A Real-time, Pipelined Incoherent Dedispersion Method and Implementation in FPGA

Wei Liu1, Qiao Meng1,3, Chen Wang2, Chenye Zhou1, Shunyu Yao1, and Irfan Tariq1

1 School of Information Science and Engineering, Southeast University, Nanjing, Jiangsu, People’s Republic of China
2 National Astronomical Observatories, Chinese Academy of Sciences, Beijing 100012, People’s Republic of China

Received 2021 June 24; accepted 2021 November 11; published 2022 January 31

Abstract

In pulsar observation, dispersion occurs due to the interstellar medium. The dispersion significantly affects the detection of pulsar signals. To overcome the dispersion effect, incoherent dedispersion methods are often applied. The traditional incoherent dedispersion methods are computationally expensive and troublesome. To deal with this problem, in this paper, we developed a Real-Time, Pipelined Incoherent Dedispersion Method (RT-PIDM). RT-PIMD only caches the summed-up time series, instead of all the frequency spectra, so the memory consumption is determined by the number of DM trails, whereas the traditional method’s memory consumption is determined by the number of frequency channels. In most of the situations, the number of frequency channels is several times more than that of DM trails, which means the memory consumption of traditional methods is more than that of RT-PIDM. With RT-PIDM, we designed a 1.2 GHz bandwidth prototype digital backend, and we finished pulsar observation with the 40 m radio telescope at Yunnan Observatory. The results demonstrate that the RT-PIDM can be implemented inside a single FPGA chip with less Block RAM, and the proposed RT-PIDM dedisperses the pulsar signal in real time and achieves the same result as compared to traditional incoherent dedispersion.

Unified Astronomy Thesaurus concepts: Radio pulsars (1353); Astronomical instrumentation (799); Transient detection (1957)

1. Introduction

Pulsars are highly magnetized neutron stars that rotate rapidly (Hewish et al. 1968; Bonazzola & Gourgoulhon 1996; Abbott et al. 2010; Caraveo et al. 2003; Stinebring et al. 2000). Pulsars produce radio beams, which sweep the sky like a lighthouse (Zhu-Xing et al. 2014). If the beam is oriented toward the earth, it will produce periodic pulses, which can be measured with radio telescopes and dedicated backends. During their travel through installer medium (ISM), these pulses are attenuated and speared over time (higher frequencies will arrive earlier than lower frequencies). This phenomenon is called dispersion, which makes these pulses hard to be detected without further processing. Meanwhile, the improvements on the observing bandwidth (Deneva 2007; Siemion et al. 2010; Baffa 2014; Vertatschitsch et al. 2015; Liu et al. 2018) and multi-beam receivers (Manchester et al. 2001; Lorimer et al. 2015) result in a large amount of data, so real-time dedispersion is vital in pulsar observation. Traditionally, two techniques are used to dedisperse the pulsar signals: coherent dispersion and incoherent dedispersion. Coherent dedisperpse technique deconvolws the transfer function of dispersion from the complex voltage signal. Incoherent dedispersion technique aligns the small frequency channels in time.

Previously, a lot of astronomical instruments with real-time dedispersion have been developed. In the mid 1980 s, Hankins et al. developed the first real-time coherent dedispersion system at the Arecibo observatory using a hardware-based implementation (Hankins & Rajkowski 1987). In their system, a quad-chipred transversal filter with 2MHz bandwidth was used. Similarly, D.C Backer et al. developed the Berkeley Pulsar Processor (BPP) instrument, which used a CMOS 1024-tap transversal filter. BPP finishes the real-time coherent dedispersion, with 100–200 MHz total bandwidth (maximum) and 50–100 channels. (Backer et al. 1996).

As the conventional radio astronomy instruments are highly specialized, with custom, complex, dedicated for individual applications, it requires 3–5 yr for designing and debugging the system (Parsons et al. 2006). Therefore, some new technologies are necessary for radio astronomy. Since the pioneering work of Weinreb (Weinreb 1961), digital processing hardware is widely used in radio astronomy. In the past decades,
Field-Programmable Gate Arrays (FPGAs), Central Processing Units (CPUs), and Graphics Processing Units (GPUs) are used as the cores of the radio astronomy instruments. These digital devices are re-programmable and flexible in developing and debugging, which can help reduce the design time of new instruments.

With the digital processing hardware, some new digital backends were developed for new telescopes with wideband receivers in the recent years. The PuMa-II backend at the Westerbork Synthesis Radio Telescope (WSRT) demonstrated that a CPU cluster can be used to implement a near real-time coherent dedispersion system for a bandwidth of 160 MHz, by dividing the bandwidth into 8 sub-bands, and then dispersing and combining the signal (Karuppusamy et al. 2008). The Green Bank Ultimate Pulsar Processing Instrument (GUPPI), functioning on FPGAs, a CPU cluster and a GPU cluster of 8 GPUs, implements a wide bandwidth coherent dedispersion system supporting a total bandwidth of 800 MHz (Ford et al. 2010). A fully real-time coherent dedispersion system based on GPUs has also been developed for the pulsar backend at the Giant Metrewave Radio Telescope (GMRT) (Kishalay & Yashwant 2016).

In the past few years, CPUs and GPUs are important for coherent dedispersion, which is computationally expensive. Compared to coherent dedispersion, incoherent dedispersion does not get the high time resolution (Hanksins 2017), but it requires less computational resources. Therefore, some researchers implemented real-time incoherent dedispersion in an FPGA. Clarke, N et al. implemented a new incoherent dedispersion method optimized for FPGA-based architectures intended for deployment on the Australian SKA Pathfinder and other Square Kilometre Array precursors for fast transients surveys (Clarke et al. 2014). Even though the implementation of incoherent dedispersion is simpler than that of coherent dedispersion, it still requires some memories for caching the spectra data. The memory requirement is mainly determined by the number of frequency channels.

To get the better performance, more frequency channels are required for incoherent dedispersion, which means the size of Block RAM in FPGA is not large enough for the incoherent dedispersion. In this paper, we present a Real-Time, Pipelined Incoherent Dedispersion Method (RT-PIDM). The memory requirement for RT-PIDM is determined by the number of DM trails, instead of the number of frequency channels, so that the real-time incoherent module can be implemented in a single FPGA without external memory. With this feature, we designed a 1.2 GHz bandwidth digital backend with a FPGA-based, real-time incoherent dedispersion module. It is implemented in a single Xilinx Virtex-6 LX240T FPGA chip, so it is compact and power-saving.

In the following sections, we present a digital backend with the real-time, pipelined incoherent dedispersion module. In Section 2.1, we present the signal processing architecture implemented in a single FPGA. The detailed comparison between coherent dedispersion and incoherent dedispersion is shown in Section 2.2.1. The real-time, pipelined incoherent dedispersion method and its implementation is shown in Sections 2.2.2 and 2.2.3. The experiment results at Yunnan Observatory are shown in Section 3. The paper closes with conclusion and acknowledgments.

2. Design of the Digital Backend

2.1. System Architecture

Figure 1 depicts the signal processing architecture, which is made up of seven distinct units. The Data Interface Unit is used
for dividing the high-speed data stream into eight separate data streams. The Parallel Fast Fourier Transform (FFT) Unit is used for parallel channelization. The Power Calculation Unit is used to calculate the power of the input signals. The Accumulation Unit is used for power accumulation. For summing or separating the power of left and right polarized signals, the Add or Independent Unit is used. The digital backend has two output data units, Ethernet Unit and Real-Time Dedisperison (RTD) Unit. The Ethernet Unit is used for the output of raw channelized data, which is searching data in this paper, while the RTD unit in FPGA is used for real-time dedispersion. Each RTD unit is used for one DM trail, hence for multiple DM trails, multiple RTD instances are required, which will create multi output RTD data streams. Due to the fact that the size of dedispered data from RTD unit is much smaller than that of the searching data, it can be used for real-time searching for pulsars and fast radio transients. As the parallel FFT unit has been introduced in another paper (Liu et al. 2017), we will mainly discuss about the newly developed real-time dedispersion module in FPGA in this paper.

2.2. Real-Time, Pipelined InCoherent Dedisperison Module

2.2.1. Coherent Dedisperison and InCoherent Dedisperison

Pulsar observations are distorted by the deleterious effects of dispersion. When an electromagnetic wave travels through the ionized interstellar medium, signal at low frequencies travels slower than that at higher frequencies due to dispersion, which causes a broadband sharp pulse from the source to be smeared out in time, when detected with a receiver having a finite bandwidth. Hence, proper correction techniques are vital for pulsar observations. As the wide observing bandwidth brings large amount of data, it is difficult to do offline data processing, so it is necessary to do real-time dedispersison.

Two techniques are widely used to eliminate the effect of dispersion: Coherent dedisperison and Incoherent dedisperison (Lorimer & Kramer 2012). In incoherent dedisperison, the observable bandwidth is divided into different channels, and the dedisperison delay of each channel is

\[
delay = 4.16 \times 10^6 \times DM \times \left( \frac{1}{v_0^2} - \frac{1}{(v_0 + \Delta v)^2} \right) \text{ms},
\]

where, \( v_0 \) is the lowest frequency, and \( \Delta v \) is the bandwidth of each frequency channel. DM is the dispersion measure. The delays are applied to align the pulses in the separate frequency channels, which are summed up to produce a dedispersed time series.

Coherent dedisperison removes the dispersive effect of the interstellar medium by convolving the digitally sampled telescope output voltages with an inverse filter function, derived from the tenuous, cold plasma dispersion law (Hankins & Rickett 1975)

\[
H(v + v_0) = \exp\left( i \frac{2\pi Dv^2}{v_0^2} \right),
\]

where, \( v_0 \) is the center frequency of the observation, \( v \) is the bandwidth of the pulsar signal, and \( D \) is related to DM

\[
DM \left( \text{pc cm}^{-3} \right) = 2.41 \times 10^{-16} \times D(\text{sec Hz}^2)
\]

Coherent dedisperison completely eliminates the dispersion effect, but it is more computationally expensive. On the other hand, incoherent dedisperison is not too expensive, but it does not remove the intra-channel dispersion. If we compare coherent dedispensison with incoherent dedisperison, we can find that if the intra-channel dispersion smearing time is smaller than time resolution, coherent and incoherent dedisperison will have the same performance. From Equation (1), we know the intra-channel smearing time is mainly related to \( \Delta v \). The small \( \Delta v \) reduces intra-channel smearing time. However, this requires a large number of frequency channels across the bandwidth, which will decrease the time resolution. Therefore, we need a trade off between intra-channel smearing time and time resolution.

Following the nyquist sampling theory, if the sampling frequency is \( f_s \), the observing bandwidth is \( f_s/2 \). The number of frequency channels is \( N_v \), so the bandwidth of each frequency channel is

\[
\Delta v = \frac{f_s}{2N_v}
\]

The highest time resolution of the system is

\[
\Delta t = \frac{2N_v}{f_s} = \frac{1}{\Delta v}
\]

In order to search for millisecond pulsars with microsecond pulse width (Backer et al. 1982; Wolszczan & Frail 1992), time resolution should be microsecond scale. For example, the time resolution \( \Delta t \) can be chosen to be 64 \( \mu s \), so the frequency resolution \( \Delta v \) is 15.625 kHz. To get the same performance as coherent dedisperison, the intra-channel smearing time in incoherent dedisperison should be smaller than the time resolution. With a specified \( \Delta v \), the intra-channel smearing time can be calculated by Equation (1), which depends on \( v_0 \) and DM. The relationship between \( v_0 \) and DM is shown in Figure 2. For example, with the chosen \( \Delta t = 64 \mu s \) and \( \Delta v = 15.625 \text{kHz} \), if the observation starts from 2000 MHz, coherent dedisperison and incoherent dedisperison will have the same performance, when \( DM \) is smaller than 3938.51 pc cm\(^{-3}\).

2.2.2. Real-Time, Pipelined Incoherent Dedisperison Method

Before describing the RT-PIDM, first we will review the widely used direct dedisperison algorithm (Barsdell et al. 2012). The direct
The dedispersion algorithm works by directly summing frequency spectra along a quadratic dispersion trail. For each time sample and dedispersion trail, the algorithm computes an array of dedispersed time series $D$ from an input spectra data set $A$. The dedispersed data is

$$D_{d,t} = \sum_{v=0}^{N_v-1} A_{v,t+\Delta t(d,v)}$$

where, $d$ represents dispersion measure, $t$ represents time when the time series data is produced, and $v$ represents frequency channel. $N_v$ is the total number of the frequency channels, and $A_{v,t}$ is the dynamic spectra of frequency channel in time series.

$\Delta t(d, v)$ is the discretized delay for the $v$ with the specified $d$

$$\Delta t(d, v) = \text{round} \left( \frac{\text{delay}_{n,v}}{\Delta t} \right)$$

(7)

where, delay$_{n,v}$ is calculated by Equation (1), and $\Delta t$ is the sampling time of the system. round is the integer function.

Incoherent dedispersion based on direct incoherent dedispersion algorithm access the frequency spectra many times to compute the dedispersed time series for different $DM$ trials, so a large amount of memory is necessary for caching the frequency spectra data. When the $N_v$ and $d$ increase, the memory size will also increase. No matter how many $DM$ trails, the memory consumption always
necessary for the incoherent dedispersion work. The reason why an external memory chip is almost always the total Block RAM inside an inexpensive FPGA chip, that is spectra data. The required size of memory is much larger than the spectral data at the same time will be summed up. The Bytes. As the dedispersed result is a time series, the aligned data is 8 bits, the total buffer size for direct dedisperison is 12 the spectral data is 3 \text{ at different times. Therefore, the size of the buffer for caching (up all the frequency spectra as dedispered data.}

A simple example about the direct incoherent dedispersion is shown in Figure 3. In the example, \( N_v \) is equal to 4. Because dispersion delay is nonlinear in time, we set \( \Delta t(d, v) \) like this in this example: \( \Delta t(d, 0) = 0, \Delta t(d, 1) = 1, \Delta t(d, 2) = 2, \) and \( \Delta t(d, 3) = 2. \) The blocks with different colors are the spectral data at different times. Therefore, the size of the buffer for caching the spectral data is \( 3 \times 4 = 12. \) If the bit width of each spectral data is 8 bits, the total buffer size for direct dedisperion is 12 Bytes. As the dedispersed result is a time series, the aligned spectral data at the same time will be summed up. The final dedispered result is shown in Figure 4.

The proposed RT-PIDM completes the incoherent dispersion in a more efficient way. As the final dedispersed result is a time series, RT-PIDM only caches the final time series, instead of all the spectral data, so that it can save a lot of memory. The block diagram of the RTD module based on RT-PIDM is shown in Figure 5. The RTD module consists of a calculation unit, a \( \text{RAM}_{\Delta t} \) and a \( \text{RAM}_D. \) The calculation unit is used for setting different delays for different spectral data, and summing up all the frequency spectra as dedispersed data. \( \text{RAM}_D \) is a RAM for caching the dedispersed data. \( \text{RAM}_{\Delta t} \) is a RAM for storing discretized delays for different frequency channels.

The calculation unit is the core part in the RTD module. As the spectral data are transferred to the RTD module from low frequency channels to high frequency channels, two important registers are in the calculation unit: \( \text{BaseAddr} \) and \( \text{OffsetAddr}. \) The \( \text{BaseAddr} \) register contains the address of the memory cell in \( \text{RAM}_D \), and it will increase by one, when a new set of \( A_{v,j} \) is coming. Different frequency channels have to be set different delays, so \( \text{OffsetAddr} \) is used for setting the offset address to \( \text{BaseAddr} \), which is the dispersion delay in the RT-PIDM. The value in \( \text{OffsetAddr} \) is read from \( \text{RAM}_{\Delta t}. \) The figure about the two registers are shown in Figure 6.

Below, we show how the RTD module works:

(a) At the beginning, \( \text{BaseAddr} \) and \( \text{OffsetAddr} \) are set to 0.
(b) When a new set of \( A_{v,j} \) is coming, \( \text{BaseAddr} \) will increase by one.
(c) Discretized dispersion delay for \( v_0 \) will be got from \( \text{RAM}_{\Delta t} \) as \( \text{OffsetAddr}. \)
(d) The data from \( \text{RAM}_D \) at \( \text{BaseAddr} + \text{OffsetAddr} \) will be read out, and added with \( A_{v,0}. \) Then the result will be written back to the same address.
(e) Repeat the step (c) and (d) for \( v_1, v_2...v_{N_v-1}. \)
(f) Read the data out from \( \text{RAM}_D \) at \( \text{BaseAddr} \), which is the dedispersed data at \( t. \)
(g) Repeat the step (b) to (f), so that the RTD module can output the dedispersed data continuously.

\[
\text{MemSize}_{\text{Direct}} = N_v \times \Delta t(d_{\text{max}}, v_{\text{max}}) \times res \text{ bits} \quad (8)
\]

where, \( N_v \) is the number of frequency channels, \( \Delta t(d_{\text{max}}, v_{\text{max}}) \) is the maximum discretized delays, and \( res \) is the bit width of spectra data. The required size of memory is much larger than the total Block RAM inside an inexpensive FPGA chip, that is the reason why an external memory chip is almost always necessary for the incoherent dedispersion work (Clarke et al. 2014).
With the same dispersed signal shown in Figure 3, an example about how the RTD module based on RT-PIDM works is shown in Figure 7. The buffer size for RT-PIDM is 3, which is the dashed box shown in Figure 7. If the bit width of spectral data is still 8 bits, the buffer size will be 3.75 Bytes. The buffer for RT-PIDM has three memory cells, which are addressed from 0 to 2, and they are used for caching the dedispersed data at $t_0$ and $t_2$ at the beginning. White blocks shown in Figure 7 is produced before $t_0$, so we can ignore them.

Next, we show how the RT-PIDM works:

(a) At $t_0$, $A_{0.0}$ (red block) is coming. $A_{0.0}$ will be added to Addr0.

(b) At $t_1$, $A_{0.1}$ (yellow block) is added to Addr1. As $A_{1.1}$ (red block) is the spectra data produced at $t_0$, but delayed $\Delta t$ due to dispersion, it should be added to Addr0.

(c) At $t_2$, $A_{0.2}$ (blue block) is added to Addr2. $A_{1.2}$ (yellow block) is delayed $2\Delta t$ due to dispersion, so these two spectra data should be added back to Addr1 and Addr0. Now, all the spectral data produced at $t_0$ have been added to Addr0, which means the data cached at Addr0 is the dispersed data for $t_0$, and it should be the output of RT-PIDM. Then the data stored at Addr0 is reset to 0.

(d) At $t_3$, new spectral data is coming. The buffer is working as a circular buffer, so $A_{0.3}$ (green block) is added to Addr0. The same as before, $A_{1.3}$ and $A_{3.3}$ are added back to Addr2-Addr1. The dedispersed data produced at $t_1$ is at Addr1, which is the output of RT-PIDM now.

(e) At $t_4$, the same as before, the dedispersed data for $t_2$ is the output of RT-PIDM, and it is at Addr2.

As we discussed before, the memory consumption for RT-PIDM is only the $RAM_D$ and $RAM_{\Delta t}$ in the RTD module. $RAM_D$ is used for caching the dedispersed data, so the depth of $RAM_D$ is $\Delta t(d_{\text{max}}, v_{\text{max}})$. Because the sum of frequency spectra is cached in the $RAM_D$, the bit width of the RAM should be $(\log_2 N_e + \text{res})$ bits. Hence, the memory consumption of $RAM_D$ is

$$\text{MemSize}_{RAM_D} = \Delta t(d_{\text{max}}, v_{\text{max}}) \times (\log_2 N_e + \text{res}) \text{ bits}$$  \hspace{1cm} (9)
Figure 8. Photo of the compact parts (This is the photo of the Craboard), including an ADC board, an FPGA board and an ARM board.

Figure 9. Block diagram of the CRABoard. The sampling rate is 2400 MSps, and the frequency of input signal ranges from 100 MHz to 1200 MHz.

Table 2
The Maximum Number of DM Trails on different Hardware Platforms

| Platform | FPGA Part Number | Block RAM Size (MB) | Maximum number of DM trails |
|----------|------------------|---------------------|----------------------------|
| ROACH2   | XC6VSX475T-1FFG1759C | 4.75                | 81                         |
| SKARAB   | XC7VX690T-2FFG1927C | 6.625               | 115                        |
| SNAP2    | XCKU115-FLVF1924   | 9.5                 | 165                        |

Figure 10. Block diagram of RTD module.
As $\text{RAM}_{\Delta t}$ is used for storing the discretized delays for all the frequency spectra, the depth of the RAM is $N_v$, and the bit width of the $D_{\text{max}, \text{max}}(d_{\text{max}}, v_{\text{max}})$. Therefore, the memory consumption of $\text{RAM}_{\Delta t}$ is

$$\text{MemSize}_{\text{RAM}_{\Delta t}} = N_v \times \log_2 \Delta t(d_{\text{max}}, v_{\text{max}}) \text{ bits}$$  \hspace{1cm} (10)$$

The total memory consumption for RT-PIDM is

$$\text{MemSize}_{\text{RT-PIDM}} = \text{MemSize}_{\text{RAM}_{\Delta t}} + \text{MemSize}_{\text{RAM}_{\Delta t}}$$ \hspace{1cm} (11)$$

Comparing $\text{MemSize}_{\text{Direct}}$ and $\text{MemSize}_{\text{RT-PIDM}}$, we can find the difference is that $\text{MemSize}_{\text{Direct}}$ is proportional to $N_v$, and $\text{MemSize}_{\text{RT-PIDM}}$ is mainly proportional to $\log_2 N_v$. Table 1 shows the comparison between $\text{MemSize}_{\text{Direct}}$ and $\text{MemSize}_{\text{RT-PIDM}}$ with fixed $\text{res}$ and $t(d_{\text{max}}, v_{\text{max}})$. From Table 1, we can see RT-PIDM consumes much less memory than the direct method.

If there are more than one $DM$ trail, we need to have more RTD instances in FPGA design, and each RTD instance will have its own memory space, so that all the RTD instances can work parallely. For example, $N_v$ is $9 \times 304 = 2754$, $\text{res}$ is 16 bits, and $\Delta t(d_{\text{max}}, v_{\text{max}})$ is 16384, which covers the max DM of 2800, when $\Delta t = 64 \mu s$, the lowest frequency of observation is 2000 MHz, and the bandwidth is 500 MHz. Then the block memory size for each RTD instance is 58.70 KB. Because of the limitation of Block RAM resource on a single FPGA chip, the maximum number of DM trails depends on the FPGA chip. Table 2 shows that the maximum number of DM trails on some popular CASPER boards. Therefore, around 100 RTD instances can be implemented on a real-world digital backend. More powerful digital backends with more Block RAM resource may be available in the future, allowing more RTD instances to be implemented on a single FPGA chip.

A digital backend with the same settings ($N_v = 2754$, $\text{res} = 16$ bits, $\Delta t(d_{\text{max}}, v_{\text{max}}) = 16,384$) has been deployed at SKA, which is used for for real-time, multi-beam transient searches (Clarke et al. 2014). The number of $DM$ trials is 448, so the FTA (Frequency-Time Array) of the SKA digital backend occupies 85.5 MBytes, which is based on the direct method.

---

4 https://casper.astro.berkeley.edu/wiki/Hardware
The Definition of Inputs and Outputs shown in the Calculation Unit

| Inputs/Outputs | Definition |
|----------------|------------|
| State          | The calculation unit is based on a state machine. 20 states are shown here. |
| en_sync        | This signal indicates that the spectra data are valid or not. |
| Data_In        | These are spectral data. |
| SOF            | This signal indicates that the valid data are coming, and it is a delay of en_sync. |
| cnt_sync       | The signal indicates the order of spectral data. |
| BaseAddr       | This is a register, caching address of the memory cell for dedispersed data. |
| OffsetAddr     | This is a register for setting different delays for different frequency channels. |
| RAM_D_ena      | This is the enable signal for reading data from RAM_D. |
| RAM_D_raddr    | This is the reading address of RAM_D, and it is the sum of BaseAddr and OffsetAddr. |
| RAM_D_wea      | This is the enable signal for writing data to RAM_D. |
| RAM_D_waddr    | It’s the writing address of RAM_D, and it is four-clock delay of RAM_D_raddr. |
| F_Addr_Equal   | This signal indicates whether RAM_D_raddr and RAM_D_waddr are the same. |
| RAM_D_din      | This is the input of RAM_D. |
| RAM_D_dout     | This is the output of RAM_D. |
| DATA_In_D6     | This is the 6-clock delay of Data_In. It’s for synchronization. |
| Dvalid         | This signal indicates that the output data of calculation unit is valid. |
| DD_Data        | This is dedispersed data from calculation unit, and it is synchronized to Dvalid. |

With RT-PIDM, the memory consumption for 448 DM trails is 25.68 MB, which is 30.03% of the memory consumption in the digital backend at SKA.

2.2.3. Implementation

We implemented one RTD module on our Cascaded Reconfigurable Architecture Board (CRABoard) shown in Figures 8 and 9 (Liu et al. 2021) to demonstrate the RT-PIDM works.

The implementation of the RTD module described in this paper is shown in Figure 10, which consists of a software part (ARM) and a hardware part (FPGA). The discretized delays, $\Delta t(d, \nu)$, are necessary in incoherent dedispersion, and they are determined by a specific $DM$ and lowest frequency of observation. As $\Delta t(d, \nu)$ should be calculated and set to hardware before the RTD module starts to work, and no real-time requirements are necessary for it, it will be easy to compute $\Delta t(d, \nu)$ in an ARM or a host computer with Equation (1) and (7). We finished the computation on the ARM module, and then write the $\Delta t(d, \nu)$ to a Block RAM in FPGA through data bus. We could also compute the delays on host computer, and then write them to FPGA via Ethernet port. It’s fast to compute the $\Delta t(d, \nu)$, so the computation delay can be ignored. After the RTD module starts to work, the computation is in real time.

In our prototype design, the sampling frequency is 2.4 GSPs, and the number of frequency channel is 2048, so $\Delta \nu$ of the system is 585.9 kHz. The time resolution is adjustable, which is from 34.1 to 243 $\mu$s. We did the experiments at Yunnan Observatory with the receiver starting from 2190 MHz, so the maximum $DM$ is

$$DM_{\text{max}} = \frac{243 \, \mu s}{4.16 \times 10^6 \times \left( \frac{1}{2190} - \frac{1}{(2190 + 0.5859)^2} \right)}$$

$$= 523.8 \, \text{(pc cm}^{-3})$$

(12)

Therefore, the maximum delay in the whole bandwidth is 240.1 ms, so $\Delta t(d_{\text{max}}, \nu_{\text{max}})$ here is 988, which requires a RAM_D with depth of 1024. The bit width of spectra data is 8 bits, so the memory consumption of the RTD module in our prototype design is

$$2048 \times 10 + 1024 \times 19 \text{ bits} = 4.875\text{KB}$$

(13)

$A_{\nu, t}$ is transferred to the RTD module along with other two signals: en_sync and cnt_sync. en_sync is used for indicating that the spectral data is valid, so the rising edge of the signal can be used to indicate that the new set of $A_{\nu, t}$ is coming. cnt_sync is used to indicate the order of the spectral data. For example, if cnt_sync is equal to $n$, it means the spectral data is $A_{\nu, t, n}$. The relationship between $A_{\nu, t}$, en_sync and cnt_sync is shown in Figure 11.

Calculation unit finishes the core dedispersion, and the core part implementation of the unit is shown in Figure 12. In the real implementation, there are four important registers in this unit: BaseAddr, OffsetAddr, $D_{\nu, t, +\Delta t}^d$ and $D_{\nu, t, +\Delta t}^r$. BaseAddr is the base address, which increases by one in every calculation round. OffsetAddr is read from RAM_D at address cnt_sync, which is synchronized to $A(\nu, t)$. For example, when $A(\nu, t)$ comes, cnt_sync is equal to $n$, so $\Delta t(d, \nu)$ will be read to the calculation unit as OffsetAddr. The sum of BaseAddr and OffsetAddr is raddr, which connects to the AddrB port on RAM_D for reading $D_{\nu, t, +\Delta t}^r$. Now, the reading out operation will take several taps of clock, the $A(\nu, t)$ will be delayed the same taps of clock for the synchronization, and then added to $D_{\nu, t, +\Delta t}^d$, as $D_{\nu, t, +\Delta t}^r$. As we mentioned in Section 2.2.2, the $D_{\nu, t, +\Delta t}^r$ will be written back to the same address, so waddr is several taps delay of raddr.
For lower DMs, some values of $\Delta t(d, v)$ read out from $RAM_{\Delta t}$ will be the same. However, reading and writing operations on the $RAM_{D}$ take some time, which means $D_{in_{d,t+\Delta t}}$ will be read out before the latest value updated to the $RAM_{D}$. To solve this issue, the $raddr$ and $waddr$ will be compared continuously. If $raddr$ is equal to $waddr$, a signal named $F_{Addr\_Equal}$ will be high, which is the selection signal for $D_{sum}$. More details about the definition of the inputs and outputs are shown in Table 3, and the timing diagram is shown in Figure 13.

### 3. Observation Results

On 2021 June 17th, we finished pulsar observation with the 40 m radio telescope at Yunnan Observatory. During the experiments, we stored RTD data from RTD unit and searching data from Parallel FFT unit. A Pulsar Digital Filter Bank (PDFB) is also deployed at Yunnan Observatory, which is developed by the Australia Telescope National Facility (ATNF) in 2008. It is a system capable of processing up to 1 GHz of bandwidth in four primary modes of operation: folding, search, spectrometer and baseband outputs (Hampson et al. 2008). The PDFB was also used for observing the same pulsar at the same time, so that we can compare the signal-noise to ratio (S/N) of the profile from our Craboard and the PDFB. Some information of the pulsar we observed is shown in Table 4, and the timing diagram is shown in Figure 13.

#### Table 4

| Parameters | Values |
|------------|--------|
| NAME       | J0835-4510 |
| DM         | 67.97 |
| PEPOCH     | 51559.319 |
| P0         | 0.089328385024s |

For lower DMs, some values of $\Delta t(d, v)$ read out from $RAM_{\Delta t}$ will be the same. However, reading and writing operations on the $RAM_{D}$ take some time, which means $D_{in_{d,t+\Delta t}}$ will be read out before the latest value updated to the $RAM_{D}$. To solve this issue, the $raddr$ and $waddr$ will be compared continuously. If $raddr$ is equal to $waddr$, a signal named $F_{Addr\_Equal}$ will be high, which is the selection signal for $D_{sum}$. More details about the definition of the inputs and outputs are shown in Table 3, and the timing diagram is shown in Figure 13.

### 3. Observation Results

On 2021 June 17th, we finished pulsar observation with the 40 m radio telescope at Yunnan Observatory. During the experiments, we stored RTD data from RTD unit and searching data from Parallel FFT unit. A Pulsar Digital Filter Bank (PDFB) is also deployed at Yunnan Observatory, which is developed by the Australia Telescope National Facility (ATNF) in 2008. It is a system capable of processing up to 1 GHz of bandwidth in four primary modes of operation: folding, search, spectrometer and baseband outputs (Hampson et al. 2008). The PDFB was also used for observing the same pulsar at the same time, so that we can compare the signal-noise to ratio (S/N) of the profile from our Craboard and the PDFB. Some information of the pulsar we observed is shown in Table 4.
in Table 4, and the information is taken from ATNF pulsar database (Manchester et al. 2005). In the experiment, we set the time resolution to 34.1 μs.

The folded result of RTD data is shown in Figure 14, and we can see a peak and a shoulder shown in the figure. To demonstrate the RTD module works, we also did the offline dedispersion on the searching data, which contains all the frequency spectra. Because we used the customized data file

---

[Figure 15. The processed result of raw spectra data: (a) The folded result of J0835-4510 with dedispersion in Phase-Frequency figure. (b) The profile of J0835-4510 with dedispersion.]

[Figure 16. (a) The details of the RTD result. (b) The details of the searching data result with offline dedispersion.]

---

https://www.atnf.csiro.au/research/pulsar/psrcat/
format, we have the folding and dedispersion code in Matlab for data processing, which is available on github.6 The results are shown in Figure 15.

To compare with the RTD result and searching data result, we zoomed in the Figure 14 and Figure 15 (b) to get more details, which are shown in Figure 16. The comparison results are shown in Table 5. The S/N, pulse width and offset between the peak and shoulder are almost the same. To make the further comparison, we put the normalized searching result and RTD result in one figure, and calculated the error between the two results. The comparison result is shown in Figure 17. The comparison result and error between the two normalized results prove that the RT-PIDM has the same result as the searching result.

Because the RTD data is just a time series and searching data is the frequency spectra, the data speeds of RTD data and searching data are

\[
\text{DataSpeed}_{\text{RTD}} = \frac{1 \text{s}}{34.1 \mu s} \times (\log_2 2048 + 8) \text{ bits} \\
= 544.1 \text{Kbps}
\]

\[
\text{DataSpeed}_{\text{SearchingData}} = \frac{1 \text{s}}{34.1 \mu s} \times 2048 \times 8 \text{ bits} \\
= 458.2 \text{Mbps}
\]

The size of the RTD data is about 862 times smaller than that of the searching data, which makes it easier to process the RTD data than process the searching data.

We stored the RTD data and searching data for 30 s. RTD data is approximately 2 MB in size, while searching data should be approximately 1.7 GB in size. Because the bandwidth of the Craboard is 1200 MHz, while the bandwidth of the receiver at Yunnan observatory is 110 MHz, a lot of data is invalid. Therefore, we only stored the valid spectral channels in the data file, which takes up around 200 MB for the 30 s data. The RTD data and the searching data are processed in Matlab, which takes about 0.9 s for the RTD data and about 6.5 s for the recorded searching data. The processing time will be much greater if full bandwidth searching data is recorded. The

\[\text{DataSpeed}_{\text{RTD}} = \frac{1 \text{s}}{34.1 \mu s} \times (\log_2 2048 + 8) \text{ bits} \]

\[= 544.1 \text{Kbps}\]

\[\text{DataSpeed}_{\text{SearchingData}} = \frac{1 \text{s}}{34.1 \mu s} \times 2048 \times 8 \text{ bits} \]

\[= 458.2 \text{Mbps}\]

Table 5

| Parameter                        | RTD Result | Searching Data Result |
|----------------------------------|------------|-----------------------|
| Signal-Noise to Ratio (S/N)      | 94.8398    | 94.877                |
| Pulse Width                      | 0.0605     | 0.0605                |
| Offset between the Peak and Shoulder | 0.021     | 0.021                |

RTD data, searching data, and the processing matlab code are all available on github.7

The PDFB results are shown in Figure 18, which are processed by PSRCHIVE (Hotan et al. 2004). The profile from PDFB looks almost the same as that from RT-PIDM, but we did not find any information about the S/N algorithm implemented in PSRCHIVE, so we read the PDFB data with Matlab,8 and then plotted the figure and computed the S/N in Matlab as we did on RTD data and searching data. The S/N algorithm used in Matlab is shown in Appendix. The processed result and detailed figure are shown in Figure 19.

The comparison result between Figure 16(a) and Figure 19(b) is shown in Table 6. The S/N, pulse width and offset between the peak and shoulder on the profile are also almost the same. Therefore, the comparison result demonstrates that the RT-PIDM implemented on the Craboard has the same result as the PDFB has.

\[\text{Table 6}\]

| Parameter                        | RTD Result | PDFB Result |
|----------------------------------|------------|-------------|
| S/N                              | 94.8398    | 91.4712     |
| Pulse Width                      | 0.0605     | 0.06054     |
| Offset between the Peak and Shoulder | 0.021     | 0.0214      |

6 https://github.com/liuweiseu/Searching_data_code.git
7 https://www.github.com/liuweiseu/pulsar_data_code.git
8 https://www.mathworks.com/help/matlab/ref/fitsread.html
The experiment results on J0835-4510 at Yunnan observatory show that the results from RT-PIDM, offline dedispersion and PDFB are consistent. With the proposed RT-PIDM, we can get the same dedispersion result on a compact and power-saving platform.

4. Conclusion

In this paper, we proposed a Real-Time, Pipelined Incoherent Dedispersion Method (RT-PIDM). With the pipelined computing architecture, only summed-up time series is stored in memory, so the memory consumption of RT-PIDM
is smaller than that of traditional methods, which makes it possible to be implemented inside a single FPGA. With the feature of RT-PIDM, we preset a prototype digital backend with 1200 MHz observing bandwidth. The results got at Yunnan Observatory demonstrate that RT-PIDM has the same result as offline incoherent dedisersion and the PDFB deployed at Yunnan observatory has. In the future, RT-PIDM will be useful for pulsar and fast transient search on some compact hardware platforms.

This work was supported by The National Natural Science Foundation of China under grant U1731120. The authors gratefully thank Yuxiang Huang, Longfei Hao et al. from Yunnan Astronomical Observatory for their help on the observation, and also acknowledge the helpful comments and suggestions from the reviewers.

**Appendix**

**Signal-to-Noise Algorithm used in this Paper**

The $S/N$ algorithm we used in this paper is

$$S/N = \frac{\sum_{F_i > 3\sigma} F_i}{\sigma \sqrt{N_{F_i > 3\sigma}}} \quad (A1)$$

where $F_i$ is the power of each bin, $\sigma$ is the variance of the bins without pulse signal, $N_{F_i > 3\sigma}$ is the bin number of pulse signal.

**ORCID iDs**

Wei Liu @ https://orcid.org/0000-0002-9131-3664

**References**

Abbott, B. P., Abbott, R., Acernese, F., et al. 2010, *ApJ*, 713, 671

Backer, D. C., Dexter, M. R., Zepka, A., et al. 1997, *PASP*, 109, 61

Backer, D. C., Kulkarni, S. R., Heiles, C., Davis, M., & Goss, W. 1982, *Nature*, 300, 615

Baffa, C. 2014, *SPIE*, 9147, 91479B

Bardsell, B. R., Bailes, M., Barnes, D. G., & Fluke, C. J. 2012, *MNRAS*, 422, 379

Bonazzola, S., & Gourgoulhon, E. 1996, *A&A*, 312, 675

Caraveo, P. A., Bignami, G. F., Deluca, A., Mereghetti, S., & Becker, W. 2003, *Science*, 301, 1345

Clarke, N., D’Addario, L., Navarro, R., & Trinh, J. 2014, *JAI*, 3, 1450004

Deneva, J. 2007, arXiv:0701181

Ford, J. M., Demorest, P., & Ransom, S. 2010, *Proc. SPIE*, 7740, 77400A

Hampson, G., Brown, A., & Vimiera, C. 2008, *Australia Telescope National Facility*, https://www.jb.man.ac.uk/~pulsar/observing/DFB.pdf

Hankins, T. H., 2017, in Proc. IAU (Cambridge: Cambridge Univ. Press), 13, 29

Hankins, T. H., & Rajkowski, J. M. 1987, *RSJ*, 58, 674

Hankins, T. H., & Rickett, B. J. 1975, *Methods in Computational Physics Advances in Research and Applications*, Vol. 14, 55

Hewish, A., Bell, S. J., Pilkington, J. D. H., Scott, P. F., & Collins, R. A. 1968, *Nature*, 217, 709

Hotan, A., Van Straten, W., & Manchester, R. 2004, *PASA*, 21, 302

Karuppusamy, R., Stappers, B., & Van Straten, W. 2008, *PASP*, 120, 191

Kishalay, D., & Yashwant, G., et al. 2016, *Exp. Astron.*, 41, 67

Liu, P., Wang, P., Li, D., et al. 2018, *PABei*, 36, 173

Liu, W., Meng, Q., Han, J.-L., et al. 2017, *Progress in Electromagnetics Research Symp.—Fall (PIERS—FALL)*, 772

Liu, W., Meng, Q., Wang, C., et al. 2021, *JINST*, 16, P08047

Lorimer, D., Esposito, P., Manchester, R., et al. 2015, *MNRAS*, 450, 2185

Lorimer, D. R., & Kramer, M. (ed.) 2012, *Handbook of Pulsar Astronomy* (Cambridge: Cambridge Univ. Press)

Manchester, R. N., Hobbs, G. B., Teoh, A., & Hobbs, M. 2005, *AJ*, 129, 1993

Manchester, R. N., Lyne, A. G., Camilo, F., et al. 2001, *MNRAS*, 328, 17

Parsons, A., Backer, D., Chang, C., et al. 2006, In: Proc. XXXth General Assembly of the International Union of Radio Science, 4

Siemion, A., Von Korff, J., McMahon, P., et al. 2010, *AcAau*, 67, 1342

Stinebring, D. R., Smirnova, T. V., Hankins, T. H., et al. 2000, *ApJ*, 539, 300

Vertatschitsch, L., Primiani, R., Young, A., et al. 2015, *PASP*, 127, 1226

Weinreb, S. 1961, Proc. of the Institute of Radio Engineers, 49, 1099

Wolszczan, A., & Frail, D. A. 1992, *Nature*, 355, 145

Zhu-Xing, Liang., Yi, L., & Weisberg, Joel. M. 2014, *MNRAS*, 439, 3712