow (technology independent), was used for designing microcontroller and its peripherals. The HT80C51 non-pipelined asynchronous microcontroller was realized in 0.14μm CMOS where transistor count was 30820 with performance (worst case) of 8.9 MIPS @ 1.8V, 0.7mW [100].

26) ASYNCHRONOUS 8051 MICROCONTROLLER CORE

A8051 [101] is an asynchronous implementation of Intel 8051 microcontroller for low voltage and low energy applications (hearing aid). The designers of the microcontroller used a number of techniques for yielding low power dissipation. To minimize system activity, they used two-stage asynchronous pipeline including instruction fetch and decode-execute stages that operate independently. The design they offer is without predictive approach and include indirect RAM access. Using partial decoding algorithm, the most significant nibble of instruction is decoded to identify type of operation and the least significant nibble identifies the addressing mode. As is common in microcontrollers, A8051 is comprised of registers, latches and decoder based memory altogether contribute to larger area. To reduce area of proposed A8051 and enhance the performance, the designer proposed methodology for interfacing asynchronous system synchronous IP memory blocks (RAM and ROM) [102].

The proposed 8-bit asynchronous microcontroller contains 4K×8 ROM as an instruction memory and 128×8 RAM as a data memory with Harvard architecture. Its design was completed in Balsa as an electronic design automation tool. The microcontroller 8051 was fabricated within the dual core microcontroller system DC8051 [103], [104] with two modes of operation: synchronous operation is based on Synopsys DW8051 IP core whereas asynchronous mode is based on A8051. The cores share 1kbyte ROM and 128byte RAM as well as 1kbyte external RAM. The DC8051 system was implemented using 130nm CMOS and the measured performance of A8051 using Dhrystone v2.1 benchmark reported as 7.4MIPS consuming 349 pJ/I.

27) VORTEX PROCESSOR

The Vortex [105] processor is based on a superscalar asynchronous processor design. Vortex CPU supports 32-bit integer data path and execute up to nine instruction per cycle. The Vortex architecture prototype is shown in fig. 21. It includes dispatcher (instruction decoder and control signal generator), a crossbar (input/output router) and functional units. All the

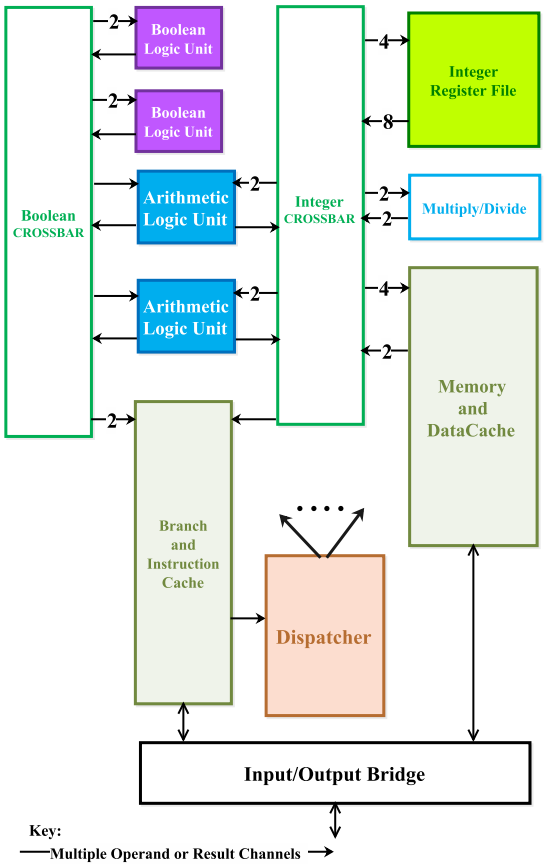


FIGURE 21. Vortex prototype: reprinted from [105].

parallel functional units communicate through central crossbar, instead of a register file. Each instruction consists of two parts: prefix and body. The prefix of instruction is used by the dispatcher for choosing a destination of the instruction: a specific functional unit or crossbar. The asynchronous low-level circuitry is based on the "integrated pipelining" templates [25]. It was fabricated as a part of Testchip2 realized using 0.15 μmG process by TSMC.

28) ARM996HS PROCESSOR

ARM996HS [106], the first licensable and clockless processor core, is a 32-bit RISC type asynchronous processor core implemented using Harvard architecture. The processor is fully compatible with ARMv5TE (ISA), debug architecture and supports 16-bit Thumb instruction set. The ARM996HS processor has 5-stage integer pipeline that includes fetch, decode, execute, memory and write-back stages. All these stages are connected with pipeline control unit. It has a 32-bit fast multiply-accumulate (MAC) block, divide coprocessor and tightly coupled memory. The memory protection unit and non-maskable interrupts provision are used for specific security enhancement.

Factors such as low electromagnetic emission, ultralow power and high robustness converge are the principals of design that were successfully achieved. The compiled code

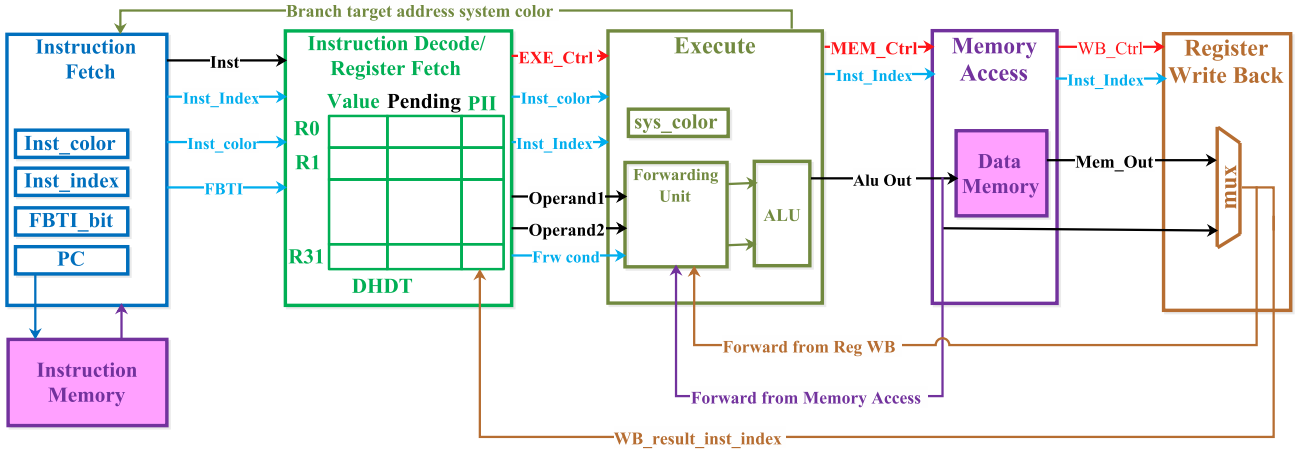


FIGURE 22. AsynRISC processor microarchitecture: reprinted from [107].

for ARM9E CPUs family can be run on ARM996HS. The Processor was implemented using handshake technology, Timeless Design Environment (TiDE) design flow, based on Haste high-level design entry language (formerly known as Tangram). The TiDE design flow is a frontend to synchronous EDA tools. The Tiempo handshake interface is used to adapt changes in environmental conditions such as supply current, voltage and temperature. The performance measured under worst condition was 54 DMIPS (1.08V, 125°C) and with nominal condition (1.2V, 25°C) was 83 Dhrystone MIPS. These statistics are based on netlist simulation (post layout) by using the Artisan Sage-X library for the 0.13μm TSMC process.

29) TAM16 MICROCONTROLLER

TAM16 [51] is a 16-bit clockless microcontroller IP core by Tiempo. It had complete and power efficient instruction set along with adapted software development kit for ultra-low power application. The software development kit include a linker, assembler, instruction set debugger and simulator. To make its instruction set binary compatible with other customers' microcontroller, the instruction set can be customized easily. Two memory interfaces, 1 UART, 3 cascadable timers, 16-bit Programmable Input/Output (PIO), interrupt controller and boot configurations pins are embedded peripherals in TAM16.

In Tiempo technology, the IP is designed for ultra low noise, ultra low power consumption and ultra-low EMI. These features make Tiempo, the fully asynchronous and DI processor, robust against fault injections. It is a commercial product for ultra-low power embedded electronics chips. TAM16, for example, is used in RFID tags, sensor networks, smart cards, e-metering devices and for low electromagnetic emission chips. The low electromagnetic emission chips are used in medical, aeronautics and automotive industries. TAM16 is available as a place and route silicon proven Verilog-netlist. By using CMOS 130nm technology, the TAM16 is designed and processed as test chip. It shows the performance of 7.1 and 15.5MIPS at 0.7V and 1.2V, respectively. The consumption of core is 33.4 and 49μA/MIPS at 0.7V and 1.2V, respectively.

30) AsynRISC

AsynRISC [107] is an asynchronous pipelined processor with instruction set similar to MIPS R2000. It has five pipeline stages as shown in fig. 22. The pipeline stages include instruction fetch, instruction decode and register fetch, instruction execution or memory address calculation, memory access and register write back. Control hazards are solved using two one-bit registers *instcolor* (in instruction fetch stage) and *syscolor* (in execution stage). As all control transfer takes place in execution stage, in IF stage, every new instruction proceeds to next stages after attaching color bit of the *instcolor* register. The color bit is later checked by execution stage by matching the color bit with *syscolor* register. If the color bits match, the instruction is executed normally, otherwise, it is discarded.

Data hazards are resolved by adding two extra fields in every general purpose register. A pending bit indicates register is up-to-date or waiting for new contents. Two bits, known as pending instruction index, records the instruction which produces the new contents for a register. At IF stage, the 2-bit instruction index register provides an index to every new instruction. As at most four instruction reside in datapath, two bits are enough for this register. It was designed and verified using Balsa asynchronous hardware description language and Balsa simulation system respectively. The performance was measured by executing a particular program having 500 dynamic instruction taking 21256799 unit[3] execution time.

31) A8051v2

A8051v2 [108], the second version of asynchronous 8051 [91], is a low-power implementation of asynchronous

[3]As unit is not mentioned, the assumed unit is ns.

8051 employing adaptive pipeline structure. While there are many dissimilarities in system architecture and instruction execution scheme, the instruction set architecture of the proposed design is fully compatible with Intel 8051 [109]. Among the major changes are inclusion of additional features for multi-cycle instruction. These features are multi-looping control, branch prediction (for unconditional branches) and single threading (in the execution stage). It was realized using adaptive micropipeline for skipping and combining pipeline stages for gaining power efficiency and performance. Stage skipping and combining mechanisms are controlled by adding extra inputs i.e EL_N for latch controller and EC_N for pipeline stage bundled with the latched data. The decision, whether or not to skip the operation of N^{th} pipeline stage, is taken by an EC_N signal. Furthermore, the decision if the N_{th} latch is transparent or not, is determined by an EL_N signal. The A8051v2 was simulated with NanoSim tool and mapped into Hynix 0.35 μm CMOS technology with a nominal voltage of 3.3V. A8051v2 @ 3.3V shows the performance of 84.2 MIPS (2316MIPS/W) measured by executing Dhrystone V2.1 benchmark.

32) PA8051

PA8051 [110] is a pipelined asynchronous 8051 soft-core microcontroller implemented in description language Balsa. Its design consists of 5-stage pipeline as shown in fig. 23. The pipeline stages include instruction fetch, instruction decode, operand fetch, execution stage and write back stages. The design also include a memory unit which is not part of the pipeline. The instruction fetch stage contains ROM interface, fetch controller and two buffers (as instruction cache to the program memory).

The memory unit provides the memory interface to *RAM − READ − ARBITOR* where the arbiter [111] is used to read/write data and read instructions from/to fetch and instruction decode units, respectively. The memory exchange with write back unit is likewise arbitrated by the *MEM − INTERFACE* as shown in the fig. 23. The PA8051 microcontorller uses 4-phase bundled data communication protocol to reduce the area cost. The design of the microcontroller is described in the CSP based asynchronous HDL language Balsa and synthesized into Xilinx netlist. The synthesis was completed with Xilinx ISE for the target device Spartan-IIE 300 ft256 FPGA.

33) NCTUAC18

NCTUAC18 [112] is a quasi-delay-insensitive microprocessor core implementation for microcontrollers. It is an 8-bit asynchronous microprocessor core with an instruction set of PIC18. The 4-stage pipeline of NCTUAC18 include instruction fetch, instruction decode, operand fetch and execution/write back stages. The instruction decode stage includes instruction decode block, branch control block, stall control and NPC control. The instruction decode block generates the control signal for the whole processor and checks whether or not the instruction is a conditional branch. If it is a conditional

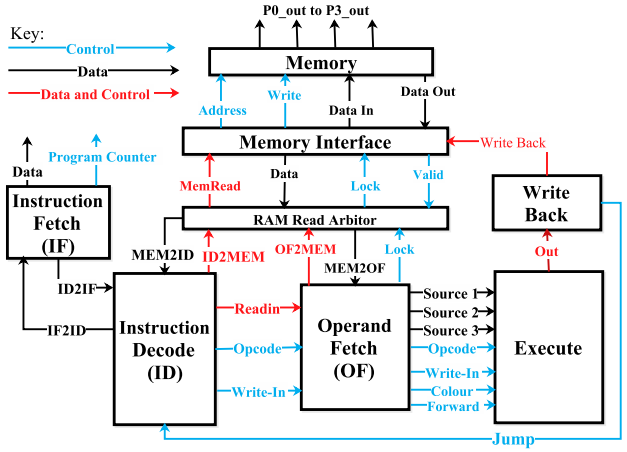


FIGURE 23. PA8051 architecture overview: reprinted from [110].

branch, the instruction decode block requests the branch control block to take over, otherwise, it requests NPC (for next PC value). On conditional branch instruction, the stall control generates request signal to NPC for generating the same PC value to retrieve the instruction. The NPC control is responsible for the correct generation of the PC value. In NCTUAC18, the execution and write back stages are combined in one stage.

The design uses Muller pipeline with 4-phase protocol, dual-rail encoding, quasi-delay insensitive timing model. The proposed design was verified by using ModelSim 6.0 and its gate level design was synthesized using Altera Quartus II software for the target FPGA Altera Cyclone EP1C20F400C8. The maximum path delay in the instruction decode stage, the critical stage of the design, was ≈455ns. The designer admitted that the design deals with branch instruction inefficiently and the quasi-delay insensitive model makes the circuit design difficult. Later on, the NCTUAC18S [113] was introduced with new stall mechanism which handled branch instructions effectively. Furthermore, an acknowledge wire was added to the instruction decode and write back stages. The wire is used to generate an acknowledgement, by the write back stage, on completion of previous instruction. With this modification, the branch instruction is stalled in the instruction decode stage until the acknowledgement is received. The NCTUAC18S was implemented with dual-rail Muller pipeline and 5-stage pipeline with separate execution and write back stages. In the modified design, it is possible to write and read data at the same time. The design was gate-level modeled in hardware description language Verilog, verified with ModelSim 6.0 and synthesized using design compiler with TSMC 0.13μm process library.

34) DRAP

The Dynamically Reconfigurable Asynchronous Processor (DRAP) [114], [115], is a processor based on novel clocked architecture called RICA [116]. The design main goal was to make an architecture that fulfill the demand of

TABLE 2. Operational cells in sample array. Reprinted from [114].

Cell	Count	Cell	Count	Cell	Count	Cell	Count
ADD/COMP	65	LOGIC	20	MUL	20	MUX	65
REG	134	MEM	4	SHIFT	35	JUMP	1
CONST	40	SUBF	16				

high-throughput mobile applications for energy efficiency and programmability using high-level languages. The DRAP processor consists of a heterogeneous array of course-grain asynchronous cells that are implemented using a reconfigurable data-path architecture. An abstract and comprehensive architecture view is presented in [114] and [117]. The design is based on operational cells where each interconnection of operational cells performs limited operations such as logic operation, addition and multiplication. The interconnection design for sample array is based on island-style structure as set-up in standard FPGA's [118]. Configurable routing switches are assembled around the operation cells to allow each cell to interface with its nearest four neighbors. Assembling routing switches, in addition, assist handshake signals and execute conditional acknowledge synchronization using technique presented in [119]. The sample array in DRAP contains 18-bit 400 asynchronous operational cells as listed in Table 2. These cells are interconnected using switches that are based on multiplexer.

Different blocks of instructions are executed by changing the operational cells and interconnects configuration similar to the architecture of CPU. For general application, an adequate mixture of the cells is selected manually while for a specified application other specific cells can be selected. Integrated circuits and the interconnect switches are controlled by the configuration bits stored in the program memory. A total of 9260 configuration bits are required for the reconfigurable core with the selected type of operational cells and interconnects. The program and data memories are interconnected to each other by using special cells of the core. The 4-phase single rail handshake protocol was adopted for the design of operational cells. A network of programmable switches plays a role of interconnection for data-path creation.

Using a UMC 0.13μm technology, the sample array was realized and compared with the architectures Custom RICA 400 [116] (0.13μm), ASIC (0.13μm), ARM7TDMI-S [120] (0.13μm) and TIC64x 8-way VLIW [121]. The algorithms bilinear demosaicing [122], 8K-point radix-2 1-D FFT [123] and 2-D DCT [124] are the benchmarks for the evaluation of the design. For the same throughput, the power consumption of each design was calculated for each benchmark. The power and area rating of the Custom RICA 400, sample DRAP and ASIC design were originated using PrimePower (from Synopsys) post-layout simulation. The ratings for ARM7TDMI-S core and TIC64x are provided in [120] and [125], respectively. All these ratings are measured @1.2V where the energy ratings are measured only for data-path without a memory. Comparison results are listed in Table 3.

TABLE 3. Comparison of DRAP. Reprinted from [114].

Algorithm	Throughput	DRAP	RICA	ASIC	ARM7	TIC64
		Power mW				
Bilinear Demosaicing	23.6MP/s	35.2	41	11.4	326.4	408
1-D FFT	100Mb/s	6.7	7.2	2.3	52.6	132
2-D DCT	45Mb/s	4.8	5.3	1.5	31.8	57.2

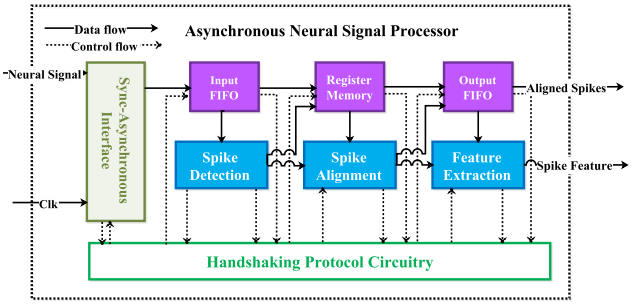


FIGURE 24. Block diagram of asynchronous neural signal processor. Reprinted from [126].

35) ASYNCHRONOUS NEURAL SIGNAL PROCESSOR

Asynchronous neural signal processor [126] is a 0.25V 460nW processor with inherent leakage suppression design for spike-sorting function. The spike sorting function was completed in three steps in this processor: spike detection, alignment and feature extraction. The algorithm employed for spike sorting in this design exhibits best suitable power-density characteristic for wireless neural signal processing in real-time [127]. The processor receives 8-bit digital data from a neural signal acquisition front-end running at 20kHz. The synchronous-asynchronous interface converts the synchronous input into 4-phase dual-rail data. All modules communicate using 4-phase dual-rail handshaking protocol. The asynchronous neural signal processor block diagram is represented in fig. 24. Both versions of the processor, synchronous and asynchronous, were realized in a 65nm CMOS for performance comparison. The asynchronous version prototype shows the 2.3x reduction in power.

36) uaMIPS

The Micro-Watt Asynchronous MIPS (uaMIPS) [128], a sub-threshold ultra-low power processor, is an asynchronous implementation of 8-bit 5-stage conventional synchronous MIPS processor. Designed for a benchmark purpose, the instruction and data memories are based on flip-flop to simplify its design. Using a pipeline oriented de-synchronization tool [129], the synchronous version was converted to 4-phase bundled data asynchronous version. The unavailable asynchronous elements in the standard cell library were manually inserted using different techniques. The asynchronous design flow is shown in fig. 25. Quasi-Delay-Insensitive (QDI) implementation was created in System Verilog CSP and proteus backend flow [130]. The QDI is not much attractive approach towards ultra-low-power design because of its performance/power ratio. The proposed

TABLE 4. Evaluation of available asynchronous processors.

Processor	Architecture	Protocol	Technology	Number of Transistors	Estimated Performance
CAM (1989) [40]	3-SP[1],16b[2]R[3]	4P[4]/DR[5]/QDI[6]	2µm and HP 1.6µmC[7]	2000	15M[8]@7V, 26M @ 10V and 105mA
FAM (1992) [65]	4-SP,32b R	4P/DR/DI[9]	0.5µm C	71000	300M
NSR (1993) [66]	5-SP,16b R	2P[10]/BD[11]/bd[12]	Actel FPGA (C)	NM[24]	1.3M
CFPP (1994) [67]	SPARC	2P/BD	Not fabricated	NM	NM
AMULET1 (1994) [68]	6-SP, based on 32b ARM6	2P bd	1µm C	58374	12.8DM (20.5K Dhrystone)
			0.7µmC	58374	22.7DM (40K Dhrystone)
TITAC (1994) [70]	No pipelining, 8b vN-M[13]	4P/QDI	1µmC	≈22068	11.2M (Controller 1),1.8M (Controller 2)
GaAs AM (1994) [71]	16b R	4P/DR/QDI	GaAs DCFL	NM	50M/W
ECSTAC (1995) [84]	Heavily pipelined 8b R like ISA	2P/bd/fundamental mode	0.7µmC	26000	28M
Fred (1996) [72]	Motorola 88100	2P/BD	VHDL model	NM	149.67M
HADES (1996) [73] [74]	4-SP	4P	VHDL model	NM	NM
MiniMIPS (1997) [41]	based on 3-SP, 32b MIPS R3000	4P/1ofN/DR/QDI	HP 0.6µmC	2000000 (For caches 1.25M)	180M @ 4W 3.3V, 100M @ 850mW 2.0V, and 60M @ 220mW 1.5V all at 25°
TITAC-2 (1997) [80]	5-SP, based on 32b MIPS R2000	2P/DR(cache and external bus uses BD)	0.5µmC	496367with 8.6Kbyte (cache memory)	52.3VAX M Dhrystone V2.1 benchmark @ 2.11W,3.3V
ASYNMPU (1997) [81]	Micropipeline	2P/BD	0.6µmC	31750(Gate count)	Equivalent to a 33MHz Intel 8/16b microprocessor
ARISC (1998) [86]	Deeper than 3-SP	4P/BD	0.35µmC	NM	74M @ 635M/W, 3.3V
ASPRO-216 (1998) [87]	Heavily pipeline 16b R	DR/QDI	0.25µmC	500,000	140(peak)M(v = 2.5)
Asynchronous 80C51 (1998) [36]	not pipelined, based on 8b 80C51 microcontroller (CISC)	4P/BD	0.5µm C	27482(CPU V3)	2.10M, 943M/W (CPU V3)
AMULET2e (1999) [79]	6-SP, 32b ARM	4P/BD	0.5µmC	454000	42M @ 280 M/W
AMULET3 (2000) [89]	6-SP	4P/BD	0.35µmC	113000	120M @ 780M/W,3.3V
A8051v1 (2002) [91]	5-SP, based on CISC architecture	4P/DR/DI	0.35µmC	104000	75.5M
Lutonium (2003) [92]	Pipelined, 8b 8051 (CISC)	4P/QDI	0.18µmC	NM	200M @ 1.8V, 1800M/W
SAMIPS (2003) [93] [94]	based on 5-SP MIPS architecture	2P/BD	BALSA [133]	NM	NM
SNAP/LE (2004) [96]	Based on 16b SNAP (ISA) [97]	4P/DR/ QDI	0.18µmC	62K + 325K	28M @ 0.6V (≈ 24pJ/I)
			0.18µm C	62K + 325K	200M @ 1.8V (≈ 218pJ/I)
BitSNAP (2005) [98]	Based on 16b SNAP (ISA) [97]	4P/DR/ QDI	0.18µmC	23.5K	6M @ 0.6V (≈ 17pJ/I)
			0.18µmC	23.5K	54M @ 1.8V (≈ 152pJ/I)
HT80C51 (2005) [99]	Non-pipelined, asynchronous implementation of 80C51	4P/BD	0.14µmC	30820	8.9M @ 1.8V, 0.7mW
A8051 (2006) [101]	2-SP(Asynchronous implementation of Intel 8051)	4P/DR/QDI	130nmC [103]	NM	7.4M @ 349 pJ/I [103]
YUPPIE[14] (2006) [134]	8b Atmel AVR ISA	De-synchronization using [135]	130nm	NM	170M @ 14pJ/I at 1.2V, 48M @ 2.7pJ/I at 0.54V
Vortex (2007) [105]	Superscalar processor	4P/DR/QDI	0.15 µmC	10.5M (6.68 M in logic, 3.82M in cache)	NM
ARM996HS (2007) [106]	5-SP (based on ARMv5TE (ISA))	4P/BD	0.13µmC	89000 gates	54DM (worst condition @ 1.08V, 125° C)
TAM16 (2008) [51]	16b ISA	DR/DI	130nmC	NM	7.1M @ 0.7V (33.4µA/M)
					15.5M @ 1.2V (49µA/M)
AsynRISC (2008) [107]	5-SP, MIPS R2000 like ISA	NM	BALSA [133]	NM	21256799[15]
A8051v2 (2008) [108]	Adaptive Pipeline, based on CISC architecture	4P/DR/DI	0.35µmC	110000	84.2 M @ 3.3V, 36.3mW, 430pJ/I

TABLE 4. *(Continued.)* **Evaluation of available asynchronous processors.**

PA8051 (2008) [110]	5-SP	4P/BD	Xilinx Spartan-IIE 300 ft256 FPGA	123510 (Gate count)	NM
NCTUAC18 (2009) [112]	4-SP(Muller pipelined, compatible with ISA of 8b PIC18)	4P/DR/QDI	Altera Cyclone EP1C20F400C8	NM	NM
Asynchronous Processor (2009) [136]	NM	2P/BD	Xilinx Spartan 3A	NM	≈65M @ 1.8V (320M/W)
DRAP (2010) [114]	based on RICA [116]	4P/BD	UMC 0.13µm	NM	In Table 3
ASPEN (2010) [137]	16b MIPS(modified Harvard architecture)	4P/DR/QDI	ST-microelectronics 90nm	ASPEN Area = 6.14mm^2	@400mV, ALU=9-15pJ/operation, Fetch+Decode ≈ 80pJ/operation
NCTUAC18S (2011) [113]	5-SP (Muller pipelined, compatible with ISA of 8b PIC18)	4P/DR/Muller Pipeline	0.13µmC	Total Area = 85180µm^2	Total Delay time=38,024ps
Opus2 (2012) [138]	DSP core	IMS, IMA[16]	90nm G process technology	2.8mm^2	2000 MMACS ,21 MMACS/mW @1V, Opus3≈8000MMACS @1V
ANSP (2013)[17] [126]	Neural Signal Processor	4P/DR	65nm C)	NM	20kHz spike sampling rate, 23pJ@0.25V
AFPGA[18] (2014) [139]	8b novel architecture	4P/DR/DI	Xilinx Spartan-6 XC6SLX9	Slice logic utilization =548 + 933 + 2020 + 548 @ table IV [139]	Average 34.2 M
uaMIPS (2015) [128]	based on 8-bit MIPS presented in [128]	4P/BD	HP 28nm HKMG,HVT(V_T = 0.6V)	NM	In fig. 26
DD1 (2015) [140]	8b AVR reduced core instruction set (8b data-path, 16b instruction (few 32b))	4P/QDI	TSMC 40LP[19]	Area = 0.37mm^2	20M @ 1.1V,75µW/M (total = 1.5mW) and 1M @ 550mV,18µW/M, 18pJ/instruction and 27M @ 1.2V,80pJ/instruction
The Bel Array (2016) [141]	2D grid of homogeneous asynchronous bit-level array	NCL[20]	28nm	$2*10^6$ bit elements in 400mm^2 chip	With a single-bit Delta Sigma 20-tap SWL FIR shows up to 2800 MSPS[21]@ 1.2V, 42.53mW
PSP-U-FPGA[22] (2016) [142]	Presented in [138], 8b architecture	Self-Timed	Xilinx 7 Series with Vivado tools	NM	11.1M(with Fibonacci algorithm)
TbSTP (2017)[23] [143]	Presented in [138]	Self-Timed	Xilinx ZYNQ ZC706	NM	53.26M(with Fibonacci benchmark) while consumes 173.4mW @ 25% runtime
AMSP430 (2017) [131]	based on MSP430	4P/BD	IBM 65nm	NM	10.5x power efficient than its synchronous implementation, 15417ns average run time with different test benches

[1] SP = Stage-Pipeline.
[2] b = bit.
[3] R = RISC (Reduced Instruction Set Computer).
[4] 4P = 4-Phase.
[5] DR = Dual Rail.
[6] QDI = Quasi-Delay-Insensitive.
[7] C = CMOS (Complementary Metal Oxide Semiconductor).
[8] M = MIPS (Million Instructions Per Second).
[9] DI = Delay Insensitive.
[10] 2P = 2-Phase.
[11] BD = Bundled Data.
[12] bd = bounded delay.
[13] vN-M = von Neumann Microprocessor.
[15] 500 dynamic instruction takes 21256799 execution time we assume SI unit n for execution time.
[14] YUPPIE = Yet another Ultra-low Power asynchronous Processor for wIreless sensor nEtwork.
[16] IMS = Intra module(source synchronous), IMA = Inter module(Asynchronous request with matched acknowledge done).
[17] ANSP = Asynchronous Neural Signal Processor.
[18] AFPGA = Asynchronous 8-bit processor mapped into an FPGA device.
[19] 40LP = A low-power bulk 40nm CMOS process.
[20] NCL = Null Convention Logic.
[21] MSPS = Mega Samples per Second.
[22] PSP using FPGA = Prototyping Self-timed Processors using FPGAs.
[23] TbSTP= Token based Self-Timed Processor.
[24] NM = Not Mentioned.

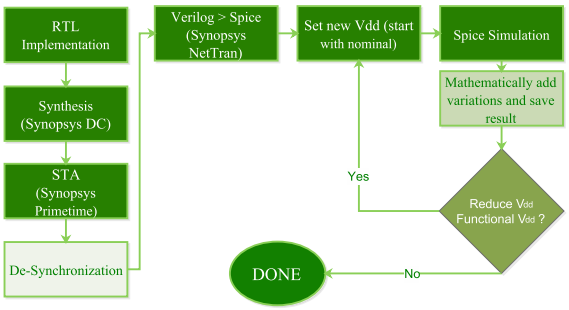


FIGURE 25. Asynchronous design flow. Reprinted from [128].

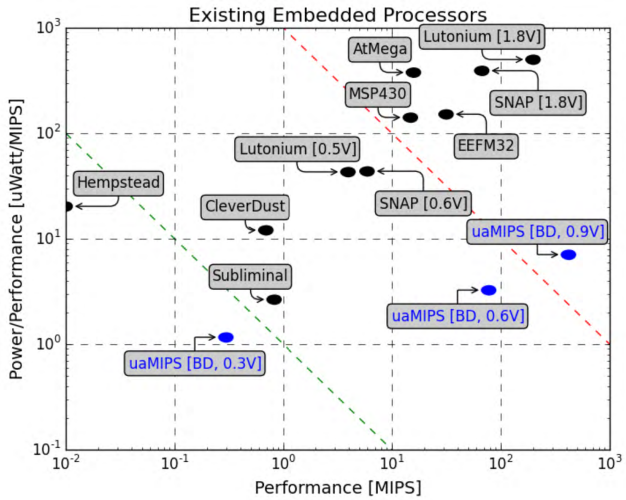


FIGURE 26. Comparison between uaMIPS and other ultra-low-power processor. Reprinted from [128].

uaMIPS asynchronous (bundled data) processor in 28nm HKMG, HVT(V_T=0.6V) shows better power efficiency as shown in fig. 26.

37) ANSYNCHRONOUS MSP430

Asynchronous MSP430 [131] design is a low power and relative timing-based asynchronous MSP430 microprocessor. It is asynchronous implementation of openMSP430 [132] 16-bit processor with RISC type ISA. The design of MSP430 had two directly connected finite state machines: decode and execute. In an asynchronous implementation, the data-path is nearly identical to its parent design. A new conjunctive stateful communication method is employed between the asynchronous finite state machines.

The MSP430 microprocessor uses 4-phase bundled data protocol with relative timing methodology as described in [131]. Both designs, asynchronous and synchronous, are designed in the same computer-aided design (CAD) tool and synthesized using the same IBM 65nm 10SF node with same EDA tools and scripts. A comparison shows that the synchronous design consumes 5% more area than asynchronous design. The asynchronous design is on the average 33% slower, consumes less than one-tenth the power and consumes one-seventh the energy per operation as compared to the synchronous. These statistics are concluded after executing different benchmark programs. Furthermore, asynchronous implementation of openMSP430 shows an improvement in power dissipation.

C. DISCUSSION

Starting with Martin's [40], we have investigated a number of asynchronous microprocessors on abstract level, and compiled their summaries into one document. During this work, we have observed that most of the designers implemented an equivalent asynchronous version of one of the available synchronous benchmark processors, such as MIPS and ARM etc, and they had adopted various specification methods, and tools. Following are some observations that we have made; this may be considered as conclusion of conducting this work.

1) Most of the proposed asynchronous microprocessors are pipelined architectures.
2) Specifically talking of the pipelined processors, most of the designers used a different number of pipeline stages to resolve data and control hazards. Some proposed their novel schemes by claiming that the synchronous methods were not directly applicable to asynchronous logic – mainly due to its distributed control nature.
3) From Table 4, one can observe that pipeline stages, technology, and voltage directly affect the performance.
4) AMULET3 [89], Lutonium [92] and SNAP/LE [96] showed better performance and power ratio, in comparison to others. This is evident in Table 4.

IV. CONCLUSION

We have elaborated on asynchronous logic design principles, along with their available electronic design and automation tools support for specifying, modeling, synthesizing, and implementing asynchronous circuits and systems. The main objective of the work, beside collecting most of the contributions towards designing asynchronous microprocessors, is defining the asynchronous design flow and summarizing the available tools, which, to the best of our knowledge, have been misunderstood or mostly overlooked. We have presented an entire encyclopedia of general, as well as, special purpose asynchronous microprocessors ever developed, irrespective of their classification, signaling mechanisms, architectures, and process. We have presented a thorough evaluation of those processors in terms of performance and area utilization.

REFERENCES

[1] S. R. Naqvi, "A non-blocking fault-tolerant asynchronous networks-on-chip router," Ph.D. dissertation, Inst. Comput. Eng., Vienna Univ. Technol., Vienna, Austria, 2013.
[2] S. Naqvi, "An asynchronous router architecture using four-phase bundled handshake protocol," in *Proc. Int. Multi-Conf. Comput. Global Inf. Technol. (IARIA)*, 2012, pp. 200–205.
[3] S. R. Naqvi, V. S. Veeravalli, and A. Steininger, "Protecting an asynchronous NoC against transient channel faults," in *Proc. IEEE 15th Euromicro Conf. Digit. Syst. Design (DSD)*, Sep. 2012, pp. 264–271.
[4] J. Sparsø and S. Furber, *Principles of Asynchronous Circuit Design: A Systems Perspective*. Norwell, MA, USA: Kluwer, 2001.
[5] A. Kondratyev and K. Lwin, "Design of asynchronous circuits by synchronous CAD tools," in *Proc. ACM 39th Annu. Design Autom. Conf.*, 2002, pp. 411–414.

[6] B. Rahbaran and A. Steininger, "Is asynchronous logic more robust than synchronous logic?" *IEEE Trans. Dependable Secure Comput.*, vol. 6, no. 4, pp. 282–294, Oct. 2009.
[7] S. R. Naqvi, R. Najvirt, and A. Steininger, "A multi-credit flow control scheme for asynchronous NoCs," in *Proc. IEEE 16th Int. Symp. Design Diag. Electron. Circuits Syst. (DDECS)*, Apr. 2013, pp. 153–158.
[8] W. A. Clark, "Macromodular computer systems," in *Proc. ACM Spring Joint Comput. Conf.*, Apr. 1967, pp. 335–336.
[9] M. J. Stucki, S. M. Ornstein, and W. A. Clark, "Logical design of macromodules," in *Proc. ACM Spring Joint Comput. Conf.*, Apr. 1967, pp. 357–364.
[10] A. J. Martin, "The limitations to delay-insensitivity in asynchronous circuits," in *Beauty is Our Business*. New York, NY, USA: Springer, 1990, pp. 302–311.
[11] K. van Berkel, "Beware the isochronic fork," *Integr., VLSI J.*, vol. 13, no. 2, pp. 103–128, 1992.
[12] N. Toosizadeh, "Enhanced synchronous design using asynchronous techniques," Ph.D. dissertation, Dept. Elect. Comput. Eng., Univ. Toronto, Toronto, ON, Canada, 2010.
[13] R. Najvirt, S. R. Naqvi, and A. Steininger, "Classifying virtual channel access control schemes for asynchronous NoCs," in *Proc. IEEE 19th Int. Symp. Asynchronous Circuits Syst. (ASYNC)*, May 2013, pp. 115–123.
[14] A. Peeters and K. van Berkel, "Single-rail handshake circuits," in *Proc. IEEE 2nd Work. Conf. Asynchronous Design Methodol.*, 1995, pp. 53–62.
[15] J. Bainbridge, *Asynchronous System-on-Chip Interconnect*. London, U.K.: Springer, 2013.
[16] T. Verhoeff, *A Theory of Delay-Insensitive Systems*. Eindhoven, The Netherlands: Eindhoven Univ. Technology, 1994.
[17] D. E. Muller, "A theory of asynchronous circuits," in *Proc. Int. Symp. Theory Switching*, Apr. 1959, pp. 204–243.
[18] K. M. Fant and S. A. Brandt. *Null Convention Logic*. Accessed: Jan. 10, 2019. [Online]. Available: http://www.theseusresearch.com/NCLPaper01.htm
[19] K. M. Fant and S. A. Brandt, "Null convention logic: A complete and consistent logic for asynchronous digital circuit synthesis," in *Proc. IEEE Int. Conf. Appl. Specific Syst., Archit. Processors (ASAP)*, Aug. 1996, pp. 261–273.
[20] G. E. Sobelman and K. Fant, "CMOS circuit design of threshold gates with hysteresis," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, vol. 2, May/Jun. 1998, pp. 61–64.
[21] M. E. Dean, T. Williams, and D. Dill, "Efficient self-timing with level-encoded 2-phase dual-rail (LEDR)," in *Proc. Adv. Res. VLSI*, 1991, pp. 55–70.
[22] I. E. Sutherland, "Micropipelines," *Commun. ACM*, vol. 32, no. 6, pp. 720–738, Jun. 1989.
[23] M. Singh and S. M. Nowick, "MOUSETRAP: Ultra-high-speed transition-signaling asynchronous pipelines," in *Proc. IEEE ICCD*, Sep. 2001, p. 0009.
[24] I. Sutherland and S. Fairbanks, "GasP: A minimal FIFO control," in *Proc. IEEE ASYNC*, Mar. 2001, pp. 46–53.
[25] A. M. Lines, *Pipelined Asynchronous Circuits*. Pasadena, CA, USA: California Inst. Technol., 1998.
[26] R. O. Ozdag and P. A. Beerel, "High-speed QDI asynchronous pipelines," in *Proc. IEEE 8th Int. Symp. Asynchronous Circuits Syst.*, 2002, pp. 13–22.
[27] C. E. Molnar, I. W. Jones, W. S. Coates, J. K. Lexau, S. M. Fairbanks, and I. E. Sutherland, "Two FIFO ring performance experiments," *Proc. IEEE*, vol. 87, no. 2, pp. 297–307, Feb. 1999.
[28] W. P. Burleson, M. Ciesielski, F. Klass, and W. Liu, "Wave-pipelining: A tutorial and research survey," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 6, no. 3, pp. 464–474, Sep. 1998.
[29] O. Hauck and S. A. Huss, "Asynchronous wave pipelines for high throughput datapaths," in *Proc. IEEE Int. Conf. Electron., Circuits Syst.*, vol. 1, Sep. 1998, pp. 283–286.
[30] B. D. Winters and M. R. Greenstreet, "A negative-overhead, self-timed pipeline," in *Proc. IEEE 8th Int. Symp. Asynchronous Circuits Syst.*, 2002, pp. 37–46.
[31] S. R. Naqvi, J. Lechner, and A. Steininger, "Protection of Muller–Pipelines from transient faults," in *Proc. IEEE 15th Int. Symp. Qual. Electron. Design (ISQED)*, Mar. 2014, pp. 123–131.
[32] K. van Berkel, *Handshake Circuits: An Asynchronous Architecture for VLSI Programming*, vol. 5. Cambridge, U.K.: Cambridge Univ. Press, 1993.
[33] A. Peeters, "Single-rail handshake circuits," Ph.D. dissertation, Dept. Math. Comput. Sci., Eindhoven Univ. Technol., Eindhoven, The Netherlands, 1996.
[34] K. van Berkel, R. Burgess, J. Kessels, A. Peelers, M. Roncken, and F. Schalij, "A fully asynchronous low-power error corrector for the DCC player," *IEEE J. Solid-State Circuits*, vol. 29, no. 12, pp. 1429–1439, Dec. 1994.
[35] K. van Berkel *et al.*, "A single-rail re-implementation of a DCC error detector using a generic standard-cell library," in *Proc. IEEE 2nd Work. Conf. Asynchronous Design Methodol.*, May 1995, pp. 72–79.
[36] H. van Gageldonk, K. van Berkel, A. Peeters, D. Baumann, D. Gloor, and G. Stegmann, "An asynchronous low-power 80C51 microcontroller," in *Proc. IEEE 4th Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, Mar./Apr. 1998, pp. 96–107.
[37] J. Kessels, T. Kramer, G. den Besten, A. Peeters, and V. Timm, "Applying asynchronous circuits in contactless smart cards," in *Proc. IEEE 6th Int. Symp. Adv. Res. Asynchronous Circuits Syst. (ASYNC)*, Apr. 2000, pp. 36–44.
[38] J. Kessels, T. Kramer, A. Peeters, and V. Timm, "DESCALE: A design experiment for a smart card application consuming low energy," in *Proc. Eur. Low Power Initiative Electron. Syst. Design*, 2001, pp. 247–262.
[39] A. J. Martin and C. D. Moore, "CHP and CHPsim: A language and simulator for fine-grain distributed computation," Dept. Comput. Sci., California Inst. Technol., Pasadena, CA, USA, Tech. Rep. CS-TR-1-2011, 2011.
[40] A. J. Martin, S. M. Burns, T. Lee, D. Borkovic, and P. J. Hazewindus, "The design and implementation of an asynchronous microprocessor," Ph.D. dissertation, Dept. Comput. Sci., California Inst. Technol., Pasadena, CA, USA, 1989.
[41] A. J. Martin *et al.*, "The design of an asynchronous MIPS R3000 microprocessor," in *Proc. IEEE 17th Conf. Adv. Res. VLSI*, Sep. 1997, pp. 164–181.
[42] M. Renaudin, P. Vivet, and F. Robin, "ASPRO: An asynchronous 16-bit RISC microprocessor with DSP capabilities," in *Proc. Proc. IEEE 25th Eur. Solid-State Circuits Conf. (ESSCIRC)*, Sep. 1999, pp. 428–431.
[43] R. Manohar and C. Kelly, "Network on a chip: Modeling wireless networks with asynchronous VLSI," *IEEE Commun. Mag.*, vol. 39, no. 11, pp. 149–155, Nov. 2001.
[44] K. A. Boahen, "A burst-mode word-serial address-event link-I: Transmitter design," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 51, no. 7, pp. 1269–1280, Jul. 2004.
[45] G. N. Patel, M. S. Reid, D. E. Schimmel, and S. P. DeWeerth, "An asynchronous architecture for modeling intersegmental neural communication," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 2, pp. 97–110, Feb. 2006.
[46] D. Edwards and A. Bardsley, "Balsa: An asynchronous hardware synthesis language," *Comput. J.*, vol. 45, no. 1, pp. 12–18, 2002.
[47] D. Edwards, A. Bardsley, L. Janin, L. Plana, and W. Toms, "Balsa: A tutorial guide, version V3.5," School Comput. Sci., Univ. Manchester, Manchester, U.K., Tech. Rep., 2006.
[48] Tiempo Secure. *ACC: Asynchronous Circuit Compiler*. Accessed: Dec. 23, 2018. [Online]. Available: http://www.tiempo-ic.com/products/sw-tools/acc.html
[49] A. Yakovlev, P. Vivet, and M. Renaudin, "Advances in asynchronous logic: From principles to GALS & NoC, recent industry applications, and commercial CAD tools," in *Proc. Conf. Design, Autom. Test Eur.*, 2013, pp. 1715–1724.
[50] C. Spear, *SystemVerilog for Verification: A Guide to Learning the Testbench Language Features*. New York, NY, USA: Springer, 2008.
[51] Tiempo Secure. *TAM16: 16-Bit Microcontroller IP Core*. Accessed: Dec. 23, 2018. [Online]. Available: http://www.tiempo-ic.com/products/ip-cores/TAM16.html
[52] U. P. de Catalunya. *Petrify: A Tool for Synthesis of Petri Nets and Asynchronous Circuits*. Accessed: Dec. 24, 2018. [Online]. Available: http://www.cs.upc.edu/~jordicf/petrify/
[53] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, "Petrify: A tool for manipulating concurrent specifications and synthesis of asynchronous controllers," *IEICE Trans. Inf. Syst.*, vol. E80-D, no. 3, pp. 315–325, 1997.
[54] T. Akram, S. R. Naqvi, S. A. Haider, and M. Kamran, "Towards real-time crops surveillance for disease classification: Exploiting parallelism in computer vision," *Comput. Electr. Eng.*, vol. 59, pp. 15–26, Apr. 2017.
[55] A. Bardsley, L. Tarazona, and D. Edwards, "Teak: A token-flow implementation for the balsa language," in *Proc. IEEE 9th Int. Conf. Appl. Concurrency Syst. Design (ACSD)*, Jul. 2009, pp. 23–31.

[56] G. K. Theodoropoulos, G. K. Tsakogiannis, and J. V. Woods, "Occam: An asynchronous hardware description language?" in *Proc. IEEE 23rd EUROMICRO Conf. New Frontiers Inf. Technol. (EUROMICRO)*, Sep. 1997, pp. 249–256.

[57] P. Endecott and S. B. Furber, "Modelling and simulation of asynchronous systems using the LARD hardware description language," in *Proc. ESM*, 1998, pp. 39–43.

[58] B. Kangsah, R. Wollowski, W. Vogler, and J. Beister, "DESI: A tool for decomposing signal transition graphs," in *Proc. 3rd ACiD-WG Workshop*, 2003, pp. 1–2.

[59] S. Frankild and H. Palbÿl. *Visual STG Lab*. Accessed: Dec. 24, 2018. [Online]. Available: http://vstgl.sourceforge.net/

[60] *WorkCraft*. Accessed: Dec. 24, 2018. [Online]. Available: https://workcraft.org/

[61] Electrical Engineering and NUS Engineering. *Asynchronous High Level Synthesis Tool (VERISYN)*. Accessed: Dec. 24, 2018. [Online]. Available: http://async.org.uk/besst/verisyn/

[62] I. Blunno and L. Lavagno, "Automated synthesis of micro-pipelines from behavioral Verilog HDL," in *Proc. IEEE 6th Int. Symp. Adv. Res. Asynchronous Circuits Syst. (ASYNC)*, Apr. 2000, pp. 84–92.

[63] *DEMO Session*. Accessed: Jan. 2, 2019. [Online]. Available: http://conferences.computer.org/async2007

[64] Handshake-Solution. *Tide—Timeless Design Environment*. Accessed: Jan. 2, 2019. [Online]. Available: http://www.handshakesolutions.com

[65] K.-R. Cho, K. Okura, and K. Asada, "Design of a 32-bit fully asynchronous microprocessor (FAM)," in *Proc. IEEE 35th Midwest Symp. Circuits Syst.*, Aug. 1992, pp. 1500–1503.

[66] E. Brunvand, "The NSR processor," in *Proc. 26th Hawaii Int. Conf. Syst. Sci.*, vol. 1, Jan. 1993, pp. 428–435.

[67] C. E. Molnar, R. F. Sproull, and I. E. Sutherland, "Counterflow pipeline processor architecture," *IEEE Design Test Comput.*, vol. 11, no. 3, p. 48, Jul. 1994.

[68] S. B. Furber, P. Day, J. D. Garside, N. C. Paver, and J. V. Woods, "AMULET1: A micropipelined ARM," in *IEEE Compcon Spring Dig. Papers*, Feb./Mar. 1994, pp. 476–485.

[69] J. V. Woods, P. Day, S. B. Furber, J. D. Garside, N. C. Paver, and S. Temple, "AMULET1: An asynchronous ARM microprocessor," *IEEE Trans. Comput.*, vol. 46, no. 4, pp. 385–398, Apr. 1997.

[70] T. Nanya, Y. Ueno, H. Kagotani, M. Kuwako, and A. Takamura, "TITAC: Design of a quasi-delay-insensitive microprocessor," *IEEE Design Test Comput.*, vol. 11, no. 2, pp. 50–63, Jun. 1994.

[71] J. A. Tierno, A. J. Martin, D. Borkovic, and T. K. Lee, "A 100-MIPS GaAs asynchronous microprocessor," *IEEE Design Test Comput.*, vol. 11, no. 2, pp. 43–49, Jun. 1994.

[72] W. F. Richardson and E. Brunvand, "Fred: An architecture for a self-timed decoupled computer," in *Proc. IEEE 2nd Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, Mar. 1996, pp. 60–68.

[73] C. J. Elston, D. B. Christianson, P. A. Findlay, and G. B. Steven, "Hades-towards the design of an asynchronous superscalar processor," in *Proc. IEEE 2nd Work. Conf. Asynchronous Design Methodol.*, May 1995, pp. 200–209.

[74] C. J. Elston, D. B. Christianson, P. A. Findlay, and G. B. Steven, "HADES—An asynchronous superscalar processor," in *Proc. IEE Colloq. Design Test Asynchronous Syst.*, 1996, p. 10.

[75] C. A. R. Hoare, "Communicating sequential processes," *Commun. ACM*, vol. 21, no. 8, pp. 666–677, 1978.

[76] A. Semenov, A. M. Koelmans, L. Lloyd, and A. Yakovlev, "Designing an asynchronous processor using Petri nets," *IEEE Micro*, vol. 17, no. 2, pp. 54–64, Mar. 1997.

[77] W. C. Holton, "The large-scale integration of microelectronic circuits," *Sci. Amer.*, vol. 237, no. 3, pp. 82–95, 1977.

[78] S. S. Patil, "Cellular arrays for asynchronous control," in *Proc. ACM Conf. Rec. 7th Annu. Workshop Microprogram.*, 1974, pp. 178–185.

[79] S. B. Furber *et al.*, "AMULET2e: An asynchronous embedded controller," *Proc. IEEE*, vol. 87, no. 2, pp. 243–256, Feb. 1999.

[80] A. Takamura *et al.*, "TITAC-2: An asynchronous 32-bit microprocessor based on scalable-delay-insensitive model," in *Proc. IEEE Int. Conf. Comput. Design, VLSI Comput. Processors (ICCD)*, Oct. 1997, pp. 288–294.

[81] J. M. C. Tse and D. P. K. Lun, "ASYNMPU: A fully asynchronous CISC microprocessor," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, vol. 3, Jun. 1997, pp. 1816–1819.

[82] S. R. Naqvi, A. Steininger, and J. Lechner, "An SET tolerant tree arbiter cell," in *Proc. IEEE 19th Int. Symp. Asynchronous Circuits Syst. (ASYNC)*, May 2013, pp. 31–39.

[83] S. R. Naqvi, T. Akram, S. A. Haider, and M. Kamran, "Artificial neural networks based dynamic priority arbitration for asynchronous flow control," *Neural Comput. Appl.*, vol. 29, no. 7, pp. 627–637, 2018.

[84] S. V. Morton, S. S. Appleton, and M. J. Liebelt, "ECSTAC: A fast asynchronous microprocessor," in *Proc. IEEE 2nd Work. Conf. Asynchronous Design Methodol.*, May 1995, pp. 180–189.

[85] S. V. Morton, "Fast asynchronous VSLI circuit design techniques and their application to microprocessor design," Ph.D. dissertation, Dept. Elect. Electron. Eng., Univ. Adelaide, Adelaide, SA, Australia, 1997.

[86] K. T. Christensen, P. Jensen, P. Korger, and J. Sparso, "The design of an asynchronous TinyRISC TR4101 microprocessor core," in *Proc. IEEE 4th Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, Mar./Apr. 1998, pp. 108–119.

[87] M. Renaudin, P. Vivet, and F. Robin, "ASPRO-216: A standard-cell Q.D.I. 16-bit RISC asynchronous microprocessor," in *Proc. IEEE 4th Int. Symp. Adv. Res. Asynchronous Circuits Syst.*, Mar./Apr. 1998, pp. 22–31.

[88] K. van Berkel, J. Kessels, M. Roncken, R. Saeijs, and F. Schalij, "The VLSI-programming language Tangram and its translation into handshake circuits," in *Proc. Conf. Eur. Design Autom.*, Feb. 1991, pp. 384–389.

[89] S. B. Furber, D. A. Edwards, and J. D. Garside, "AMULET3: A 100 MIPS asynchronous embedded processor," in *Proc. IEEE Int. Conf. Comput. Design*, Sep. 2000, pp. 329–334.

[90] ARM. *The THUMB Instruction SET*. Accessed: Jan. 14, 2019. [Online]. Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0210c/CACBCAAE.html

[91] J.-H. Lee, W.-C. Lee, and K.-R. Cho, "A novel asynchronous pipeline architecture for CISC type embedded controller, A8051," in *Proc. IEEE 45th Midwest Symp. Circuits Syst. (MWSCAS)*, vol. 2, Aug 2002, pp. II-675–II-678.

[92] A. J. Martin *et al.*, "The Lutonium: A sub-nanojoule asynchronous 8051 microcontroller," in *Proc. IEEE 9th Int. Symp. Asynchronous Circuits Syst.*, May 2003, pp. 14–23.

[93] Q. Zhang and G. Theodoropoulos, "Towards an asynchronous MIPS processor," in *Advances in Computer Systems Architecture* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2003, pp. 137–150.

[94] Q. Zhang and G. Theodoropoulos, "Modelling SAMIPS: A synthesisable asynchronous MIPS processor," in *Proc. 37th Annu. Symp. Simulation*, 2004, pp. 205–212.

[95] D. Edwards. *An Integrated Framework for Formal Verification and Distributed Simulation of Asynchronous Hardware*. Accessed: Jan. 2, 2019. [Online]. Available: http://www.cs.bham.ac.uk/research/projects/parlard/

[96] V. Ekanayake, C. Kelly, IV, and R. Manohar, "An ultra low-power processor for sensor networks," *ACM SIGOPS Oper. Syst. Rev.*, vol. 38, no. 5, pp. 27–36, 2004.

[97] C. Kelly, IV, V. Ekanayake, and R. Manohar, "SNAP: A sensor-network asynchronous processor," in *Proc. IEEE 9th Int. Symp. Asynchronous Circuits Syst.*, Jun. 2003, pp. 24–33.

[98] V. N. Ekanayake, C. Kelly, and R. Manohar, "BitSNAP: Dynamic significance compression for a low-energy sensor network asynchronous processor," in *Proc. 11th IEEE Int. Symp. Asynchronous Circuits Syst. (ASYNC)*, Mar. 2005, pp. 144–154.

[99] Handshakesolution. *Handshake Solutions HT80C51*. Accessed: Jan. 3, 2019. [Online]. Available: http://www.keil.com/dd/chip/3931.htm

[100] T. van Hoek, "Designing a high-speed asynchronous 80C51 microcontroller," M.S. thesis, Fac. Elect. Eng., Eindhoven Univ. Technol., Eindhoven, The Netherlands, 2008.

[101] K.-L. Chang and B.-H. Gwee, "A low-energy low-voltage asynchronous 8051 microcontroller core," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2006, pp. 3181–3184.

[102] K.-L. Chang, B.-H. Gwee, and Y. Zheng, "A semi-custom memory design for an asynchronous 8051 microcontroller," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2008, pp. 3398–3401.

[103] K.-L. Chang, T. Lin, W.-G. Ho, K.-S. Chong, B.-H. Gwee, and J. S. Chang, "A dual-core 8051 microcontroller system based on synchronous-logic and asynchronous-logic," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2013, pp. 3022–3025.

[104] K.-L. Chang, B.-H. Gwee, J. S. Chang, and K.-S. Chong, "Synchronous-logic and asynchronous-logic 8051 microcontroller cores for realizing the Internet of Things: A comparative study on dynamic voltage scaling and variation effects," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 3, no. 1, pp. 23–34, Mar. 2013.

[105] A. Lines, "The Vortex: A superscalar asynchronous processor," in *Proc. 13th IEEE Int. Symp. Asynchronous Circuits Syst. (ASYNC)*, Mar. 2007, pp. 39–48.
[106] A. Bink and R. York, "ARM996HS: The first licensable, clockless 32-bit processor core," *IEEE Micro*, vol. 27, no. 2, pp. 58–68, Mar./Apr. 2007.
[107] M.-C. Chang and D.-S. Shiau, "Design of an asynchronous pipelined processor," in *Proc. IEEE Int. Conf. Commun., Circuits Syst. (ICCCAS)*, May 2008, pp. 1093–1096.
[108] J. H. Lee, Y. H. Kim, and K. R. Cho, "A low-power implementation of asynchronous 8051 employing adaptive pipeline structure," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 55, no. 7, pp. 673–677, Jul. 2008.
[109] Intel Corporation, *Intel Microprocessor and Peripheral Handbook: Microprocessor*, vol. 1. Santa Clara, CA, USA: Intel Corporation, 1987.
[110] C.-J. Chen, W.-M. Cheng, R.-F. Tsai, H.-Y. Tsai, and T.-C. Wang, "A pipelined asynchronous 8051 soft-core implemented with balsa," in *Proc. IEEE Asia–Pacific Conf. Circuits Syst. (APCCAS)*, Nov./Dec. 2008, pp. 976–979.
[111] S. R. Naqvi and A. Steininger, "A tree arbiter cell for high speed resource sharing in asynchronous environments," in *Proc. Conf. Design, Autom. Test Eur.*, 2014, p. 295.
[112] C.-J. Chen *et al.*, "A quasi-delay-insensitive microprocessor core implementation for microcontrollers," *J. Inf. Sci. Eng.*, vol. 25, no. 2, pp. 543–557, 2009.
[113] T. Hung-Yue, W.-M. Cheng, Y.-T. Chang, C.-J. Chen, and F.-C. Cheng, "A self-timed dual-rail processor core implementation for microcontrollers," in *Proc. IEEE Int. Conf. Electron. Devices, Syst. Appl. (ICEDSA)*, Apr. 2011, pp. 39–44.
[114] K. A. Fawaz *et al.*, "A dynamically reconfigurable asynchronous processor for low power applications," in *Proc. IEEE 8th Symp. Appl. Specific Processors (SASP)*, Oct. 2010, pp. 93–96.
[115] K. A. Fawaz *et al.*, "A dynamically reconfigurable asynchronous processor for low power applications," in *Proc. IEEE Conf. Design Archit. Signal Image Process. (DASIP)*, Oct. 2010, pp. 76–83.
[116] S. Khawam, I. Nousias, M. Milward, Y. Yi, M. Muir, and T. Arslan, "The reconfigurable instruction cell array," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 16, no. 1, pp. 75–85, Jan. 2008.
[117] K. A. Fawaz, "A dynamically reconfigurable asynchronous processor," Ph.D. dissertation, School Eng., Univ. Edinburgh, Edinburgh, U.K., 2012. [Online]. Available: https://www.era.lib.ed.ac.uk/bitstream/handle/1842/9442/Fawaz2012.pdf
[118] J. Rose and S. Brown, "Flexibility of interconnection structures for field-programmable gate arrays," *IEEE J. Solid-State Circuits*, vol. 26, no. 3, pp. 277–282, Mar. 1991.
[119] K. Fawaz, T. Arslan, and I. Lindsay, "Conditional acknowledge synchronisation in asynchronous interconnect switch design," in *Proc. IEEE NASA/ESA Conf. Adapt. Hardw. Syst. (AHS)*, Jul./Aug. 2009, pp. 126–131.
[120] *ARM7 Thumb Family Datasheet*, document 0035-3/02.02, ARM, 2002.
[121] S. Agarwala *et al.*, "A 600-MHz VLIW DSP," *IEEE J. Solid-State Circuits*, vol. 37, no. 11, pp. 1532–1544, Nov. 2002.
[122] X. Li, B. Gunturk, and L. Zhang, "Image demosaicing: A systematic survey," *Proc. SPIE*, vol. 6822, p. 68221J, Jan. 2008.
[123] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, no. 90, pp. 297–301, 1965.
[124] D. W. Trainor, J. P. Heron, and R. F. Woods, "Implementation of the 2D DCT using a Xilinx XC6264 FPGA," in *Proc. IEEE Workshop Signal Process. Syst., SIPS Design Implement.*, Nov. 1997, pp. 541–550.
[125] G. Martinez, "TMS320VC5501/02 power consumption summary," Appl. Rep. TI SPRAA48, Texas Instrum. Incorporated, Dallas, TX, USA, 2004.
[126] T.-T. Liu and J. M. Rabaey, "A 0.25 V 460 nW asynchronous neural signal processor with inherent leakage suppression," *IEEE J. Solid-State Circuits*, vol. 48, no. 4, pp. 897–906, Apr. 2013.
[127] V. Karkare, S. Gibson, and D. Markovic, "A 130-μW, 64-channel neural spike-sorting DSP chip," *IEEE J. Solid-State Circuits*, vol. 46, no. 5, pp. 1214–1222, May 2011.
[128] R. Diamant, R. Ginosar, and C. Sotiriou, "Asynchronous sub-threshold ultra-low power processor," in *Proc. IEEE 25th Int. Workshop Power Timing Modeling, Optim. Simulation (PATMOS)*, Sep. 2015, pp. 89–96.
[129] J. Cortadella, A. Kondratyev, L. Lavagno, and C. P. Sotiriou, "Desynchronization: Synthesis of Asynchronous Circuits From Synchronous Specifications," *IEEE Trans. Comput.-Aided Design Integr.*, vol. 25, no. 10, pp. 1904–1921, Oct. 2006.
[130] P. A. Beerel, G. D. Dimou, and A. M. Lines, "Proteus: An ASIC flow for GHz asynchronous designs," *IEEE Design Test Comput.*, vol. 28, no. 5, pp. 36–51, Sep./Oct. 2011.
[131] D. Bhadra and K. S. Stevens, "Design of a low power, relative timing based asynchronous msp430 microprocessor," in *Proc. Conf. Design, Autom. Test Europe*, 2017, pp. 794–799.
[132] O. Girard. *OpenMSP430*. Accessed: Jan. 3, 2019. [Online]. Available: https://opencores.org/project/openmsp430
[133] A. Bardsley and D. A. Edwards, *Balsa: An Asynchronous Circuit Synthesis System*. Manchester, U.K.: Univ. Manchester, 1998.
[134] L. Necchi, L. Lavagno, D. Pandini, and L. Vanzago, "An ultra-low energy asynchronous processor for wireless sensor networks," in *Proc. 12th IEEE Int. Symp. Asynchronous Circuits Syst.*, Mar. 2006, p. 85.
[135] I. Blunno, J. Cortadella, A. Kondratyev, L. Lavagno, K. Lwin, and C. Sotiriou, "Handshake protocols for de-synchronization," in *Proc. IEEE 10th Int. Symp. Asynchronous Circuits Syst.*, Apr. 2004, pp. 149–158.
[136] Y. Liu, G. Xie, P. Chen, J. Chen, and Z. Li, "Designing an asynchronous fpga processor for low-power sensor networks," in *Proc. IEEE Int. Symp. Signals, Circuits Syst. (ISSCS)*, Jul. 2009, pp. 1–6.
[137] S. Ghosh, J. Tessier, and M. A. Bayoumi, "ASPEN: An asynchronous signal processor for energy efficient sensor nodes," in *Proc. 17th IEEE Int. Conf. Electron., Circuits, Syst. (ICECS)*, Dec. 2010, pp. 268–272.
[138] M. Laurence, "Introduction to Octasic asynchronous processor technology," in *Proc. IEEE 18th Int. Symp. Asynchronous Circuits Syst. (ASYNC)*, May 2012, pp. 113–117.
[139] M. Herrera and F. Viveros, "Asynchronous 8-bit processor mapped into an FPGA device," in *Proc. IEEE Colombian Conf. Commun. Comput. (COLCOM)*, Jun. 2014, pp. 1–7.
[140] S. Keller, A. J. Martin, and C. Moore, "DD1: A QDI, radiation-hard-by-design, near-threshold 18uW/MIPS microcontroller in 40 nm Bulk CMOS," in *Proc. 21st IEEE Int. Symp. Asynchronous Circuits Syst. (ASYNC)*, May 2015, pp. 37–44.
[141] A. Przybylski, K. Haque, and P. Beckett, "The Bel array: An asynchronous fine-grained co-processor for DSP," in *Proc. IEEE 10th Int. Conf. Signal Process. Commun. Syst. (ICSPCS)*, Dec. 2016, pp. 1–7.
[142] M. Fiorentino, Y. Savaria, C. Thibeault, and P. Gervais, "A practical design method for prototyping self-timed processors using FPGAs," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2016, pp. 1754–1757.
[143] M. Fiorentino, Y. Savaria, and C. Thibeault, "FPGA implementation of token-based self-timed processors: A case study," in *Proc. 15th IEEE Int. New Circuits Syst. Conf. (NEWCAS)*, Jun. 2017, pp. 313–316.

ZAHEER TABASSAM was born in Haripur, Pakistan, in 1994. He received the B.S. degree in electronics from The University of Haripur, Haripur, in 2016, and the M.S. degree in electrical engineering from the COMSATS University Islamabad at Wah, Pakistan, in 2019, where he has been a Research Associate with the Department of Electrical and Computer Engineering, since 2017. His current research interests include asynchronous processors and logic, neuromorphic computing, brain-inspired computing, and electronic systems.

SYED RAMEEZ NAQVI was born in Islamabad, Pakistan, in 1983. He received the B.Sc. degree in computer engineering from the COMSATS Institute of Information Technology, Islamabad, in 2005, and the M.Sc. degree in electronic engineering from The University of Sheffield, Sheffield, U.K., in 2007. He was awarded a fully funded scholarship for the Ph.D. degree by The PhD School of Informatics, Vienna University of Technology, Vienna, Austria, from 2009 to 2013, where he worked on fault-tolerant asynchronous logic with the Embedded Computing Systems Group, Institute of Computer Engineering. Since 2014, he has been an Assistant Professor with the Department of Electrical and Computer Engineering, COMSATS University Islamabad, Wah Cantonment, Pakistan, where he is teaching at both undergraduate and postgraduate levels and leading a research group on digital systems design and VLSI. He has published 30 research articles in various international conferences and journals.

TALLHA AKRAM received the B.S. degree in computer engineering from the COMSATS University Islamabad at Abbottabad, Pakistan, in 2006, the M.S. degree in embedded systems and control engineering from Leicester University, U.K., in 2008, and the Ph.D. degree in computer vision and pattern recognition from Chongqing University, China, in 2014. He is currently an Assistant Professor with the Electrical and Computer Engineering Department, COMSATS University Islamabad at Wah, Pakistan. He is the author of number of peer-reviewed journals and conferences. His research interests include computer vision, pattern recognition and machine learning, artificial intelligence, and applied optimization.

MUSAED ALHUSSEIN was born in Riyadh, Saudi Arabia. He received the B.S. degree in computer engineering from King Saud University, Riyadh, in 1988, and the M.S. and Ph.D. degrees in computer science and engineering from the University of South Florida, Tampa, FL, USA, in 1992 and 1997, respectively. Since 1997, he has been on the Faculty of the Computer Engineering Department, College of Computer and Information Science, King Saud University. He is currently the Founder and the Director of Embedded Computing and Signal Processing Research Laboratory. His research interests include the typical topics of computer architecture and signal processing and with an emphasis on VLSI testing and verification, embedded and pervasive computing, cyber-physical systems, mobile cloud computing, big data, eHealthcare, and body area networks.

KHURSHEED AURANGZEB received the B.S. degree in computer engineering from the COMSATS Institute of Information Technology at Abbottabad, Pakistan, in 2006, the M.S. degree in electrical engineering (System-on-Chip) from Linköping University, Sweden, in 2009, and the Ph.D. degree from Mid Sweden University, Sundsvall, Sweden, in 2013. From 2013 to 2016, he was an Assistant Professor/HoD with the Electrical Engineering Department, Abasyn university, Peshawar, Pakistan. He is currently an Assistant Professor with the College of Computer and Information Science, King Saud University, Riyadh, Saudi Arabia. His research interests include wireless visual sensor networks, design methods and implementation of embedded systems, applied image/signal processing, image compression, traffic monitoring, cloud computing, edge computing, the Internet of Things, smart grids, smart buildings, machine learning, and deep learning.

SAJJAD ALI HAIDER received the B.S. degree in computer engineering from the COMSATS University Islamabad (CUI) at Wah, Pakistan, in 2005, the M.S. degree in embedded systems and control engineering from Leicester University, U.K., in 2007, and the Ph.D. degree from Chongqing University, China, in 2014. Since 2005, he has been with the Department of Electrical Engineering, CUI Wah, where he is currently an Assistant Professor. His research interests include embedded systems, control systems, and machine learning.