A Practical Comparison of Asynchronous Design Styles

D.W. Lloyd[1] & J. D. Garside[2]

[1]Micro Cores Development Division, STMicroelectronics, Bristol, U.K,

[2]Department of Computer Science, The University of Manchester, U.K.

dave.lloyd@st.com, jgarside@cs.man.ac.uk

Abstract

It is well known that single-rail, bundled-delay circuits provide good area efficiency but it can be difficult to match them with appropriate delay models. Conversely delay insensitive circuits such as those employing dual-rail codes are larger but it is easier to ensure timing correctness. In terms of speed, bundled-delay circuits need conservative timing but dual-rail circuits can require an appreciable completion detection overhead.

This paper compares designs in both of these styles and also a delay-insensitive 1-of-4 coded circuit using the practical example of an ARM Thumb instruction decoder. The results show that, through the application of careful optimizations, the 1-of-4 circuits out-performed single-rail circuits and reduced the power compared to dual-rail circuits.

1: Introduction

A robust VLSI design not only matches circuit design style to the function to be implemented, but also exploits the chosen fabrication technology. Single-rail datapaths are area-efficient in representing data in an asynchronous system requiring, only a single wire per bit. Their correct operation, however, requires a 'data-valid' control signal to indicate that a functional unit has completed its operation.

The required 'data-valid' signal may be obtained from the input signal(s) either by replicating the critical path of the functional unit or employing a discrete delay circuit which must embrace the worst-case delay of the slowest data signal through the unit and defines its operational speed.

Datapath elements, such as register files, are easily modelled when produced in custom VLSI layout; both switching and wiring delays can be duplicated. Other units, such as ALUs etc. are more problematic and the difficulties are exacerbated by the differences in wiring delay introduced by compiling the circuit.

Extensive validation is required to ensure that the timing relationships are maintained in all circumstances. As technology shrinks, parasitic elements make accurate delay matching increasingly difficult and time-consuming. This problem increases with the size and complexity of asynchronous systems. There has been a consequent escalation in the size of safety margins added to the matched delays and a growing dependence on the accuracy of sophisticated and expensive design tools.

Asynchronous design is particularly sensitive to errors in delay matching; slowing down a global clock to achieve timing closure is not an option here. Design approaches which remove the need to match delays thus present an attractive alternative. One such approach is to encode the data using a delay insensitive (DI) code which enables a functional unit to detect the completion of each operation using circuitry which generates a 'data-valid' signalling the result (a new codeword).

The use of a DI code has overheads however, with the choice of code affecting the performance, cost and power consumption. This paper compares the original single-rail design of the AMULET3 Thumb decoder with designs employing two DI encodings: a dual-rail code and a 1-of-4 code.

The Thumb decoder was chosen as it has unusual delay characteristics (detailed in section 6.1) which created significant matching problems for the single-rail design used in AMULET3. The lack of static timing analysis tools in the present AMULET design flow meant that the critical path had to be deduced from extensive simulation directed by manual analysis. The extra confidence and reduction in the required timing validation provided by completion detection would be very welcome. These benefits should be obtained with the smallest overhead. To achieve this dynamic logic and an engineering approach are employed.

1522-8681/01 $10.00 © 2001 IEEE

2: Thumb Instruction Capability

'Thumb' is an extension to the ARM architecture and may be viewed as a compressed form of a subset of the ARM instruction set. It comprises twenty one instruction formats drawn from the standard 32-bit ARM instruction set recoded into 16-bit op-codes to increase the code density [1].

AMULET3 is the first asynchronous ARM to include Thumb capability; it use a separate Thumb decompression stage prior to the main ARM decode. If a packet received from memory contains an ARM instruction then the Thumb stage collapses, forwarding the instruction through to the next stage with minimal delay. If the packet contains Thumb instructions the stage expands to allow sufficient time to perform the Thumb decompression. The decompressed instructions are then passed to the following stage where they are decoded as ARM instructions. A similar approach is used in the ARM7TDMI, but the clocked pipeline used on this processor prevents the time slot collapsing when running ARM code.

AMULET3 can fetch two Thumb instructions on each memory access which are passed as a packet to the Thumb stage; this perform two cycles, decompressing each in turn. The elastic nature of the asynchronous pipeline accommodates this 'one-in, two-out' behaviour automatically. The clocked pipelines used by the ARM7TDMI and ARM9TDMI processors cannot readily allow this behaviour and so, to simplify pipeline control, a single Thumb instruction only is fetched per memory access. This doubles the required number of memory cycles.

3: Thumb Decompression Logic

The decompressor performs a direct translation from 16-bit Thumb instructions to 32-bit ARM instructions. The AMULET3 Thumb decompressor implements THUMBv1 defined by ARMv4t instruction set architecture [1].

A one-to-one mapping between Thumb instructions and their ARM equivalents simplifies the decompression logic. Decompression is achieved by filling each field in the equivalent ARM instruction from the corresponding field in the Thumb instruction. This process is illustrated in figure 1. The example converts the Thumb ADD of a constant to a register (ADD rd, #Constant) into its ARM equivalent (ADD rd, rd, #Constant). A simple lookup table produces major and minor opcodes and the immediate value and register specifiers are zero-extended to fill the larger ARM instruction fields.

This means that the Thumb decoder – at least in AMULET3 – comprises a relatively small amount of random logic and a significant number of wide multiplexers. This is important as multiplexers are particularly suited to a dynamic CMOS implementation.

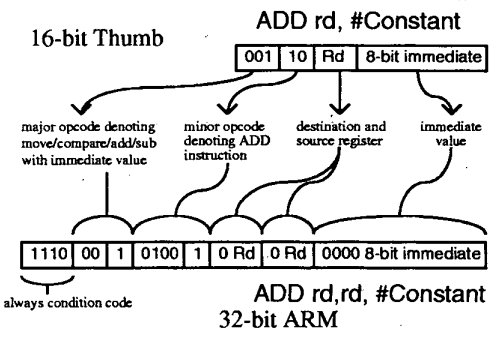

Figure 1: Thumb decompression

4: Representation of Data

The AMULET processors have, to-date, been constructed using single-rail datapaths, the designers being influenced by the low cost and the ease of design they offer.

In a single-rail design a 'data-valid' control signal is bundled with the result wires to indicate that a functional unit has completed a processing task. Additional circuity is provided to delay its passage such that it becomes valid only after the result is valid. Single-rail datapaths can only support data-dependent operation if a complex iterative function allows early termination. Operation dependence can be achieved by providing different task-dependent matched delays. The Thumb decode stage exploits the latter of these approaches to increase its performance (see section 6.1).

A DI code implementation also uses additional circuitry to produce a 'data-valid' signal. However this circuitry is employed to monitor the output of the unit to detect the production of new results, rather than trying to match the delay through the unit. This allows the DI code implementation to exploit whatever data and operational dependence the unit displays. Unfortunately, unlike delay matching, completion detection can take place only after the result has been produced. This clearly adds an extra overhead to the cycle-time of the functional unit. Circuit techniques to reduce this overhead are discussed in section 5.

Completion detection is the process of identifying certain codewords from the patterns of ones and zeros that appear on the wires. Faster detection can be achieved if all wires are returned to the zero state – called the spacer – between the transmission of each code word; this is called return-to-zero (RTZ) signalling.

Completion detection can be further simplified by restricting the number of logic ones that can appear on the wires for each codeword. 1-of-N or 1-Hot codes [2] represent data such that only a single wire of a set of N can be raised to logic one for each codeword. Thus completion detection is now reduced to a simple logical-OR of the wires.

Dual-rail codes are the most widely used form of 1-of-N

codes for constructing asynchronous systems [2]. The dual-rail code is a 1-of-2 code, employing two wires to encode each bit; a logic one is represented by raising one of the two wires high, a logic zero by raising the other wire high. The spacer is represented by both wires being held low.

Part of the popularity of the dual-rail code can be attributed to their relatively low cost representation of a large number of data wires. The cost of the 1-of-N codes as N –and hence the required number of wires – increases soon becomes excessive. It is far more efficient to represent a large number of bits as a set of concatentated dual-rail codes. Whilst this requires an OR-AND function to detect completion of the evaluation phase across the multiple pairs of wires it still only requires an OR function to detect the reset phase.

Recently [3] interest has focused on 1-of-4 codes. As can be seen from table 1, a 1-of-4 code encodes two bits onto four wires [2] using an approach similar to dual-rail encoding. This proves to be a very efficient encoding.

bits	code
spacer	0000
00	0001
01	0010
10	0100
11	1000

Table 1: 1-of-4 encoding

Table 2 illustrates the reasons for the interest in 1-of-4 codes. It presents a comparison of the 1-of-N codes and the single rail representation considered in terms of two metrics:

- Area efficiency: the number of wires required to encode each bit.
- Energy efficiency: the number of wire transitions required to send each bit.

This table does not consider the relative implementation costs, the costs of indicating data validity or the overheads required for handshaking. These are presented in later sections. For completeness the table does include the area and energy efficiency for dual-rail and 1-of-4 non-return-to-zero (NRTZ) codes. These encodings have the advantage of not requiring the insertion of the spacer between consecutive codewords; each new codeword is indicated by a transition on one of the wires.

It should also be noted that whereas the table presents the average case energy efficiency for the single-rail approach, it presents the obligatory energy for the DI codes. If two consecutive bits applied to the circuit are at the same logic level, the single-rail approach can convey this information with no changes on the data wires. In contrast, the DI codes *require* a change on the data wires to indicate that these are two separate data.

	Area wires/bit	**Energy** transitions/bit
single-rail	1	1/2 (average)
dual-rail (RTZ)	2	2
dual-rail (NRTZ)	2	1
1-of-4 (RTZ)	2	1
1-of-4 (NRTZ)	2	1/2

Table 2: Comparison of encodings

Table 2 suggests that for word lengths that are multiples of 2-bits, the RTZ 1-of-4 code has the same area cost per bit as the RTZ dual-rail code but provides twice the energy efficiency. Results presented in section 7.2 demonstrate that further efficiency increases are possible. Ostensibly using a NRTZ code allows the power to be reduced even further; however the use of signal transitions rather than levels makes completion detection far more costly.

5: Dynamic versus Static Circuits Styles

There are area and delay overheads resulting from representing the datapath using a dual-rail or 1-of-4 code. These arise from:

- the additional wiring required to support the 1-of-N encoding.
- the extra circuitry required for completion detection.

This additional logic contributes to the delay overhead. However, the most significant contribution arises from the need to extend the cycle time to accommodate the RTZ phase. The incorporation of this phase can result in a doubling of the cycle time.

The designer can make the most significant reductions in these costs by changing from a static to a dynamic circuit style. Unfortunately these reductions are usually traded for an increase in design effort or a reduction in the robustness of the design.

Both static and dynamic CMOS circuits have been used to construct single-rail datapaths for the AMULET processors. Significantly, dynamic logic is employed wherever a replication of a critical path is required for timing purposes. For example, it is used in the register bank and for arithmetic operations. This is a consequence of the monotonic operation of dynamic logic which make it easier to control and to characterise. Conversely, if a functional unit is constructed using

static logic, separate delay matching circuits are used to provide timing information.

Construction of static CMOS circuits to support a 1-of-N code is straightforward. The evaluation function is realised in the n-stack whilst the reset (RTZ) function is constructed in the p-stack. The arrival of a spacer dictates to the circuit that it should enter the reset phase of operation. This is detected using transistors in the p-stack. As the number of inputs to a circuit increases the required number of p-transistors also increases with the result that the cost of the circuit rapidly becomes untenable.

Dynamic logic aims to reduce area and speed costs by eliminating the need to implement both the evaluation and the reset functions using transistor stacks. The expensive p-stack employed for the reset function is replaced with a single p-transistor. This p-transistor is used to precharge the summand node (figure 2).

Removal of the p-stack reduces the area cost and decreases the circuit delay. These reductions are obtained at the price of several operational and constructional constraints. For a single-rail implementation these constraints would present a cost overhead but, for a 1-of-N code operating using an RTZ protocol, they closely match the required operational implementation characteristics:

- dynamic logic must precharge between each evaluation. This matches the reset phase required by the RTZ operation of the DI codes. Further, because the reset phase is initiated by a precharge signal ('nprech' in figure 2) rather than the passage of the spacer it permits multiple circuit stages to precharge concurrently. Precharge also provides opportunities to simplify completion detection allowing the circuitry employed to detect completion of precharge to be replaced by a matched delay [5].
- dynamic logic is an incomplete logic as it can only implement non-inverting functions. Thus if a single-rail implementation requires an inverted version of the signal a separate path must be built for it. This is not an additional overhead for 1-of-N codes as the required multiple paths are already present.

Domino circuits use an inverting static gate on their outputs to ensure that, after precharge, the gate output – which forms the input to next gate – is at logic zero [7]. The domino gate shown in figure 2 uses a static inverter.

Domino logic does not offer the same robustness as static logic: the summand node in a domino logic is only driven whilst the circuit is precharging or evaluating. When undriven it is vulnerable to loss of state as a consequence of leakage current, poor noise immunity and charge sharing. These potential failure mechanisms can be controlled either by careful control of the operation of each domino gate or by adding extra failure prevention circuitry to each gate. It is possible to optimise a design by trading between more control or extra circuitry. This is explored in section 6.1.

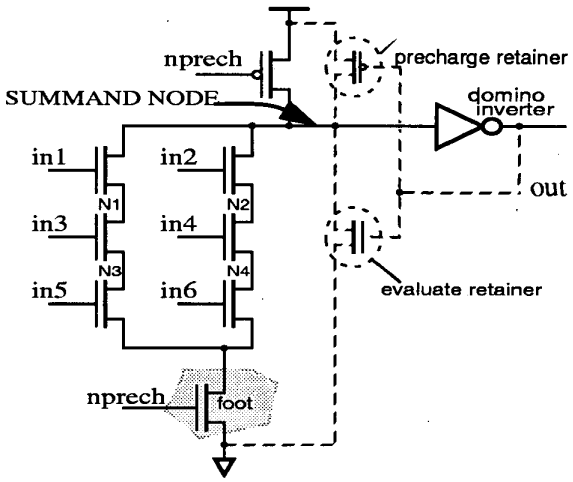

Figure 2: Generalised domino gate

Figure 2 illustrates the use of precharge and evaluate keepers. These replace charge lost through leakage. Use of keepers adds a small area overhead due to the extra transistors and a small increase in delay caused by the half-latch hysteresis. Keepers are not required if the period of time for which the gate is undriven is guaranteed to be sufficiently small.

The keepers can also perform the secondary task of improving noise immunity. Domino gates can fail if noise is capacitively coupled into the domino gate during the evaluation phase of its cycle causing false output states. The use of DI codes provides a reduction in capacitive coupling; by interdigitating the wires required for a dual-rail implementation the coupling can be reduced as only (at most) one of the adjacent wires can switch in any evaluation. This effect is further enhanced using a 1-of-4 encoding.

The floating nodes present in domino logic make it particularly sensitive to charge sharing. This occurs when transistors within the domino gate are connected to form a capacitive charge divider resulting in a reduction in the precharge voltage which may cause the output to switch falsely. Charge sharing can be decreased by reducing the number of discharged transistors that can be connected to a precharged node. By arranging the n-stacks to perform a multiplexor operation we can ensure that, for each evaluation phase, only a single stack is connected to the precharged node. This approach – which is employed whenever possible in this work – significantly reduces charge sharing problems. For example, if the circuit in figure 2 is operated as a multiplexor, inputs 'in1' and 'in2' are used to select which n-stack will be connected to the precharged summand node, thereby limiting the number of discharged nodes that can be connected to form a capacitive divider.

Figure 2 also illustrates the use of an extra transistor (the 'foot') which, during precharge, prevents a potential short-circuit through the n-stack to ground.

6: Thumb Decoder Implementations

It is assumed for each implementation that the Thumb decoder is embedded in a processor constructed using the same design style. For example, we assume that the dual-rail implementation receives data from upstream dual-rail stages and outputs its results to downstream dual-rail stages. This removes the need to consider the costs of decoding/encoding data as it passes through the Thumb decoder stage and allows the evaluation of different methods for controlling the inter-stage operation.

6.1: Control Path

The ARM architecture specifies its instruction set by a status bit called the 'T-bit'. In AMULET3 this bit is included within the instruction packet. The Thumb decode stage uses the T-bit to determine whether an instruction packet contains an ARM instruction or Thumb instructions.

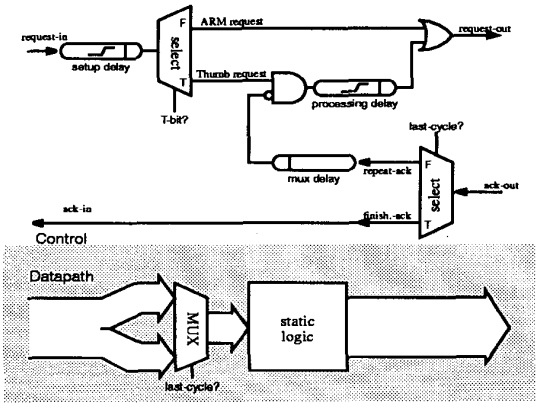

Figure 3: Single-rail control

Single-rail Control

Figure 3 illustrates the control for a single-rail static logic implementation of the Thumb decoder. This design was used for the AMULET3 processor.

The decoder performs a different sequence of operations depending on which instruction set is active. A *select* is provided to steer the incoming request, 'request-in', straight to the downstream stage for ARM instructions or via the AND-gate and matched delay for Thumb instructions. This matched delay provides timing for a first Thumb decompression. The downstream stage will respond with 'ack-out' when it has latched the results. If a second decompression is required 'ack-out' can be steered through a further sequence of matched delays. The consequent 'ack-out' will then directly provide the acknowledge for the upstream stage. A rising edge on 'repeat-ack' is used to change to 'last-cycle' thereby switching the *select* and the datapath multiplexor.

Matched delays were used to provide timing as the datapath was constructed using static logic. They also provide

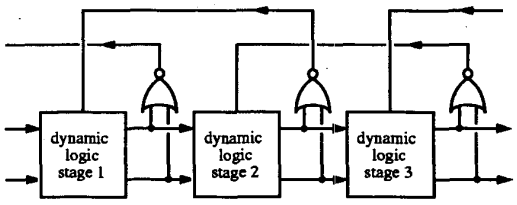

Figure 4: Synchronising dynamic logic

setup times for data against the rising edge of a request signal. As significant hold times were not required, asymmetric delay matching circuits could be employed in most cases; the full delay is provided for a rising edge and a minimal delay for a falling edge.

Timing for the first decompression is provided by the matched delay 'processing delay' in conjunction with 'setup delay', the T-bit *select* and the two logic gates. The matched delay 'mux delay' provides time for the datapath multiplexor to switch. This allows the second Thumb instruction to flow into the decompression logic where decompressing starts. Simultaneously, the RTZ actions on the 'request-out', 'ack-out' handshake pair will be initiated.

Timing the second decompression is more complex as part of the decompression time is concurrent with the RTZ actions on the control path. Additionally, the matched delay 'processing delay' provides timing twice; firstly for the falling edge and then again for the rising edge of 'request-out'. A further complication is that the static logic – whose timing is being matched – is a complex structure with widely variable logic depths. In these circumstances maintaining the correct timing relationships between the control path and the datapath is a demanding task and considerable margins must be employed for safety.

DI Control

A DI implementation simplifies the design of the control path as completion detection replaces delay matching and setup times are not required as the logic will automatically wait until the required data arrives.

To enable this wait the control path uses static logic (only the datapath is dynamic) and so the length of time the control path can wait is unrestricted. In contrast, for the dynamic datapath, leakage currents (severely) restrict this waiting time. If the waiting time needs to be increased charge-replacement keepers must be employed. The maximum time a dynamic circuit must wait is determined by its evaluate and precharge strategy. Figure 4 illustrates a typical approach. This is based on the structure first proposed by Williams [4] to control multiple stages of dynamic DI logic.

Each functional block has a completion detector which provides control for the preceeding stage (figure 4 demonstrates the technique for an RTZ dual-rail implementation and so a NOR-gate provides completion detection). Crucially the operation of this and other schemes [8] requires that, once a

particular stage has precharged, its preceeding stage is forced to return to the *undriven state* where it waits for the arrival of data. The length of time that a stage is undriven is determined by the throughput of data. Whilst this strategy works well for systems where the throughput can be anticipated, for a number of reasons it is inappropriate as a control strategy for the AMULET3 pipeline.

Firstly, the AMULET3 pipeline can stall due to a large variety of causes with the consequence that an individual stage may be forced to spend long periods of time neither driven by evaluate nor precharge. Thus a direct application of this approach would require both precharge and evaluate keepers on all domino gates to prevent a loss of state.

More seriously, the decompression logic may be undriven – waiting for data – for considerable periods of time; for example a program coded in ARM instructions will never use it. During this time the dynamic nodes within each domino gate are not refreshed, precharge is sustained merely by the charge replacement supplied by the very-small weak-feedback transistors present in its precharge keepers. As the reliability of this approach is open to question this work takes an alternative approach keeping the dynamic logic in precharge – driven by large precharge transistors – until it is required. Unfortunately, compared to Williams' scheme, this will increase the latency of each stage as delays through the control path result in the data arriving slightly before the dynamic logic is taken out of precharge.

Request activated DI control

Figure 5 illustrates a control strategy devised to coordinate the activities for a DI implementation of the Thumb decode stage. The structure and operation is very similar to that employed for single-rail control (figure 3). The most significant difference lies in the removal of circuitry to provide the setup and timing delays.

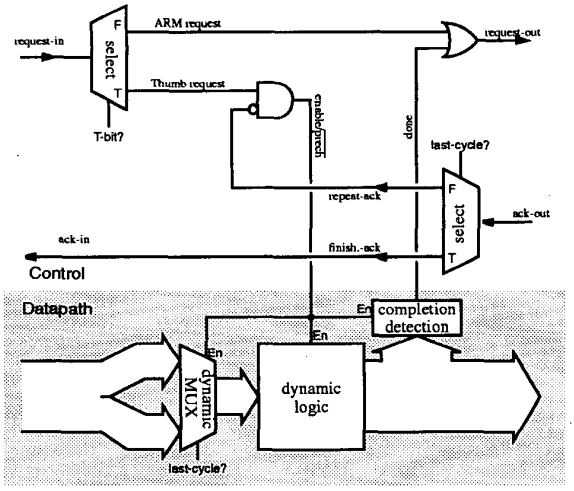

Figure 5: Request activated control

The datapath (multiplexor, processing logic and completion detector) is implemented using domino logic. It is held in precharge until it is enabled ('enable/$\overline{\text{prech}}$') by an explicit request ('request-in') from the preceeding stage –the datapath is thus *request activated*. The 'done' signal, produced by the completion detection circuitry, is used to form the request to the next stage. This is different from Williams' design where the done signal is fed back to control the preceeding stage and the request to the downstream stage is implicit in the arrival of the encoded data.

A disadvantage of this request activated scheme is that it requires an extra signal 'request-out' to be passed with the result to the following stage. The advantage of this scheme is that the provision of this enable signal ('done') allows latching stages to be implemented using low cost transparent latches and standard latch controllers (for safety completion detection could be provided on the output of these latches) instead of the expensive C-element based latching stages otherwise required. This minimises the changes required to convert the AMULET3 pipeline from a single-rail to a DI implementation.

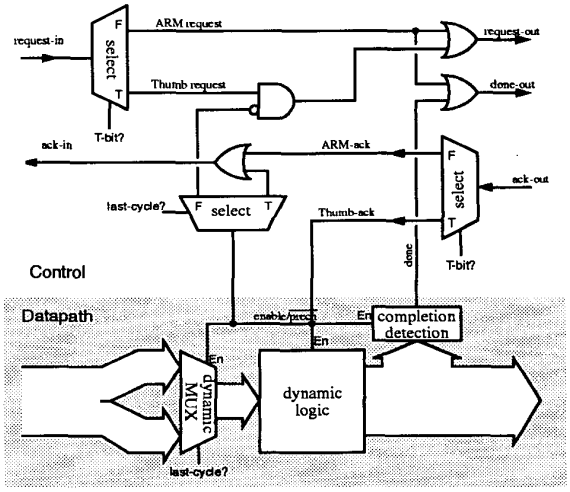

Figure 6: Acknowledge activated control

Acknowledge activated DI control

The request activated strategy has the disadvantage that the decode stage is allowed to stall waiting for the downstream latches to become available. Safe operation therefore requires evaluate keepers to maintain the *data* state. The charge replacement provided by keepers is sufficient during these pipeline stalls as the stall only lasts for short periods of time, typically a fraction of a stage cycle time.

Previous work has shown that the precharge and evaluate keepers can be removed if it is guaranteed that downstream latches are available before the domino logic evaluates [6]. This is achieved by generating the enable signal ('enable/$\overline{\text{prech}}$') from the downstream acknowledge ('ack-out'), rather

than from the upstream request ('request-in'); the acknowledge will be raised when the downstream latch is available and will only be lowered, returning the dynamic logic to precharge, after the result has been latched. This control structure is illustrated in figure 6.

In this scheme the request is not available to be used to enable the latches so a separate signal 'done-out', generated by the completion detection circuitry, is used instead. 'request-out' and 'done-out' are combined within the latch controller (not shown) to generate 'ack-out'[6]. To allow the same downstream latch-controller to be used for ARM as well as Thumb instructions 'done-out' is supplied for both.

The removal of keepers requires an increase in the complexity of the control and the latency of the stage. However there is however a reduction in costs for the datapath of two transistors per gate output. Whether this reduction in costs justifies this increase in control will be evaluated later.

6.2: Data Path

The data path is dominated by multiplexers which are used to select between ARM and Thumb instructions. They also perform the decompression by selecting information from the appropriate Thumb instruction fields and steering it to the corresponding ARM instruction fields.

Figure 7 illustrates the structure of the datapath for the Thumb decode. The following stage (ARM decode) receives either an ARM instruction and an immediate value or a decompressed Thumb instruction and an immediate value.

The following discussion will focus on the Thumb immediate processor unit as the task it performs is easy to understand yet the obstacles to its DI implementation embody those for the whole unit. The task of the immediate processor is to produce either literal values or the offsets to be used in address calculations. This task is performed very frequently as immediate values are present in fourteen of the twenty-one Thumb instruction formats.

To obtain an immediate value a partial decoding of the instruction is required to find the length and position of the immediate field (if any) for that particular format. Immediate fields can be 3 bits, 5 bits, 7 bits, 8 bits or 11 bits in length and the position of the field within the 16 bit instruction varies according to the particular format.

Immediate values are use to form literals or address offsets. If the immediate value is to be used as a literal after decoding it can be passed straight onto the following pipeline stage. If the value is to be used as an address offset it must first be scaled according to the size of the data transfer. If the offset is to be used by a branch operation it must also be sign-extended.

The operations required of the immediate processor can be categorised according to the range of sizes of immediate values used. Three categories were distinguished: small immediates (3-bit and 5-bit), big immediates (7-bit and 8-bit) and

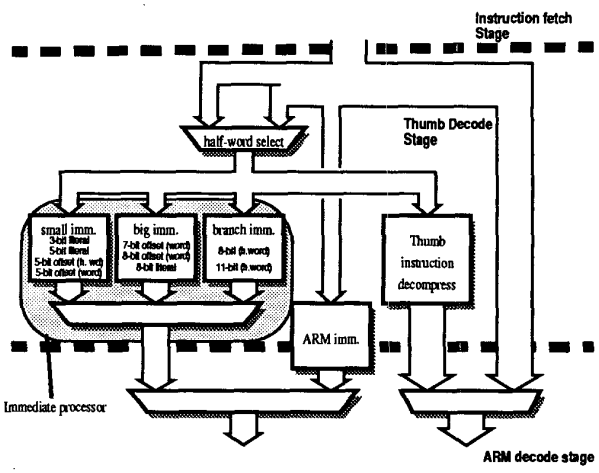

Figure 7: Thumb decode datapath

branch immediates (8-bit and 11-bit).

Within each category different operations (scaling, sign-extending, ...) are performed according to the particular instruction. Use of these categories simplifies (and so speeds up) decoding of the instructions.

To achieve high speed operation each category speculatively produces results for all Thumb instructions. This is illustrated in figure 7 where results are produced by 'small imm.', 'big imm.' and 'branch imm.' and a multiplexor is used to select the desired result. Speculation allows decoding and processing to be undertaken using a staged-refinement approach. For each category the decoding and processing

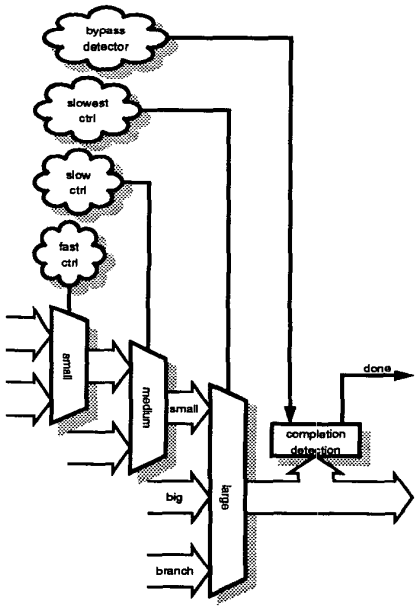

Figure 8: Stage control

task is divided into a number of stages, where each stage performs a multiplexor-type operation. The stages are arranged such that the control information for the first stage requires very little decoding of the instruction and so allows this control information to be generated very quickly. Subsequent stages are ordered so that generation of the control for later stages requires an increasing amount of instruction decoding and hence an increasing amount of time to perform (figure 8). The time overhead for these later, slower stages is hidden as these operations are performed concurrently with the processing of the earlier stages. This is essential as the operation of the later multiplexor controllers is complex requiring their dynamic logic to be split into multiple, independently precharged sections to avoid charge sharing.

The single-rail design was taken from the AMULET3 processor and so the target cycle time for the dual-rail and 1-of-4 designs were based on its performance. Rapid cycle-time was the primary design criterion.

The characteristics of the implementation technology (VLSI Technology 0.35μm three-layer metal) limit the number of series transistors in the n-stack (and hence the circuit inputs) to four. Operations which require more than this number of inputs have to be realised as multiple stage logic with resultant increases in cost and delay.

Single-rail and dual-rail implementations of an operator require a similar number of series transistors in the n-stack. Significantly, the 1-of-4 encoding generally allows an operator to be implemented using fewer series transistors and thus multiple input operators in a smaller number of stages. These savings occur because the 1-of-4 encoding requires fewer symbols to encode the binary inputs of a multiple input gate. For example, a two input 1-of-4 circuit can process four binary digits, whereas a two input dual-rail circuit can only process two binary digits. A 1-of-4 implementation typically results in a circuit with a wide NOR-structure. This limits the size of operation that can be realised in a single stage of logic because of problems with charge-sharing and reduced drive.

Cycle-time constraints and similarities in the depth of logic required to implement operators meant that for the single-rail and dual-rail realisations only very similar design optimizations were available. However for the 1-of-4 realisation the different circuit costs allowed different design optimizations.

To achieve the required performance the design requires each of the three categories of immediate to be evaluated speculatively and, if required, one of these results selected. A substantial amount of energy could be saved if the decoding and processing tasks were invoked only as and when necessary. This could be achieved if the first stage of control is able to identify for each category whether or not an instruction format belongs to that category. If the format does not belong, then the control will not enable the corresponding first stage multiplexor preventing data from passing through to enable the subsequent stages. To identify formats requires partial instruction decoding; unfortunately Thumb instructions have a dense and non-uniform encoding and so it can take six or more bits to identify a format. For both the single-rail and dual-rail representations this would require six or more symbols, a corresponding number of transistors in series (in most cases this number can be reduced by grouping formats, but not in every case) and so a multi-stage logic realisation. This arrangement is too slow to meet the maximum cycle time restrictions.

The 1-of-4 encoding can represent 6 bits using only three symbols (2 bits/symbol), only requiring three transistors in series (a single stage of logic) and so the decoding operation can be performed within the available time.

In this scheme formats which do not require immediate processing do not produce results and so consequently completion detection will not occur. To overcome this an extra block of control, called the 'bypass detector' (figure 8) is required to detect all formats that do not require immediate processing. If one of these is detected the 'bypass detector' triggers completion detection directly.

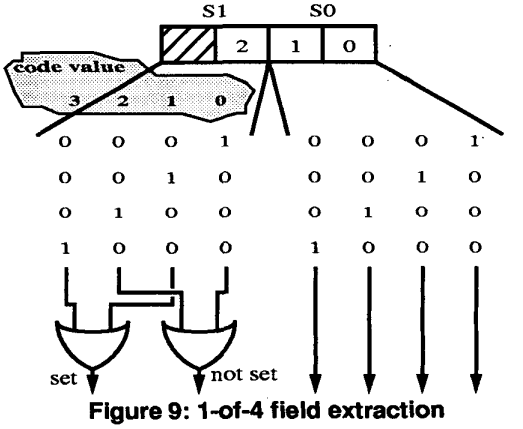

Figure 9: 1-of-4 field extraction

Unfortunately the 1-of-4 approach also resulted in extra costs not incurred by the other data representations. These occur as a consequence of encoding multiple bits within a single codeword. This creates problems in a range of situations: one occurs when the operator requires inputs or outputs which are neither aligned on the appropriate boundaries nor are multiples of two bits. For example, figure 9 illustrates the process of decoding a 3-bit immediate value (bit[0], bit[1] and bit[2]). Two 1-of-4 symbols (S0 and S1) are required to encode the 3-bits. As shown in the figure only the lower digit position (corresponding to bit[2] in symbol (S1)) is required. The upper digit position in S1 will be used to represent information for some other field in the instruction and the state of this should be ignored. The decoding process employs two logical OR operations. The 'set' OR tests to see if bit[2] is set by checking for a '1' in the digit positions corresponding to code values for '3' and '1'in the 1-of-4 symbol. Whereas the

'not set' OR checks digit positions for values '2' or '0'. Neither the dual-rail nor single-rail codes require this decoding.

Extra logic is also required for a shift operation: in both single-rail and dual-rail representations a symbol represents a single bit, so a one place shift is merely a wiring operation; in a 1-of-4 code this requires appreciable logic.

	latency	cycle time	normalized cycle time
single-rail (worst)	**2.29 ns**	**2.29 ns**	**1.00**
dual-rail (best)	1.04 ns	2.14 ns	0.93
dual-rail (worst)	1.26 ns	2.41 ns	1.05
dual-rail (average)	**1.21 ns**	**2.32 ns**	**1.01**
1-of-4 (best)	1.10 ns	1.93 ns	0.84
1-of-4 (worst)	1.14 ns	2.01 ns	0.88
1-of-4(average)	**1.12 ns**	**1.98 ns**	**0.86**
bypass (best)	*0.71 ns*	*1.53 ns*	*0.69*
bypass (worst)	*0.75 ns*	*1.62 ns*	*0.71*
bypass(average)	***0.73 ns***	***1.59 ns***	***0.69***

Table 3: Request activated datapath

7: Results

For each of the different implementations a similar design approach was used. The logic was constructed using gates (static or dynamic) obtained from standard cell libraries. The single-rail implementation used a commercial static standard cell library whereas cell libraries for the dynamic dual-rail and 1-of-4 gates had to be constructed for this project. The same design rules were employed for all circuits based on a VLSI Technology, Inc 0.35 µm three-layer metal process.

The following numbers were obtained by simulating schematics using the EPIC Powermill and Timemill simulation tools. The conditions used for the simulation were 'typical' silicon, Vdd=3.3V and 20°C. The three implementations were excited using the same stimulus, the exception being the removal of the spacer for the single-rail design. The stimulus operated by cycling through all the Thumb instruction formats presenting each format to the circuit whilst varying the number of zeros and ones present in the immediate and register fields. All numbers include the overheads due to the input multiplexor and completion detection (if required).

7.1: Datapath performance

Table 3 summarises the performances for each approach. The cycle times include the time taken to precharge (if required). For single-rail design the worst case performance is presented as this value (plus a significant safety margin) defines the timing of the delay matching circuits.

The 1-of-4 design clearly outperforms the other two approaches when processing immediates and it increases its advantage for formats that are able to bypass the immediate unit.

The performance of the DI implementations depends on instruction format and so will vary according to the particular instruction mix of the code. To simplify the presentation of results, average values are used in table 4.

It can be seen from table 4 that the removal of the keepers for the acknowledge activated design results in a marginal increase in performance compared to the request activated scheme shown in table 3.

	average cycle-times		normalized cycle time
	request activated	acknowledge activated	
dual-rail	2.32 ns	2.28 ns	0.99
1-of-4	1.98 ns	1.95 ns	0.99

Table 4: Datapath cycle times

7.2: Datapath power

Table 5 presents the average power consumed whilst applying the stimulus. The single-rail design consumes the least power which can be attributed to its static operation.

The 1-of-4 design without the power saving optimization consumed about a third less power than dual-rail design. This is slightly worse than the reduction by a half anticipated by table 2. The additional power is consumed by the extra circuitry used to overcome problems encoding multiple bits within a single code word.

However, through the exploitation of the design strengths of 1-of-4 encoding, a further power reduction is obtained. The 1-of-4 with power saving (supporting bypass) consumes less than half the power of the dual-rail design and only slightly more than the single-rail design.

7.3: Datapath size

Results were obtained by comparing the number of transistors in each implementation. These results do not include the additional transistors that would be required for the delay

	request activated	normalized power
single-rail	12 mW	1.0
dual-rail	38 mW	3.2
1-of-4	24 mW	2.0
1-of-4 (power saver)	16 mW	1.3

Table 5: Datapath power consumption

matching circuity in the single-rail design.

The choice of a multiplexor based design prevented the use of the 1-of-4 encoding from supplying the area reductions that were identified in section 6.2. The 1-of-4 implementation only required half as many multiplexers as the dual-rail design, but each of these was twice the size of a dual-rail multiplexor. In addition the 1-of-4 encoding requires considerable extra logic (196 transistors) to implement the one bit shift required for half-word aligning the address offsets. (This is performed as a wiring operation for the other two approaches.)

The elimination of keepers from the domino gates by the application of acknowledge activated control resulted in a 15% reduction in the number of transistors for both the dual-rail and 1-of-4 approaches. This small reduction and a marginal increase in datapath performance were achieved by a large increase in the complexity of the control path. Overall it is probably better to use the request activated control and retain the keepers to improve the noise immunity and thus the robustness of the design.

	request activated	normalized size
single-rail	764	1.0
dual-rail	1201	1.6
1-of-4	1378	1.8

Table 6: Numbers of transistors

8: Conclusions

It is possible to produce asynchronous circuits in a number of styles, each with its own advantages. The appropriate circuit style can vary with the application, however the results of this study suggest that delay insensitive codes have a great deal of promise for fast, safe circuits.

The most surprising result was that the 1-of-4 data representation was not only significantly more power-efficient than the dual-rail style but more power-efficient than expected. This is primarily due to the different circuit costs which facilitated different design optimisations.The major disadvantage of this design style occurs when the required field of a data word does not match a whole number of encoded symbols. If such codes are to be employed the alignment of the symbols is an important configuration – for example in a processor instruction two 'adjacent' bits may not always be the most appropriate choice.

9: Acknowledgements

The VLSI design work has leant heavily on CAD tools from Compass Design Automation (now part of Avant!) and EPIC Design Technology, Inc. (now part of Synopsis).

The authors would also like to thank other members of the AMULET research group at Manchester University.

10: References

[1] Jaggar, D., "Advanced RISC Machines Architecture Reference Manual". Prentice Hall, 1996. ISBN 0-13-736299-4.

[2] Verhoeff T, "Delay-insensitive codes - an overview", Distributed Computing vol 3. 1998.

[3] Abrial A, Bouvier J, Renaudin, M, Vivet P "A Contactless Smart-Card Chip based on an Asynchronous 8-bit Microcontroller" Proc. 4th ACID Workshop, Grenoble, France, January 31st, 2000.

[4] Williams T, Horowitz M, "A Zero-Overhead Self-Timed 160 ns 54b CMOS Divider", IEEE Journal of Solid State Circuits 26(11) November 1991.

[5] Chou W, Beerel P. A, Ginosar R, Kol R, Myers C. J, Rotem S, Stevens K, Yun K.Y, "Average-case optimized technology mapping of one-hot domino circuits" Proc. Asynch'98, 1998.

[6] Furber S, Liu J, "Dynamic Logic in Four-Phase Micropipelines", Proc. Async 96, Aizu-Wakamatsu, Japan, March 18-21 1996.

[7] Dally W.J, Poulton J. W, "Digital Systems Engineering". Cambridge University Press, 1998. ISBN 0 521 59292 5.

[8] Singh M, Nowick S, "High-Throughput Asynchronous Pipelines for Fine-Grain Dynamic Datapaths", Proc. Aync'2000, Eilat, April 2000.