



# Fast Asynchronous VLSI Circuit Design Techniques and their Application to Microprocessor Design

Shannon V. Morton, B. E. (Hons.)

Department of Electrical and Electronic Engineering  
The University of Adelaide  
Adelaide, South Australia.

January 22, 1997

## *Addenda*

### *Section 2.2 - Asynchronous hardware*

This thesis is focussed on practical asynchronous circuit design with an emphasis on micro-processors, and therefore only those processors which have been designed to fabrication have been included in the literature discussion. Fabricated test structures and coded microprocessors do not provide enough reliable data to justify their discussion in this context, although in Section 8.1 a description of coded superscalar processors, as compared to the author's, is given.

#### *Section 2.2.2 - AMULET I and II*

The AMULET I design was the first generation attempt at implementing a sixth generation commercial ARM6 processor, and was built with significantly less man-power and resources. This makes comparisons between them difficult, and the performance gap of 50% should be treated cautiously. The subsequent AMULET II processor has since demonstrated improved performance over the ARM6.

### *Section 3.5 - The ECS representation*

The ECS representation is intended to enable asynchronous circuits to be specified in a clear and concise format which models the interaction of data and control wires. It is not intended as a formal tool for synthesis, and as such has not been developed using formal methods. Instead, an intuitive description of the representation has been presented based on the practical implementation of asynchronous circuits, as this is the major focus of the thesis.

### *Section 4.1 - Algebraic improvements of a TS*

The simplifications described in this section are synonymous with those of boolean logic.

#### *Section 5.3.1 - Dynamic logic*

The nature of the data stream will also impact the power dissipation of a dynamic versus a static gate.

#### *Section 5.3.3.2 - Self-timed pseudo-nmos logic*

This circuit has a fast completion detection time compared to the static logic tree, as evidenced in Table 6.8. Compared to a typical dynamic gate however, it will be slow.

### *Chapter 6 - Self-timed Architectures*

The pseudo self-timed architectures presented in this chapter do not require additional safety margins. The computation and completion paths are closely matched in layout, and an implicit margin is already included in the handshaking overhead to compensate for any variations.

# Contents

Abstract	ix
Declaration	x
Preface	xi
Acknowledgements	xii
List of Figures	xiii
List of Tables	xvi
<b>1 Introduction</b>	<b>1</b>
1.1 Advantages of asynchronous systems	2
1.1.1 Global communication	2
1.1.2 Data dependent computation times	3
1.1.3 Resilience to operating conditions	3
1.1.4 Reduced power dissipation	4
1.1.5 Incremental improvements	4
1.1.6 Synthesis and verification	4
1.1.7 Power spectrum	5
1.2 Disadvantages of asynchronous systems	5
1.2.1 Control complexity	5
1.2.2 Testability	5
1.2.3 Area overhead	6
1.2.4 Operating speed	6
1.2.5 Integration and software support	7

1.3	Asynchronous paradigms . . . . .	7
1.3.1	Timing assumptions . . . . .	7
1.3.2	Control signalling . . . . .	8
1.3.3	Signal encoding . . . . .	9
1.3.4	Summary . . . . .	10
1.4	Thesis outline . . . . .	10
<b>2</b>	<b>Related Work</b>	<b>12</b>
2.1	Synthesis and verification . . . . .	12
2.1.1	Tangram . . . . .	13
2.1.2	Communicating processes . . . . .	13
2.1.3	Signal transition graphs (STGs) . . . . .	14
2.1.4	STG related synthesis . . . . .	15
2.1.5	Other contributions . . . . .	15
2.2	Asynchronous hardware . . . . .	15
2.2.1	Micropipelines and the CFPP . . . . .	16
2.2.2	AMULET I and II . . . . .	17
2.2.3	DCC error detector . . . . .	17
2.2.4	Caltech microprocessor . . . . .	18
2.2.5	Other contributions . . . . .	18
2.3	Summary of related works . . . . .	18
2.4	A need for speed . . . . .	19
<b>3</b>	<b>Event Controlled Systems (ECS) Design Representation</b>	<b>21</b>
3.1	Conventions . . . . .	21
3.2	Event controlled elements . . . . .	22
3.2.1	Muller-C element . . . . .	22
3.2.2	Merge gate . . . . .	23
3.2.3	Send gate . . . . .	24
3.2.4	Feed gate . . . . .	25
3.2.5	Restore gate . . . . .	26
3.2.6	Until gate . . . . .	27
3.2.7	Latching element . . . . .	28

3.2.8	Delays and logic functions . . . . .	29
3.2.9	Relative gate speeds . . . . .	29
3.2.10	Micropipeline module exceptions . . . . .	29
3.3	Some formalisms for gate representations . . . . .	30
3.4	Analysis of methodologies . . . . .	31
3.5	The ECS representation . . . . .	33
3.5.1	Transforming signals into the temporal domain . . . . .	34
3.5.2	ECS operators and temporal equations . . . . .	36
3.5.3	Some ECS gate examples . . . . .	40
3.5.4	Comparative gate representations . . . . .	43
3.5.5	Some example TS's and their corresponding circuits . . . . .	43
3.5.6	Precedence and properties of temporal operators . . . . .	45
3.5.7	Interconnectivity of gates in ECS . . . . .	46
3.5.8	Principles of error detection . . . . .	47
3.6	Summary . . . . .	50
<b>4</b>	<b>Fast Asynchronous Circuit Techniques</b>	<b>51</b>
4.1	Algebraic improvements of a TS . . . . .	51
4.1.1	Useless TE substitutions . . . . .	54
4.1.2	Useful TE substitutions . . . . .	55
4.1.3	Taking advantage of the <i>typical</i> scenario . . . . .	55
4.2	Improving acknowledge times . . . . .	56
4.2.1	Example: sharing of a common unit . . . . .	59
4.2.2	Example: data latching circuits . . . . .	60
4.2.3	Comments on improving acknowledgements . . . . .	63
4.3	Activating functional units . . . . .	64
4.3.1	Conditionally activated parallel units . . . . .	64
4.3.2	Generating a $\partial out$ event in the general sense . . . . .	66
4.3.3	Generating a $\partial out$ event for exclusively triggered units . . . . .	68
4.3.4	Splitting a tree of <i>select</i> gates into individual <i>feed</i> gates . . . . .	69
4.4	Reducing event path delays . . . . .	69
4.4.1	Moving metastability detection out of the event path . . . . .	69

4.5	Summary	73
<b>5</b>	<b>Asynchronous Pipelines</b>	<b>74</b>
5.1	FIFO pipelines	75
5.1.1	Micropipeline 2P FIFOs	75
5.1.2	4P FIFO circuits	76
5.1.3	A fast ECS FIFO	77
5.1.4	Comparison of FIFO designs	79
5.2	Pipelines with processing delays	80
5.3	Precharge pipelines: general concepts	82
5.3.1	Dynamic Logic	83
5.3.2	Requirements of a PP for dynamic logic	84
5.3.3	Methods of completion and precharge detection	86
5.4	Decoupled 4P precharge pipelines	90
5.4.1	Implementations for $PP\alpha$ , $PP\beta$ , and $PP\gamma$	90
5.4.2	Performance comparisons	91
5.5	ECS precharge pipelines	92
5.5.1	$PP\alpha$ implementation	92
5.5.2	$PP\beta$ implementation	93
5.5.3	$PP\gamma$ implementation	94
5.5.4	Performance comparisons	95
5.6	Comparison of ECS and D4P PP structures	96
5.7	Summary	97
<b>6</b>	<b>Self-Timed Architectures</b>	<b>98</b>
6.1	Strict self-timing requirements	99
6.2	Designing and utilizing self-timed units	101
6.3	Adder Structures	102
6.3.1	Self-timed ripple carry implementation	103
6.3.2	Self-timed ripple select implementation	105
6.3.3	Comparison of ST adders	106
6.3.4	Pseudo self-timing (PST)	107
6.3.5	PST ripple carry implementation	108

6.3.6	PST ripple select implementation . . . . .	109
6.3.7	Comparison of PST and ST adders . . . . .	110
6.4	Incrementer structures . . . . .	112
6.4.1	Self-timed incrementer . . . . .	113
6.4.2	Incrementer performance . . . . .	114
6.5	Comparator structures . . . . .	115
6.5.1	Possible implementations . . . . .	116
6.5.2	Comparator tree . . . . .	117
6.5.3	Comparator performance . . . . .	119
6.6	Multiplier structures . . . . .	120
6.6.1	Exploiting self-timed operation . . . . .	121
6.6.2	Simple partial product generation . . . . .	121
6.6.3	Radix 4 Booth encoding for generating partial products . . . . .	122
6.6.4	Recoding Booth's algorithm to improve performance . . . . .	124
6.6.5	Implementation, floorplanning, and area usage . . . . .	125
6.6.6	Performance and comparisons . . . . .	126
6.6.7	Potential improvements . . . . .	128
6.7	Summary . . . . .	130
<b>7</b>	<b><i>ECSTAC</i>: A Pipelined Microprocessor</b> . . . . .	<b>131</b>
7.1	Design considerations . . . . .	132
7.2	Instruction set architecture . . . . .	133
7.3	Architectural overview . . . . .	135
7.4	Processor sub-systems . . . . .	137
7.4.1	Instruction decode . . . . .	137
7.4.2	Operand fetch . . . . .	138
7.4.3	Adder, comparator, and stack processor . . . . .	142
7.4.4	Arithmetic and logical unit . . . . .	147
7.4.5	Order unit . . . . .	148
7.4.6	Registers and scoreboarding . . . . .	149
7.4.7	Program counter . . . . .	151
7.5	Testability issues . . . . .	154

7.5.1	Delay modelled <i>Vtt</i> bus . . . . .	154
7.5.2	Interface delays . . . . .	155
7.5.3	Scan testing . . . . .	155
7.6	Simulation results . . . . .	156
7.6.1	Sub-system simulations . . . . .	157
7.6.2	Core simulation environments . . . . .	158
7.6.3	General purpose instruction streams . . . . .	159
7.6.4	Instruction streams for determining bottlenecks . . . . .	161
7.6.5	Comparisons . . . . .	162
7.7	Summary . . . . .	164
<b>8</b>	<b><i>ECSCCESS</i>: A Superscalar Microprocessor</b>	<b>166</b>
8.1	Other asynchronous superscalar microprocessors . . . . .	167
8.1.1	SCALP . . . . .	167
8.1.2	Fred . . . . .	168
8.1.3	Rotary pipeline processor . . . . .	169
8.2	Characteristics of <i>ECSCCESS</i> . . . . .	170
8.3	Instruction set architecture . . . . .	171
8.4	General architecture . . . . .	173
8.5	Implementation of the <i>shore</i> . . . . .	174
8.5.1	Controlling RAW hazards . . . . .	175
8.5.2	Controlling WAR hazards . . . . .	176
8.5.3	Structure of the pre-FU unit . . . . .	177
8.5.4	Generating the return event to the sun . . . . .	178
8.5.5	Switching network . . . . .	179
8.6	Implementation of the <i>sun</i> and <i>moons</i> . . . . .	179
8.6.1	Globe controller . . . . .	180
8.6.2	PC controller . . . . .	181
8.6.3	Branch moon controller . . . . .	182
8.6.4	Stack moon controller . . . . .	182
8.7	Implementation of functional units . . . . .	183
8.7.1	AID unit . . . . .	183



8.7.2	MEM unit . . . . .	184
8.7.3	CMP unit . . . . .	184
8.8	Floorplanning issues . . . . .	185
8.8.1	Size of the ocean . . . . .	185
8.8.2	Size of the switching network . . . . .	185
8.8.3	A floorplan based on the minimum FU width . . . . .	186
8.8.4	Floorplanning for a larger FU width . . . . .	187
8.9	Simulation results . . . . .	188
8.10	Comparisons . . . . .	190
8.11	Extensions and improvements . . . . .	191
8.11.1	Incorporating interrupts . . . . .	191
8.11.2	Exception handling . . . . .	193
8.11.3	Reducing the ocean width for WAR and RAW hazards . . . . .	194
8.12	Summary . . . . .	195
<b>9</b>	<b>Conclusions</b>	<b>196</b>
9.1	Further work . . . . .	199
<b>A</b>	<b>Fundamental Temporal Equations and Corresponding ECS Gates</b>	<b>201</b>
<b>B</b>	<b>ISA of the <i>ECSTAC</i> Microprocessor</b>	<b>203</b>
B.1	Memory instructions . . . . .	203
B.1.1	Two byte instructions . . . . .	203
B.1.2	Four byte instruction . . . . .	204
B.1.3	The unused mode . . . . .	204
B.2	ALU instructions . . . . .	204
B.2.1	Two byte instruction (short mode) . . . . .	204
B.2.2	Three byte instructions (long mode) . . . . .	205
B.3	Branch instructions . . . . .	205
B.3.1	One byte instruction - CALL . . . . .	205
B.3.2	Two byte instructions - BRANCH . . . . .	206
B.4	Stack instructions . . . . .	206
B.5	Special instructions . . . . .	207

<b>C</b>	<b>ISA of the <i>ECSCCESS</i> microprocessor</b>	<b>208</b>
C.1	Branch instructions . . . . .	208
C.2	Interrupt instructions . . . . .	209
C.3	MOVE instruction . . . . .	210
C.4	LDC instruction . . . . .	210
C.5	FU instructions . . . . .	211
C.5.1	Register unit . . . . .	211
C.5.2	Arithmetic unit . . . . .	212
C.5.3	Multiply, divide, and sqrt unit . . . . .	212
C.5.4	Shifter and logical unit . . . . .	213
C.5.5	Comparator unit . . . . .	213
C.5.6	Memory unit . . . . .	214
C.5.7	Floating point units and co-processors . . . . .	215
	<b>Bibliography</b>	<b>216</b>

# Abstract

Over the past decade a variety of asynchronous synthesis techniques have been proposed. The majority of these have been concerned with generating provably correct circuits with high reliability, whereas others have focussed on producing circuits with low power dissipation. However in taking such approaches the resulting circuits are usually swamped with a large number of gates in the critical paths and are consequently inefficient in terms of speed.

This thesis describes a collection of novel design techniques engineered for high speed operation (such as fast pipeline control circuits and pseudo self-timed computations). In addition, a new gate representation is proposed to better reflect their functionality in an asynchronous domain. As an illustration of these design techniques two microprocessors have been implemented:

- *ECSTAC* is styled as a linear pipeline with a load/store architecture and an 8 bit data path and a 24 bit address path. It employs fast pipeline control circuits and utilizes some interesting asynchronous techniques for bypassing stages, controlling data hazards, and register fetching. *ECSTAC* has been fabricated using ES2's 0.7 $\mu$ m DLM CMOS process and demonstrated a peak operating speed of 28 Mips.
- *ECSCCESS* is structured to take advantage of self-timed data dependent computations and to employ functional parallelism. It has a 32 bit data path and can provide for up to 32 single precision (16 double precision) functional units which interact directly with each other, thus enabling out-of-order execution and global results forwarding. Their operation is fully decoupled from branches and interrupts to minimize stalling. Emphasis has been placed on maintaining a high throughput to the functional units. It employs novel design techniques for rapid data hazard detection between units, PC updating, and decoupled branch evaluation and branch target determination. *ECSCCESS* has been simulated in VHDL with delays comparable to those of the 0.7 $\mu$ m standard cell library used in *ECSTAC*, and demonstrated a peak operating speed of 181 Mips.